MÉTODOS DE OPTIMIZACIÓN

VNiVERSiDAD
D SALAMANCA

José Manuel Gutiérrez

2010

PREFACIO

> "Quid accedere perfecto potest? nihil, aut perfectum non erat, cui accessit".
>
> (Séneca, *Epistolae morales ad Lucilium*, 66)

La presente monografía es una introducción a los Métodos de Optimización, con una selección de material orientada hacia las aplicaciones empresariales. La exposición está imbricada con el desarrollo de los elementos de teoría de la computación necesarios para el desarrollo del aspecto algorítmico de las matemáticas.

El contenido propiamente de optimización se centra en la optimización lineal. El lector puede quizás criticar lo inusual del tratamiento, en un libro introductorio como éste. En el caso del método del símplex, se parte de la sencilla idea de desarrollar el método del símplex como caso particular de los métodos de direcciones de mejora.

El autor quiere agradecer a su colega José Manuel Cascón los comentarios aportados, y su generosa disposición a hacerse cargo de la edición final del texto.

Índice

1. PRIMERA PARTE: NOCIONES SOBRE TEORÍA DE LA COMPUTACIÓN

1. Algunos conceptos básicos de computación

1.1. Algoritmos

La *teoría de la computación* (*TC* en lo sucesivo) recoge los aspectos "avanzados" de la informática. Es una parte de las matemáticas. En las cuestiones de TC que abordaremos, nuestro tratamiento no será matemático (y, por lo tanto, no será riguroso), sino intuitivo.

Cuando se habla en TC, hay que distinguir entre *problema* e *instancia*. Sumar dos números naturales es un problema; sumar el 4 y el 7 es una instancia del anterior problema. Hallar el área de un triángulo es un problema; hallar el área de un triángulo isósceles de base 5 metros y altura 2 metros es una instancia del anterior problema. En los problemas, los datos son genéricos; en sus instancias, los datos son concretos (normalmente, números). En TC se trabaja casi siempre con problemas, no con sus instancias. Por el contexto quedará claro en cada caso cuándo utilizamos el término "problema" en el sentido técnico de la TC que acabamos de considerar, y cuándo lo hacemos en su sentido corriente.

Sea un problema matemático. Un *algoritmo* para este problema es una secuencia de instrucciones que lo resuelve en un número finito de etapas.

Precisemos algunos puntos de la anterior definición:

Secuencia de instrucciones. El algoritmo reduce la resolución de un problema complejo a la de tareas más sencillas (*instrucciones*) que el ejecutor puede llevar a cabo directamente; estas instrucciones han de ser ejecutadas en un cierto orden. La dificultad posible de las instrucciones depende de quién sea el ejecutor (una persona, o una máquina). Se llama *programa* a un algoritmo cuyo ejecutor es un computador.

Resolver el problema significa que, si el problema tiene solución, el algoritmo la encuentra (normalmente al menos una, si existen varias), y si no la tiene, el algoritmo así lo detecta.

El algoritmo ha de terminar tras ejecutar un *número finito* de instrucciones.

Consideremos el algoritmo que todos hemos aprendido de niños para sumar dos números naturales, como por ejemplo:

$$\begin{array}{r} 3\ 5\ 3 \\ +\ 8\ 7\ 2 \\ \hline 12\ 2\ 5 \end{array}$$

En este algorimo se supone que el ejecutor sabe ejecutar directamete las dos siguientes instrucciones:

1. Sumar dígitos $(0, 1, ..., 9)$
2. Sumar una unidad a un número natural del 10 al 18.

1.2. Arquitectura de un ordenador

Los primeros ordenadores se construyeron por los británicos durante la segunda guerra mundial, para descifrar las claves secretas alemanas. Fueron los sucesivos *Colossus,* concebidos desde el principio como máquinas matemáticas.

La estructura de un ordenador es la llamada *arquitectura de von Neumann*. El ordenador consta de cuatro partes, llamadas *unidades*:

Unidad de memoria, que almacena información. Hay dos tipos de memoria:

- *Memoria RAM* (random access memory), cuyo contenido puede ser alterado durante la ejecución del programa. Es a este tipo de memoria al que nos referiremos en general.

- *Memoria ROM* (read only memory) y memoria externa, cuyos contenido no puede ser alterados durante la ejecución del programa. Los contenidos de la memoria ROM no puede ser cambiados por el usuario (al menos fácilmente); ejemplo de memoria externa es el disco duro.

Unidad aritmético-lógica (o, simplemente, *unidad aritmética*). Es una calculadora rápida, que realiza las cuatro operaciones fundamentales, y operaciones lógicas (resolución de tablas veritativas).

Unidad de control, que ordena al resto de las unidades la ejecución de cada una de las tareas que resultan del programa, en la secuencia correcta. Como veremos, puede tomar *decisiones sencillas*.

Periféricos, que permiten al ordenador comunicarse con el exterior. Pueden ser:

- *de entrada*, para introducir información en el ordenador (v.g., el teclado, la conexión con la red);

- *de salida*, para sacar información del ordenador (v.g., la pantalla, la impresora, la conexión con la red).

1.3. Lenguajes

Los *lenguajes informáticos* nos permiten comunicarnos con el ordenador. Son lenguajes *formalizados* (las matemáticas se escriben también, en principio, en un lenguaje formalizado). Hay dos tipos de lenguajes informáticos:

Lenguajes de bajo nivel. Son los que están "cerca de la máquina" (metafóricamente, "abajo"). El lenguaje interno de la máquina es el *lenguaje máquina*, formado por dígitos binarios; el *lenguaje ensamblador* es una adaptación del lenguaje máquina que resulta un poco más cómoda para los seres humanos.

Lenguajes de alto nivel. Son los que están "cerca del usuario humano de la máquina" (metafóricamente, "arriba"). Están formados por palabras inglesas y símbolos matemáticos. Son de uso relativamente cómodo para los seres humanos. Se agrupan en familias (FORTRAN, COBOL, PASCAL, BASIC,...), cada una de las cuales cuenta con muchas versiones, que van cambiando con los años.

A nivel académico, se utiliza cada vez más el *pseudocódigo*. Es un lenguaje semi-formalizado. Una vez que está escrito un programa en pseudocódigo, el paso a un lenguaje informático concreto es una operación puramente mecánica.

Ante un problema, el usuario de un ordenador tiene en principio dos opciones para disponer de un programa que lo resuelva: (1) comprarlo (normalmente dentro de un grupo de programas, llamado *paquete*); (2) escribirlo él mismo. Aparte de las consideraciones de costo, la primera opción tiene las ventajas de la comodidad y la calidad de un producto hecho por profesionales, y la segunda la de la adaptación a las necesidades específicas del usuario. Combinando las ventajas de ambos, una tercera posibilidad es el uso de un entorno de cálculo (v.g., Matlab, Mathematica). Un *entorno de cálculo* dispone de una colección de programas junto a una opción de programación, que permite adaptar estos programas a las necesidades del usuario, además de construir otros nuevos.

2. Programación

2.1. Instrucciones

Vamos a introducir en lo que sigue las nociones básicas de programación, en pseudocódigo.

Un programa consta de una serie de instrucciones. Según el lenguaje que consideremos, el orden de ejecución de estas instrucciones puede ser:

(a) El orden en que se encuentran escritas (*modo secuencial*).

(b) Cada instrucción lleva una etiqueta (un número natural no nulo), y, después de un espacio en blanco, aparece la instrucción propiamente dicha. Entonces las instrucciones se ejecutan en orden creciente de sus etiquetas.

Hoy en día, la posibilidad (a) se ha impuesto. Sin embargo, se admite la posibilidad de que alguna instrucción lleve etiqueta; ya veremos cómo se utiliza.

En una instrucción pueden aparecer los siguientes elementos:

1. *Constantes*. Son de dos tipos:

- *Constantes numéricas*, que son simplemente números en notación decimal (con "punto" para marcar el inicio de la parte decimal). Cada ordenador trabaja internamente con cierto número de decimales, que se puede duplicar, triplicar,... (*doble precisión*, *triple precisión*,...).

- *Constantes alfanuméricas*, que son secuencias de dígitos, letras (del alfabeto latino) y signos matemáticos. Ya veremos su uso.

Cuando en lo sucesivo hablemos de "constantes", normalmente nos referiremos a "constantes numéricas".

2. *Variables*. Se trabaja con ellas como se hace en matemáticas ordinarias. Cada variable se identifica mediante su nombre (v.g., x). Las reglas para que una expresión sea aceptable como nombre de variable dependen de cada lenguaje, pero las siguientes son normas generales: (1) sólo podemos usar dígitos y letras del alfabeto latino; (2) el primer símbolo ha de ser una letra del alfabeto latino; (3) los nombres de variable no pueden ser *palabras reservadas* (palabras que aparecen en el lenguaje con un significado asignado: normalmente nombres de instrucciones, que veremos enseguida).

Por ejemplo, serían nombres de variable aceptables:

x

$z123$

$inventariodemercancias$

No lo serían, en cambio:

$23z$

print

3. *Signos matemáticos*. Corresponden a los tres tipos de operadores habituales en matemáticas (aritméticos, lógicos y relacionales). Son los mismos que utilizamos habitualmente en matemáticas: $+,-,=,<,\leq$ ó $<=,>$... Para el producto se suele utilizar $*$, para la división $/$, para la potenciación $**$ ó $\uparrow$, para la negación $\sim$, para la disyunción lógica $|$, para la conjunción lógica $\&$, y para "distinto de" $<>$. Las reglas de aplicación de los paréntesis son las habituales.

Así, por ejemplo,

$$\frac{(x+5)^2}{2}$$

se escribiría

$$(\mathrm{x}+5) * *2/2$$

4. *Proposiciones*. Como sabemos, intuitivamente, una proposición es un enunciado del que se puede decir inequívocamente si es verdadero o es falso. En el ordenador sólo se aceptan proposiciones matemáticas escritas de manera formalizada; en los lenguajes informáticos habituales, no se pueden escribir cuantificadores ($\forall$, $\exists$). Cuando se ejecuta un programa, al ordenador solo le interesa el valor lógico de las proposiciones (esto es, si son verdaderas o falsas).

5. *Nombres de instrucciones*. En programación hay muchas instrucciones distintas. Cada una tiene una *sintaxis* estricta, esto es, se ha de escribir con ciertos símbolos en cierto orden. En los lenguajes de alto nivel, para identificar cada instrucción se utilizan en su sintaxis determinadas palabras inglesas (v.g., "print"): son los nombres de instrucciones.

Vamos ahora a introducir las principales instrucciones sencillas. Las agruparemos en tres clases:

(a) Instrucciones de asignación

(b) Instrucciones de control, que a su vez pueden ser:

- Instrucciones de secuenciación
- Instrucciones alternativas
- Instrucciones repetitivas

(c) Instrucciones de entrada y salida

2.2. Instrucciones de asignación

Instrucción (básica) de asignación (:=)

Sintaxis

x:=y

donde:

x es un nombre de variable

y es una constante, o una variable de valor conocido, o una combinación de las anteriores (constantes y variables de valor conocido).

Descripción

La instrucción de asignación realiza dos funciones:

(1) Define la variable x si no estuviera ya definida, esto es, le asigna un registro en memoria, dotado con una dirección propia. En lo sucesivo, los valores almacenados en ese registro serán los de la variable x.

(2) Le asigna a la variable x el valor actual de y.

Ejemplos

Suponemos que las siguientes instrucciones se realizan sucesivamente:

x:=7

x:=x+1

y:=x**2

y:=x+y

El valor final sería: x=8, y=72. Si ahora (no habiendo en el programa previamente a las anteriores otras instrucciones de asignación) escribiéramos la instrucción

y:=z**2

ésta sería errónea, al no estar previamente definida la variable z.

Nota

Una sintaxis alternativa es de la forma:

x←y

Así, por ejemplo,

x←7

2.3. Instrucciones de secuenciación

Estas instrucciones le indican al ordenador qué instrucción ha de ejecutar a continuación, o si no ha de ejecutar ninguna.

Instrucción (básica) de secuenciación (;)

Sintaxis

A; B

donde:

A y B son instrucciones.

Descripción

Tras ejecutarse la instrucción A, la instrucción ";" señala que ahora se pasa a ejecutar la instrucción escrita a su derecha (esto es, B).

Ejemplo

Considérese el siguiente trozo de programa:

x:=7;
x:=x+1; y:=
x**2; y:=x+y

Nótese que el cambio de renglón es irrelevante en la escritura del programa.

Instrucción **go to**

Sintaxis

go to n

(el espacio en blanco entre **go** y **to** es opcional)

donde:

n es la etiqueta de otra instrucción.

Descripción

Se *transfiere control* a la instrucción que tiene la etiqueta n (esto es, se pasa a ejecutarla). La instrucción n puede estar antes o después de la **go to**.

Ejemplo

x:=7;
y:=x**2;
go to 7;
x:=x+1;
7 y:=x+y;
y:=y**2

Si no hubiera ninguna instrucción en el programa con la etiqueta "7", se produciría un error.

Instrucción **end**

Sintaxis

end

Descripción

Se detiene y da por terminada definitivamente la ejecución del programa.

Ejemplo

....; **end**;...;**end**

Al menos ha de haber una instrucción **end**, pero puede haber más de una: el programa puede disponer que en determinado punto se tome una decisión, y, dependiendo del valor de ciertas variables, se pueda acabar en uno u otro sitio (v. infra la instrucción **if**).

2.4. Instrucciones alternativas

En las instrucciones alternativas se debe tomar una decisión: dependiendo del valor de verdad de cierta proposición, se hace una u otra cosa.

Instrucción **if**

Sintaxis

if P **then** A [**else** B]

(escribiendo una expresión entre corchetes en la sintaxis se quiere decir que esa expresión es de escritura opcional; si no se escribe, el ordenador supondrá que esa parte de la instrucción tiene cierto contenido *por defecto*)

donde:

P es una proposición

A y B son grupos de instrucciones (una o varias instrucciones, en cada caso).

Descripción

Se analiza si la proposición P es verdadera o falsa. Si es verdadera, se ejecuta el grupo de instrucciones A; si es falsa, se ejecuta el grupo de instrucciones B. Si no se escribe **else** B, no se hace nada si P es falsa (y se pasa a la instrucción siguiente).

Comentario

En esta instrucción el ordenador toma una decisión muy sencilla. Este tipo de decisiones son las únicas que puede tomar, y la capacidad de tomarlas es lo que diferencia esencialmente un ordenador de una calculadora.

Ejemplo 1

x:=3; y:=2;

if x≤3 **then** x:=x**2 **else** x:=5

Resultado: x=9, y=2.

Ejemplo 2

x:=3; y:=2;

if x=7 **then** x:=x**2 **else** x:=5

Resultado: x=5, y=2.

Ejemplo 3

x:=3; y:=2;

if x=7 **then** x:=x**2; z:=8

Resultado: x=3, y=2, z=8.

Nota sobre sintaxis

Para crear un grupo de instrucciones, en la instrucción **if** y en otras, se utilizan las palabras "begin" (para señalar el comienzo del grupo) y "end" (para señalar su terminación). Así pues, las palabras "begin" y "end" actúan aquí con una función similar a la de los paréntesis. No hay que confundir este uso de la palabra "end" con la instrucción **end**.

Ejemplo 4

x:=3; y:=2;

if x≤7 **then** begin x:=x**2; y:=x+y end **else** x:=5

Resultado: x=9, y=11.

Ejemplo 5

x:=1; y:=2;

3 **if** x≤3 **then** begin y:=2*y; x:=x+1; **go to** 3 end

Resultado: x=4, y=16.

En este ejemplo se ha creado un proceso iterativo, y se han llevado a cabo 3 iteraciones. La variable x ha ido cambiando a lo largo del proceso, y nos ha indicado cuando acaba: se le llama *variable contadora*. En las dos primeras instrucciones se *inicializan* las variables que aparecen en el proceso iterativo.

2.5. Instrucciones repetitivas

Las instrucciones repetitivas crean un proceso iterativo.

Instrucción **while**

Sintaxis

while P **do** A

donde:

P es una proposición

A es un grupo de instrucciones.

Descripción

Se analiza si la proposición P es verdadera o falsa. Si es falsa, se da por terminada la instrucción (y se pasa a la instrucción siguiente); si es verdadera, se ejecuta el grupo de instrucciones A, y se vuelve al inicio, considerando si P es verdadera o falsa... Y así sucesivamente.

Comentario

Para que la ejecución de la instrucción termine tras un número finito de iteraciones, debe de haber en el grupo de instrucciones A, suponiendo que la proposición P sea inicialmente verdadera, instrucciones que puedan alterar el valor de verdad de P.

Ejemplo 1

x:=1; y:=2;

while x≤3 **do** begin y:=2∗y; x:=x+1 end

Se hace lo mismo que en el *Ejemplo 5* de la instrucción **if**.

Ejemplo 2

x:=4; y:=2;

while x≤3 **do** begin y:=2∗y; x:=x+1 end

Ejemplo 3

x:=0; y:=2;

while x+y≤25 **do** begin y:=3∗y; x:=x+2 end

Instrucción **for to**

Sintaxis

for i:=s **to** t **do** A

donde:

i es una variable

s y t son constantes, o variables de valor conocido, o combinaciones de las anteriores

A es un grupo de instrucciones.

Descripción

Se asigna a la variable i el valor inicial s. Se analiza si $i \leq t$. Si es falso, se da por terminada la instrucción; si es verdadero, se ejecuta el grupo de instrucciones A y se incrementa el valor de i en una unidad. Se vuelve ahora al inicio, y se analiza si $i \leq t$... Y así sucesivamente.

Esquema (figura).

Comentarios

1. Lo que se puede hacer con la instrucción **for to** también se puede hacer con la **while**, pero no recíprocamente. La **for to** es una instrucción especializada, que sirve para los procesos iterativos más sencillos, en los que una variable contadora (i) varía entre un tope inferior (s) y uno superior (t), incrementándose cada iteración en una unidad.

2. En la instrucción **for to** se realizan en cada iteración dos *operaciones técnicas* (que forman parte de su misma estructura, y no hay que explicitar): la comprobación de que la variable contadora no excede el tope superior (¿es $i \leq t$?), y

el incremento de esta variable en una unidad (i:=i+1). En la instrucción **while** se realiza una operación técnica en cada iteración: la comprobación del valor de verdad de la proposición P.

Ejemplo 1

DATOS: n$\in \mathbb{N}^*$

...

x:=1;

for i:=1 **to** n **do** x:=x$*$i;

...

Los programas se escriben para problemas, con datos genéricos. En cada ejecución se han de introducir los datos concretos mediante una instrucción (que aún no conocemos), pasándose del problema a una instancia suya. Al final, los resultados han se ser comunicados mediante otra instrucción (que tampoco conocemos todavía); en este ejemplo, el resultado vendrá dado por el valor final de la variable x.

¿Qué hace el anterior algoritmo? Ilustremos su funcionamiento con una instancia sencilla, $n = 4$:

$x = 1$

$i = 1$, $x = 1$, $i = 2$

$x = 1 \cdot 2 = 2$, $i = 3$

$x = 1 \cdot 2 \cdot 3 = 6$, $i = 4$

$x = 1 \cdot 2 \cdot 3 \cdot 4 = 24$, $i = 5$

El anterior algoritmo calcula el factorial de n (esto no es una demostración, sino una mera ilustración).

Ejemplo 2

Alteramos el anterior algoritmo para sumar los n primeros números:

DATOS: n$\in \mathbb{N}^*$

...

x:=0;

for i:=1 **to** n **do** x:=x+i;

...

(El elementro neutro de la suma es el cero.)

Ejemplo 3

Reescribimos el Ejemplo 1, utilizando la instrucción **while** en vez de la **for to**:

DATOS: n$\in \mathbb{N}^*$

...

x:=1; i:=1;

while i≤n **do** begin x:=x∗i; i:=i+1 end;

...

Ilústrese su funcionamiento con la instancia $n = 4$. Reescríbase análogamente el Ejemplo 2 utilizando la instrucción **while**.

Ejemplo 4

Considérense ahora los Ejemplos 1 y 3, pero siendo ahora la línea de datos:

DATOS: n∈ $\mathbb{N}$

¿Qué ocurre cuando se ejecuta la instancia $n = 0$? En el Ejemplo 1 se trata del caso $s > t$.

2.6. Instrucciones de entrada y salida

En las instrucciones de entrada se introduce información durante la ejecución del programa (típicamente, datos), y en las de salida se extrae (típicamente, resultados). En pseudocódigo se escriben en lenguaje ordinario.

Nota sobre sintáxis

En pseudocódigo el texto escrito en lenguaje ordinario queda señalado en cursiva (o subrayado, si es manuscrito).

Ejemplo 1

Podemos ya completar el programa del Ejemplo 1 de la subsección anterior:

DATOS: n∈ $\mathbb{N}^*$

Introdúzcase n;
x:=1;
for i:=1 **to** n **do** x:=x∗i;
escríbase x; **end**

2.7. Subrutinas

Una *subrutina* es un programa que forma parte de otro mayor. Típicamente, la subrutina se ha de ejecutar varias veces cuando se ejecuta el programa mayor, y se dispone sintácticamente de manera que haya que escribirla sólo una vez; cada vez que se necesita, se *llama* la subrutina.

Supongamos que durante la ejecución de un programa complejo haya que calcular muchas veces el factorial de un número; el programa para clacular el factorial forma una subrutina a la que denotamos "Jeremías". Esquemáticamente, la disposición sería la siguiente:

.....

.....; **call subroutine** Jeremías; .....
.....; **call subroutine** Jeremías; .....
.....
begin subroutine Jeremías;.....; **end subroutine** Jeremías

Al segmento de programa entre **begin subroutine** y **end subroutine** solo se puede acceder mediante transferencia de control desde una correspondiente instrucción **call subroutine**.

El cálculo de valores de las funciones habituales en matemáticas (aparte de las aritméticas fundamentales) se realiza mediante subrutinas que están en memoria ROM: son las llamadas *funciones de biblioteca*. A cada función se le da un nombre (frecuentemente con tres letras), y se utiliza la notación funcional normal. Por ejemplo:

x:=7;
y:=sin(x);
z:=cos(y**2+int(x))+log(2*x)*tan(x)

2.8. Variables matriciales

El ordenador puede utilizar matrices (en informática se utiliza a veces el término "array" en vez del de "matriz", lo cual es incorrecto en castellano). Pueden ser unidimensionales, bidimensionales (las matrices propiamente dichas) o tridimensionales:

$$a = (a_i)_{i=1,\ldots,n}\ ,\ b = (b_{ij})_{\substack{i=1,\ldots,m\\ j=1,\ldots,n}}\ ,\ c = (c_{ijk})_{\substack{i=1,\ldots,m\\ j=1,\ldots,n\\ k=1,\ldots,s}}$$

En principio, las variables matriciales no se definen automáticamente mediante la instrucción de asignación, la primera vez que aparecen; han de ser definidas, antes de poder ser utilizadas, mediante una instrucción específica, en la que se indican sus dimensiones. Esta instrucción puede tener una forma del tipo:

dim a(n), b(m,n), c(m,n,s)

donde:

m, n y t son constantes (números naturales), o variables de valor conocido.

La instrucción anterior asigna a las variables matriciales los correspondientes registros en memoria, dotados con direcciones propias (para cada variable matricial y sus variables componentes). Una vez definidas, pueden utilizarse tanto las variables matriciales como tales, como sus variables componentes.

En bastantes lenguajes, las variables matriciales no han de ser necesariamente definidas explícitamente (aunque siempre se puede hacer, si se desea). Si no se definen, el ordenador las define asignando valores por defecto a las dimensiones (como, por otra parte, también se hace con las componentes de la matriz).

Ejemplo 1

dim u(2,3), v(2,3), w(2,3);
introdúzcanse v, w;
u:=v+w;
z:=7;
u_{22}:=v_{12} * *2-z+w_{31}

Ejemplo 2

dim u(2,2), v(2,3), w(3,2);
introdúzcanse v, w;
u:=v*w

En pseudocódigo no hace falta escribir explícitamente la instrucción de definición de variables matriciales. Eso sí, exigiremos que, antes de utilizar una variable matricial, se conozcan previamente sus dimensiones.

Ejercicio

Supóngase que en una clase hay m alumnos con n asignaturas cada uno. Escríbase un programa que almacene las notas de los alumnos y calcule el promedio de notas de cada alumno.

Solución

Almacenamos las notas de los alumnos en una matriz $(a_{ij})_{\substack{i=1,\dots,m\\ j=1,\dots,n}}$, donde a_{ij} es la nota del alumno i en la asignatura j.

DATOS: m,n$\in \mathbb{N}^*$, $(a_{ij})_{\substack{i=1,\dots,m\\ j=1,\dots,n}}$
Introdúzcanse m, n, a;
for i:=1 **to** m **do**
    begin
    b_i:=0;
    for j:=1 **to** n **do** b_i:=b_i+a_{ij};
    b_i:=b_i/n;
    escríbase b_i
    end;
end

Nota sobre sintaxis

Al escribirse un programa, normalmente se sangran los grupos de instrucciones, para que sean así fácilmente identificables visualmente.

Ilústrese el funcionamiento del algoritmo en una instancia con dos alumnos y tres asignaturas (notas: 1, 3, 5; 10, 8, 9).

2.9. Ejemplos

Ejemplo 1

Escríbase un programa para hallar el mínimo de n números dados.

Solución

El algoritmo sigue la idea del óptimo provisional, que al final del proceso se convierte en óptimo definitivo (ejemplo: chica que busca marido, y tiene sucesivos novios). En cada iteración se compara el óptimo provisional (novio) con un nuevo candidato, y el mejor de ambos pasa a ser el nuevo óptimo provisional.

DATOS: n$\in \mathbb{N}^*$, $(a_i)_{i=1,\dots,n}$, con n$\geq$2
Introdúzcanse n, a;
b:=a_1;
for i:=2 **to** n **do**
    if b>a_i **then** b:=a_i;
escríbase b;
end

Cuestiones: (1) Ilústrese el funcionamiento del algoritmo con una instancia sencilla. (2) Altérese el anterior algoritmo para que halle el máximo de n números dados. (3) ¿Qué ocurriría en el algoritmo si se hubiera escrito b$\geq a_i$ en vez de b>a_i? Caso de que resolviera el mismo problema, ¿lo haría trabajando más, o menos?

Ejemplo 2

Escríbase un programa para ordenar de menor a mayor n números dados.

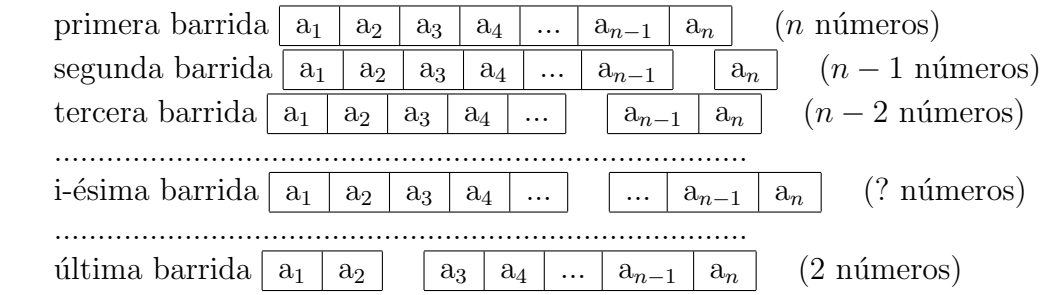
Solución

Hay varias maneras de resolver este problema; lo haremos por el *método de la burbuja* (que, ciertamente, no es el más "eficiente").

El proceso se organiza en una serie de barridas. En la primera barrida se considera todo el vector. Se comparan los dos primeros números; si están en el orden correcto (estamos ordenando de menor a mayor) se dejan como están, y caso contrario se trocan. Después se comparan los números segundo y tercero; si están en el orden correcto se dejan como están, y caso contrario se trocan. Ahora

se hace lo mismo con los números tercero y cuarto..., y así sucesivamente, hasta el final. Al concluirse la primera barrida, el último número está ya en la posición correcta (es el mayor de todos).

En la segunda barrida se hace lo mismo que en la primera, pero ahora considerando solamente los primeros $(n-1)$ números. Al final de la misma, el penúltimo número ya estará en la posición correcta (será el segundo más grande). En la tercera barrida se hace otra vez lo mismo, pero ahora considerando solamente los primeros $(n-2)$ números... En la última barrida se consideran solamente los dos primeros números; si están en el orden correcto se dejan como están, y caso contrario se trocan. Al final de todo el proceso los números quedan ordenados de menor a mayor. Esquemáticamente:



En la i-ésima barrida se consideran $(n-i+1)$ números.
Ilústrese el método de la burbuja con el vector $(6,4,7,1,2,3)$.
Programa:

DATOS: n$\in \mathbb{N}^*$, $(a_i)_{i=1,\ldots,n}$, con n$\geq$2
Introdúzcanse n, a;
for i:=1 **to** n-1 **do**
    for j:=1 **to** n-i **do**
        if $a_j > a_{j+1}$ **then** *tróquense* a_j, a_{j+1};
escríbase a;
end

En general, para trocar los valores de dos variables x e y, se realiza mediante tres instrucciones:
f:=x; x:=y; y:=f
Cuestiones: (1) Ilústrese el funcionamiento del algoritmo con la instancia $(6,4,7,1,2,3)$. (2) Altérese el anterior algoritmo para que ordene de mayor a menor. (3) ¿Qué ocurriría en el algoritmo si se hubiera escrito $a_j \geq a_{j+1}$ en vez de $a_j > a_{j+1}$?

Ejemplo 3

Dados $p, n \in \mathbb{N}^*$, el siguiente programa enumera todas las p-tuplas tomadas de $\overline{n} = \{1, ..., n\}$ en orden lexicográfico:

DATOS: n$\in \mathbb{N}^*$, p$\in \mathbb{N}^*$

introdúzcanse n, p; **for** i:=1 **to** p-1 **do** a_i:=1; a_p:=0; h:=p;
while h>0 **do**
    if a_h<n
        then begin a_h:=a_h+1; h:=p; *escríbase* $(a_1, a_2, ..., a_p)$ end
        else begin a_h:=1; h:=h-1 end;
end

Se empieza por hacer un análisis sintáctico del programa. Aparte de los datos, n y p, las variables son $(a_1, a_2, ..., a_p)$, que representa las sucesivas p-tuplas, y h, la variable que va a gobernar el proceso.

Vamos a ejecutar el programa en el caso de la instancia $n = 5$, $p = 3$. Surgirán a medida que lo hacemos las ideas en que se basa el algoritmo.

$a_1 = 1$	$a_2 = 1$	$a_3 = 0$	$h = 3$	
		$a_3 = 1$		$(1, 1, 1)$
		$a_3 = 2$		$(1, 1, 2)$
		$a_3 = 3$		$(1, 1, 3)$
		$a_3 = 4$		$(1, 1, 4)$
		$a_3 = 5$		$(1, 1, 5)$
		$a_3 = 1$	$h = 2$	
	$a_2 = 2$		$h = 3$	$(1, 2, 1)$
...	...	...	...	...

Cada vez que se da un salto, hay que especificar la acción que se acaba de ejecutar cuando se sigue, y los valores de las variables en ese momento.

$a_1 = 1$	$a_2 = 2$	$a_3 = 5$	$h = 3$	$(1, 2, 5)$
		$a_3 = 1$	$h = 2$	
	$a_2 = 3$		$h = 3$	$(1, 3, 1)$
...	...	...	...	...
$a_1 = 1$	$a_2 = 5$	$a_3 = 5$	$h = 3$	$(1, 5, 5)$
		$a_3 = 1$	$h = 2$	
	$a_2 = 1$		$h = 1$	
$a_1 = 2$			$h = 3$	$(2, 1, 1)$
...	...	...	...	...

La variable h representa la posición en la que estamos trabajando. Así, en la instancia que consideramos ahora, cuando $h = 3$ estamos en la tercera posición de la terna, cuando $h = 2$, en la segunda, y cuando $h = 1$, en la primera. La pregunta: ¿es a_h<n?, significa: ¿podemos crecer todavía en la posición h? Si la respuesta es negativa, se intenta crecer en la posición $(h - 1)$.

$a_1 = 2$	$a_2 = 5$	$a_3 = 5$	$h = 3$	$(2, 5, 5)$
		$a_3 = 1$	$h = 2$	
	$a_2 = 1$		$h = 1$	
$a_1 = 3$			$h = 3$	$(3, 1, 1)$
...	...	...	...	...

Cuando en el orden lexicográfico se retrocede de una posición h, al no poderse crecer más en ella, a una posición anterior, se sabe que en la siguiente tupla el valor en esa posición h que se deja va a ser 1; por ello en el algoritmo se asigna ya el valor $a_h = 1$ cuando se retrocede.

Todo algoritmo ha de finalizar "por si mismo" tras un número finito de etapas. En nuestra instancia:

$a_1 = 5$	$a_2 = 5$	$a_3 = 5$	$h = 3$	$(5, 5, 5)$
		$a_3 = 1$	$h = 2$	
	$a_2 = 1$		$h = 1$	
$a_1 = 1$			$h = 0$	
FIN				

Ejemplo 4

Dados $p, n \in \mathbb{N}^*$, siendo $p \leq n$, consideramos un programa que enumera todas las p-permutaciones tomadas de $\overline{n} = \{1, ..., n\}$ en orden lexicográfico. Para ello ello añadimos un *filtro* al algoritmo anterior. Se generan todas las p-tuplas y se filtran, desechando aquellas que no sean p-permutaciones. En la instancia ejecutada en el ejemplo anterior, todas las ternas entre $(1, 1, 1)$ y $(1, 2, 2)$ serían desechadas, y la primera admitida sería $(1, 2, 3)$. El algoritmo resultante no es particularmente "eficiente":

DATOS: n$\in \mathbb{N}^*$, p$\in \mathbb{N}^*$, con p$\leq$n

introdúzcanse n, p; **for** i:=1 **to** p-1 **do** a_i:=i; a_p:=p-1; h:=p;
while h>0 **do**
    if a_h<n
        then begin a_h:=a_h+1; h:=p; **if** $(a_1, a_2, ..., a_p)$ *es una*

permutación **then** *escríbase* $(a_1, a_2, ..., a_p)$ end
else begin a_h:=1; h:=h-1 end;

end

Obsérvese que la inicialización se ha retocado, de manera que la primera p-tupla propuesta sea ya una p-permutación (en la instancia ejecutada en el ejemplo anterior, sería $(1, 2, 3)$).

El alumno deberá intentar escribir con detalle la "instrucción":

if $(a_1, a_2, ..., a_p)$ *es una permutación* **then** *escríbase* $(a_1, a_2, ..., a_p)$

Resolveremos más tarde esta cuestión.

Ejemplo 5

Dados $p, n \in \mathbb{N}^*$, siendo $p \leq n$, vamos a escribir un programa que enumera todas las p-combinaciones tomadas de $\overline{n} = \{1, ..., n\}$.

En general, por cada p-combinación hay $p!$ p-permutaciones. Por ejemplo, correspondiendo a la 3-combinación $\{2, 3, 5\}$ tenemos seis 3-permutaciones: $(2, 3, 5)$, $(2, 5, 3)$, $(3, 2, 5)$, $(3, 5, 2)$, $(5, 2, 3)$, $(5, 3, 2)$. De todas ellas podemos escoger una, y solo una, que las "represente"; por ejemplo $(2, 3, 5)$. En general, dada una p-combinación $\{a_1, a_2, ..., a_p\}$, habrá una, y una sola, p-permutación $(a_1, a_2, ..., a_p)$ tal que sus componentes estén (estrictamente) ordenados de menor a mayor ("en escalera": $a_1 < a_2 < ... < a_p$). En consecuencia, es lo mismo enumerar p-combinaciones que p-tuplas en escalera. El siguiente algoritmo lo hace (se tienen p-tuplas en escalera si se escriben entre paréntesis, y p-combinaciones si se escriben entre llaves):

DATOS: n$\in \mathbb{N}^*$, p$\in \mathbb{N}^*$, con p$\leq$n

introdúzcanse n, p; **for** i:=1 **to** p-1 **do** a_i:=i; a_p:=p-1; h:=p; a_{p+1}:=n+1;
while h>0 **do**
  if $a_h < a_{h+1} - 1$
  then
    begin
    a_h:=a_h+1; **if** $h < p$ **then**
      begin **for** i:=h **to** p-1 **do** a_{i+1}:=a_i+1; h:=p end;
    escríbase $\{a_1, a_2, ..., a_p\}$
    end
  else h:=h-1;
end

Se empieza por hacer un análisis sintáctico del programa. Aparte de los datos, n y p, las variables $a_1, a_2, ..., a_p$ y h desempeñan análogas funciones a las que

tienen en el Ejemplo 3 (también aquí cabe hablar de "posiciones", aunque se trate de p-combinaciones, considerando la interpretación alternativa como p-tuplas en escalera). La variable $a_p + 1$ toma inicialmente el valor 6, y luego ya no cambia.

Vamos también a ejecutar el programa en el caso de la instancia $n = 5$, $p = 3$, y surgirán a medida que lo hacemos las ideas nuevas que añade este algoritmo a las del Ejemplo 3.

$a_1 = 1$	$a_2 = 2$	$a_3 = 2$	$h = 3$	
		$a_3 = 3$		$\{1, 2, 3\}$
		$a_3 = 4$		$\{1, 2, 4\}$
		$a_3 = 5$		$\{1, 2, 5\}$
			$h = 2$	
	$a_2 = 3$	$a_3 = 4$	$h = 3$	$\{1, 3, 4\}$
		$a_3 = 5$		$\{1, 3, 5\}$
			$h = 2$	
	$a_2 = 4$	$a_3 = 5$	$h = 3$	$\{1, 4, 5\}$
			$h = 2$	
			$h = 1$	
$a_1 = 2$	$a_2 = 3$	$a_3 = 4$	$h = 3$	$\{2, 3, 4\}$
		$a_3 = 5$		$\{2, 3, 5\}$
			$h = 2$	
	$a_2 = 4$	$a_3 = 5$	$h = 3$	$\{2, 4, 5\}$
...	...	...	...	...

La pregunta: ¿es $a_h < a_{h+1} - 1$?, significa, como en el Ejercico 3: ¿podemos crecer todavía en la posición h? Pero ahora la cota máxima de crecimiento no es constante, salvo para la posición p: depende de qué número esté en la posición $h + 1$. Si la respuesta es negativa, se intenta crecer en la posición $(h - 1)$.

$a_1 = 2$	$a_2 = 4$	$a_3 = 5$	$h = 3$	$\{2, 4, 5\}$
			$h = 2$	
			$h = 1$	
$a_1 = 3$	$a_2 = 4$	$a_3 = 5$	$h = 3$	$\{3, 4, 5\}$

¿Qué hace la instrucción **for** i:=h **to** p-1 **do** a_{i+1}:=a_i+1 ? Construir una "escalera", con peldaños de una unidad de altura, a partir del valor actual de a_h, en las posiciones $(h + 1), (h + 2), ..., p$.

$a_1 = 3$	$a_2 = 4$	$a_3 = 5$	$h = 3$	$\{3, 4, 5\}$
			$h = 2$ |	
			$h = 1$ |	
			$h = 0$ |	
FIN				

3. Eficiencia y complejidad

3.1. Eficiencia

Todos tenemos una idea intuitiva de la eficiencia de un algoritmo. Toscamente dicho, un algoritmo será más eficiente que otro cuando resuelve el problema más rápidamente (a igualdad de ordenador). Vamos a precisar algo más este concepto, aunque nuestro enfoque es simplificado e imperfecto.

Sea un algoritmo que resuelve un problema matemático. La *eficiencia* de dicho algoritmo viene dada por el número de operaciones a realizar en el peor de los casos, en función del tamaño de la instancia.

Analicemos algunos puntos de la anterior definición:

Número de operaciones. No es satisfactorio definir "eficiencia" a partir de tiempos de ejecución de los problemas, porque estos tiempos dependen de los ordenadores que se utilicen. En vez de esto se cuenta el número de operaciones. A falta de una definición más perfecta, consideraremos como operaciones las cuatro operaciones aritméticas fundamentales, y la operación de *comparación*. Asumimos implícitamente que todas cuestan el mismo tiempo. No tendremos en cuenta el resto de tareas realizadas por el ordenador . Así, consideramos prácticamente instantánea la instrucción de asignación "pura" (si no contiene operaciones aritméticas). Tampoco tendremos en cuenta las instrucciones de entrada y salida, pese a poder ser bastante lentas, porque se ejecutan pocas veces.

En el peor de los casos. De todas las instancias que puedan surgir, consideraremos la que exija el mayor número de operaciones.

En función del tamaño de la instancia. Los algoritmos se escriben para problemas, con datos genéricos. En general, si la instancia es "más grande", exigirá un mayor número de operaciones. El tamaño de la instancia se mide con un solo número natural $n \in \mathbb{N}^*$. Aunque la determinación matemática del tamaño de la instancia sea difícil, se hace en la práctica una aproximación que surja de manera "natural", y que resulte ser satisfactoria. Así, en todos los problemas de grafos tomaremos como tamaño de la instancia el número de vértices; en el problema

de ordenar n números de menor a mayor, el tamaño de la instancia será n; en el problema de hallar la traza de una matriz cuadrada de dimensiones $n \times n$, será n...

En consecuencia, la eficiencia es una función:

$$\begin{array}{rl} f: & \mathbb{N}^* \to \mathbb{N}^* \\ & n \to f(n) \end{array}$$

Dado un tamaño de la instancia n, considero todas las instancias del problema de tamaño n. Escogemos la peor, es decir, aquella en la que el número de operaciones del algoritmo es máximo; este número de operaciones es $f(n)$.

Ejemplos de eficiencias: $f(n) = n^2 + 2n + 7$, $f(n) = 2^n + n^3 + 2n^2 + 4n$, $f(n) = 5 \cdot 2^n + 6n^3 + n^2$.

Aparte de la eficiencia de un algoritmo, hay que considerar la memoria que exige su ejecución. Aunque es un tema importante, no lo consideraremos ahora.

Eficiencia asintótica.

Consideremos los tiempos de ejecución (en el peor de los casos) en cierto ordenador de dos algoritmos, con eficiencias n^3 y 2^n, para tres tamaños de instancia:

	$n = 20$	$n = 40$	$n = 60$
n^3	0'008 seg.	0'064 seg	0'216 seg.
2^n	1 seg.	12'7 días	366 siglos

Es claro que es crucial el comportamiento de la eficiencia cuando crece el tamaño de la instancia. La *eficiencia asintótica* es una simplificación de la eficiencia que refleja lo que pasa cuando se hace grande el tamaño de la instancia.

En la eficiencia $f(n) = n^2 + 2n + 7$ el término relevante para valores grandes de n es n^2; para $f(n) = 2^n + n^3 + 2n^2 + 4n$ lo es 2^n, y para $5 \cdot 2^n + 6n^3 + n^2$ lo es también 2^n, al ser el factor 5 constante.

Sean

$$\begin{array}{rl} f: & \mathbb{N}^* \to \mathbb{N}^* \\ & n \to f(n) \end{array} \quad , \; \begin{array}{rl} g: & \mathbb{N}^* \to \mathbb{N}^* \\ & n \to g(n) \end{array}$$

Si se cumple que

$$\exists k > 0 \text{ tal que } f(n) \leq k \cdot g(n) \; \forall n \in (\mathbb{N}^* \sim \{n_1, ..., n_p\})$$

se dice que $f(n)$ es del orden de $g(n)$; en símbolos: $f(n)$ es $O(g(n))$.

Si $f(n)$ es la eficiencia de un algoritmo, entonces se dice que su eficiencia asintótica es $O(g(n))$.

Ejemplos

La eficiencia $n^2 + 2n + 7$ es $O(n^2)$. En efecto:

$n^2 + 2n + 7 \leq n^2 + 2n^2 + 7n^2 = 10n^2$

La eficiencia $2^n + n^3 + 2n^2 + 4n$ es $O(2^n)$. En efecto:

$2^n + n^3 + 2n^2 + 4n \leq 2^n + 2^n + 2 \cdot 2^n + 4 \cdot 2^n = 8 \cdot 2^n$, salvo para un número finito de valores de n (compruébese que $n^3 \leq 2^n$ es sólo cierto si se excluye un número finito de casos).

Análogamente $5 \cdot 2^n + 6n^3 + n^2$ es $O(2^n)$.

Es inmediato que $4n^3 + 2n^2 + 3n$ es $O(n^3)$. Por otra parte,

$4n^3 + 2n^2 + 3n \leq 4 \cdot 2^n + 2 \cdot 2^n + 3 \cdot 2^n = 9 \cdot 2^n$, salvo para un número finito de valores de n.

No decimos, sin embargo, que $4n^3 + 2n^2 + 3n$ es $O(2^n)$, sino que es $O(n^3)$, ya que esta segunda acotación es más "ajustada".

A partir de ahora, cuando hablemos de "eficiencia" nos referiremos a la "eficiencia asintótica", a no ser que lo contrario se diga o quede claro por el contexto.

Tipos de eficiencia

Sea un algoritmo con eficiencia $O(g(n))$.

Se dice que el algoritmo es (de eficiencia) polinomial si

$$g(n) = n^p, \text{ con } p > 0$$

Así, por ejemplo, $O(n^3)$ ó $O(n^{2/3})$.

Se dice que el algoritmo es (de eficiencia) exponencial si

$$g(n) = a^n, \text{ con } a > 1$$

Así, por ejemplo, $O(2^n)$ ó $O(\frac{5^n}{2^n})$.

Se dice que el algoritmo es (de eficiencia) logarítmica si

$$g(n) = \log(n)$$

El algoritmo es neperiano, pero es equivalente a los efectos la base que se escoja para el logaritmo (ejercicio). Obviamente, la eficiencia logarítmica es muy buena.

Hay otros tipos eficiencia. Los dos más importantes son la eficiencia polinomial y la exponencial.

Interpretación de la eficiencia

En primer lugar, consideremos la interpretación del tipo de eficiencia. Si un algoritmo es polinomial, podrá en general ser ejecutado si el ordenador es lo suficientemente potente (esto puede ser matizado cuando la eficiencia es un polinomio

de grado alto y el tamaño de la instancia es elevado); así pues, el algoritmo resuelve el problema no solo en teoría, sino también en la práctica. Por otra parte, si el algoritmo es exponencial, la ejecutabilidad del algoritmo dependerá de las particularidades de cada instancia y, en especial, de su tamaño; así pues, no podremos decir que el algoritmo resuelve el problema en la práctica *en general.* Esta última afirmación admite excepciones (la eficiencia considera siempre el caso más pesimista), y hay algoritmos exponenciales que presentan un buen funcionamiento en la práctica; también ingeniosas variantes permiten a veces resolver instancias de tamaños elevados.

En segundo lugar, si tenemos dos algoritmos para el mismo problema, uno de ellos con eficiencia $O(n^2)$ y el otro con eficiencia $O(n^3)$, podremos decir en general que el primero será preferible al segundo; también aquí puede haber excepciones.

Estructuras de datos

Dado un algoritmo, su *estructura de datos* es el sistema mediante el cual se almacenan y organizan los datos del problema que resuelve.

Así por ejemplo, en un algoritmo para grafos, habrá estructuras de datos diferentes según que el grafo se trate a partir de su definición, su matriz de adyacencia o su matriz de incidencia.

Cuando se da un algoritmo en pseudocódigo, no se detalla muchas veces la estructura de datos que se utiliza.

¿Puede cambiar la estructura de datos la eficiencia de un algoritmo? Sí, e incluso, si se utiliza una estructura de datos particularmente desafortunada, puede llegar a quedar afectada la eficiencia asintótica.

Ejemplo 1

Hallar la eficiencia y la eficiencia asintótica del algoritmo visto antes para calcular el factorial de un número natural n. (El tamaño de la instancia es n.)

Esquema del algoritmo (la instrucción **for to** crea un *bucle*):

$\left\lceil \begin{array}{l} {\scriptstyle i=1,\ldots,n} \\ \\ \\ \text{1 multiplicación} \\ \text{2 operaciones técnicas} \end{array} \right.$

Recordemos que en cada iteración de una **for to** se realizan dos *operaciones técnicas* (la comparación al principio de la que hubiera sido iteración $(n+1)$ se desprecia habitualmente).

Eficiencia:

$f(n) = n[1+2] = 3n$

Eficiencia asintótica: $O(n)$ (es la llamada *eficiencia lineal*).

Ejemplo 2

Hallar la eficiencia y la eficiencia asintótica del algoritmo visto antes para almacenar las notas de los alumnos y calcular el promedio de sus notas, habiendo m alumnos con n asignaturas cada uno.

Supondremos que $n \leq m$, y así el tamaño de la instancia será m.

Esquema del algoritmo:

$\lceil$ i=1,...,m
$\quad\lceil$ j=1,...,n
$\quad\quad$
$\quad\quad$
$\quad\quad$ 1 suma
$\quad\quad$ 2 operaciones técnicas
$\quad$ 1 división
$\quad$ 2 operaciones técnicas

Aquí tenemos un bucle dentro de otro bucle (un *nido*).

Eficiencia:

$f(m) = m[n(1+2)+1+2] = 3m[n+1] \leq 3m(m+1) = 3m^2 + 3m$

Eficiencia asintótica: $O(m^2)$.

Ejemplo 3

Hallar la eficiencia y la eficiencia asintótica del algoritmo visto antes para calcular el mínimo de n números dados. (El tamaño de la instancia es n.)

Recordemos que $n \geq 2$.

Esquema del algoritmo:

$\lceil$ i=2,...,n
$\quad$
$\quad$
$\quad$ 1 comparación
$\quad$ 2 operaciones técnicas

Eficiencia:

$f(n) = (n-1)[1+2] = 3n-3$

Eficiencia asintótica: $O(n)$.

Ejemplo 4

Hallar la eficiencia y la eficiencia asintótica del algoritmo visto antes para ordenar de menor a mayor n números dados. (El tamaño de la instancia es n.)

Esquema del algoritmo:

$\lceil$ i=1,...,$(n-1)$

  $\lceil$ j=1,...,$(n-i)$

    1 comparación

    3 sumas

    2 operaciones técnicas

  $\rfloor$

  2 operaciones técnicas

$\rfloor$

Hemos supuesto que la "instrucción informal" *tróquense* a_j, a_{j+1} se realiza mediante las tres instrucciones:

$$\text{f:=a}_j\text{; a}_j\text{=a}_{j+1}\text{; a}_{j+1}\text{=f}$$

(Con el cambio de variable $t := j + 1$, y escribiendo después a_t en vez de a_{j+1},se hubiera podido pasar de las tres sumas del bucle interior a solo una.)

Eficiencia:

Vamos primero a hacerlo incorrectamente:

$f(n) = (n-1)[(n-i)(1+3+2)+2]$

Observemos que esta expresión no solo depende de n, sino también de i, lo cual es imposible. Lo que está entre corchetes es efectivamente el número de operaciones que se realizan en la i-ésima vuelta del bucle exterior. Pero este número no es constante para todas las vueltas: depende de i, o sea, de la vuelta en que estemos. Así que la expresión correcta es:

$f(n) = \sum_{i=1}^{n-1}[(n-i)(1+3+2)+2]$

(Al no ser todos los sumandos iguales, no se ha podido reducir el sumatorio a un producto). Operando ahora:

$f(n) = \sum_{i=1}^{n-1}[6(n-i)+2] = 6\sum_{i=1}^{n-1} n - 6\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 2$

$= 6n(n-1) - 6(n(n-1)/2) + 2(n-1) = 3n(n-1) + 2(n-1) = 3n^2 - n - 2$

Eficiencia asintótica: $O(n^2)$.

Ejemplo 5

En el programa ya visto para enumerar todas las p-permutaciones tomadas de $\overline{n}$ en orden lexicográfico, quedó pendiente detallar la instrucción informal

if $(a_1, a_2, ..., a_p)$ *es una permutación* **then** *escríbase* $(a_1, a_2, ..., a_p)$

Vamos ahora a escribir el correspondiente subprograma, y a hallar su eficiencia y eficiencia asintótica. Partimos de una p-tupla $(a_1, a_2, ..., a_p)$.

Definimos una variable dicotómica v, que tomará el valor 0 si no hay repeticiones en la tupla, y el valor 1 si las hay. El proceso de comprobación se organiza

en una serie de barridas. En la primera barrida, para ver si hay alguna repetición, se compara a_1 con las componentes que le siguen: primero con a_2, después con a_3,..., y por último con a_p. En la segunda barrida se compara a_2 sucesivamente con a_3, a_4,...,a_p. En general, en la *í*-ésima barrida se compara a_i sucesivamente con a_{i+1}, a_{i+2},...,a_p. Por último, en la última barrida se compara a_{p-1} con a_p.

Subprograma:

```
v:=0;
for i:=1 to p-1 do
    for j:=i+1 to p do
        if a_i=a_j then v:=1;
if v=0 then escríbase (a_1, a_2, ..., a_p)
```

Para calcular la eficiencia de este algoritmo (el tamaño de la instancia es p), consideramos su esquema:

```
⌈ i=1,...,(p−1)
|   ⌈ j=(i+1),...,p
|   |
|   |  1 comparación
|   ⌊  2 operaciones técnicas
⌊ 2 operaciones técnicas
1 comparación
```

Eficiencia:

$f(p) = \left(\sum_{i=1}^{p-1}[(p-i)(1+2)+2]\right)+1$

$= \left(\sum_{i=1}^{p-1}[3(p-i)+2]\right)+1 = \left(3\sum_{i=1}^{p-1}p - 3\sum_{i=1}^{p-1}i + \sum_{i=1}^{p-1}2\right)+1$

$= (3p(p-1) - 3(p(p-1)/2) + 2(p-1)) + 1 = ((3/2)p^2 + p/2 - 2) + 1.$

Eficiencia asintótica: $O(p^2)$.

Ejemplo 6

El algoritmo del ejemplo 5 produce cierta insatisfacción. Supóngase que $n = 9$, $p = 7$, y se considera la tupla $(2, 2, 3, 4, 5, 6, 7)$. Ya en la primera vuelta del bucle exterior, y dentro de ella en la primera vuelta del bucle interior, se hace $v = 1$. Este valor de v ya no va a variar, y sin embargo el algoritmo, en vez de desechar inmediatamente esta tupla, sigue trabajando con ella hasta completar todas las comparaciones posibles entre parejas de sus componentes.Una manera de mejorar el algoritmo sería añadir una etiqueta a la instrucción **while**, por ejemplo

7 **while** h>0 **do**

y luego sustituir la instrucción informal

if $(a_1, a_2, ..., a_p)$ *es una permutación* **then** *escríbase* $(a_1, a_2, ..., a_p)$

por el subprograma:

for i:=1 **to** p-1 **do**
  for j:=i+1 **to** p **do**
    if a_i=a_j **then** goto 7;
escríbase $(a_1, a_2, ..., a_p)$

Esquema de este algoritmo:

$\lceil$ i=1,...,(p−1)
  $\lceil$ j=(i+1),...,p
    1 comparación
    2 operaciones técnicas
  $\lfloor$
  2 operaciones técnicas
$\lfloor$

Es el mismo esquema que el del algoritmo del Ejemplo 5, salvo por la última operación. En consecuencia, su eficiencia es:

$f(p) = \sum_{i=1}^{p-1}[(p-i)(1+2)+2] = (3/2)p^2 + p/2 - 2$

El algoritmo del Ejemplo 5 y éste tienen la misma eficiencia (la diferencia en una unidad es irrelevante). ¿Cómo es esto posible, si el segundo es intuitivamente mejor? Consideremos, para $n = 9$ y $p = 7$, la tupla $(1, 2, 3, 4, 5, 6, 6)$. En esta instancia, la "peor posible", los dos algoritmos realizan el mismo número de operaciones. El concepto de eficiencia contempla el peor caso posible. Así, aunque el algoritmo del Ejemplo 5 es peor que el del Ejemplo 6 para casi todas las instancias, produce el mismo número de operaciones en la instancia peor.

Hay conceptos alternativos de "eficiencia promedio" que intentan solucionar esta cuestión. Sin embargo, adolecen de la necesidad de definir lo que es una "instancia promedio", la cual puede no coincidir con lo que para nosotros sea el caso promedio. Aunque estos conceptos de eficiencia promedio pueden ser útiles, no eliminan el papel central del concepto habitual de eficiencia.

3.2. Complejidad

El concepto de eficiencia se aplica a algoritmos; el concepto de complejidad se aplica a problemas (aunque a veces, confusamente, se utilizan en la literatura los términos "complejidad de algoritmos" y "complejidad de problemas"). La *complejidad* de un problema va a dar una "medida" de su *dificultad*; esto es, del grado en que se puede resolver con algoritmos eficientes.

Aunque el estudio de la complejidad, como el de la eficiencia, es matemáticamente exigente, vamos a dar un tratamiento muy informal (y tosco).

Si para cierto problema conocemos un algoritmo polinomial que lo resuelve, lo podemos calificar (aunque sea asumiendo una actitud de optimismo) como problema "fácil". Sin embargo, si para otro problema no se conoce un algoritmo polinomial que lo resuelve, ¿lo podremos catalogar como "difícil"? Como máximo, lo podríamos catalogar como "provisionalmente difícil", ya que ese algoritmo polinomial podría ser descubierto mañana; obviamente, en matemáticas no interesan las afirmaciones "provisionales". El enfoque que vamos a seguir es el de agrupar los problemas en familias de problemas de similar nivel de dificultad, a las que se llama *clases*.

Hemos de empezar por considerar dos tipos de problemas: *problemas decisionales* y *problemas no decisionales*. Son problemas decisionales aquellos cuya solución es un sí o un no (v.g.: averiguar si un grafo es acíclico). El resto de los problemas son no decisionales. A estos últimos pertenecen los *problemas de optimización* y los *problemas de evaluación* (en estos no se requiere hallar soluciones óptimas, sino el valor óptimo). Hay que empezar por abordar la complejidad de los problemas decisionales.

Complejidad de los problemas decisionales

La mayoría de los problemas decisionales que nos interesan se pueden formular de manera que pertenezcan a la llamada *clase NP* de problemas decisionales. Nos restringiremos a los problemas de esta clase, en cuya definición no entraremos.

La *clase P* de problemas decisionales está formada por aquellos problemas para los que existe un algoritmo polinomial que los resuelve. La clase P está incluida en la clase NP (evítese la tentación de leer "NP" como "no polinomial), lo cual se expresa brevemente:

$$P \subseteq NP$$

El problema más importante de la teoría de la computación, aún no resuelto, es:

$$\text{¿es } P = NP \text{ ó } P \subsetneq NP?$$

En un intento de evaluar la dificultad de los problemas de la clase NP para los que no se ha encontrado hasta ahora un algoritmo polinomial, se introduce la clase de los NP-completos. La *clase de los problemas NP-completos* está formada por aquellos problemas de la clase NP que cumplen la siguiente condición: si se encontrara un algoritmo polinomial para alguno de los problemas NP-completos, entonces existirían algoritmos polinomiales para todos los problemas de la clase NP (y, en consecuencia, se tendría que $P = NP$).

Así se puede decir que los problemas NP-completos "dominan" en cuanto a nivel de dificultad a los problemas de la clase NP. Pese a los grandes esfuerzos realizados, no se ha encontrado hasta ahora un algoritmo polinomial para los problemas NP-completos; muchos sospechan que no existe. En todo caso, tampoco se ha demostrado hasta ahora que sea imposible resolver mediante algoritmos polinomiales los problemas NP-completos, y por tanto el problema $P = NP$ sigue abierto.

Gran parte de los problemas decisionales relevantes (de la clase NP) han sido clasificados, adscribiéndose a la clase P ó a la de los problemas NP-completos.

Complejidad de los problemas no decisionales

A cada problema de optimización se le puede asignar un problema decisional asociado (en realidad, una familia de ellos, porque aparece un parámetro adicional). De manera análoga ocurre con los problemas de evaluación. Estos problemas decisionales asociados son relevantes a la hora de estudiar la complejidad de los problemas originales. Un problema de optimización, o de evaluación, es al menos tan difícil como el correspondiente problema decisional asociado.

Sea un problema que no esté necesariamente en la clase NP de problemas decisionales. Se dice que el problema es *NP-duro* si se cumple la siguiente condición: si se encontrara un algoritmo polinomial que lo resolviera, entonces $P = NP$. No se descarta que, suponiendo que $P = NP$, haya problemas NP-duros para los que no existan algoritmos polinomiales.

Toscamente dicho, los problemas NP-duros son al menos tan difíciles como los NP-completos. Se verifica que es NP-duro todo problema de optimización cuyo problema decisional asociado es NP-completo.

Aplicación

Si tenemos un problema clasificado como NP-completo (o NP-duro), resulta imposible para nosotros, a efectos prácticos, disponer de un algoritmo polinomi-

al que lo resuelva. En consecuencia, si no nos funcionan los algoritmos *exactos* disponibles, nos tendremos que conformar con algún algoritmo *heurístico* (esto es, que da una solución aproximada) que sí nos funcione.

4. Ejercicios sobre computación

1. Considérese el problema de calcular el valor de un polinomio de grado n en un punto x:

$$P(x) = \sum_{i=0}^{n} a_i \cdot x^i$$

Para este problema consideramos el siguiente algoritmo:

DATOS: n$\in \mathbb{N}^*$, $(a_i)_{i=0,1,\ldots,n}$, x$\in \mathbb{R}$
Introdúzcanse n, a, x; v:=a_0;
for i:=1 **to** n **do**
    begin
    y:=1;
    for j:=1 **to** i **do** y:=y$*$x;
    v:=v+$a_i$$*$y
    end;
escríbase v; **end**

Hállense la eficiencia y la eficiencia asintótica del algoritmo (el tamaño de la instancia es n, el grado del polinomio).

SOLUCIÓN

Eficiencia: $(3/2)n^2 + (11/2)n$. Eficiencia asintótica: $O(n^2)$

2. Para el problema anterior, se tiene la fórmula:

$$P(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + x \cdot (... + x \cdot a_n)...))$$

A partir de ella, consideramos el siguiente algoritmo:

DATOS: n$\in \mathbb{N}^*$, $(a_i)_{i=0,1,\ldots,n}$, x$\in \mathbb{R}$
Introdúzcanse n, a, x; v:=a_n;
for i:=1 **to** n **do**
    v:=v$*$x+a_{n-i} ;
escríbase v; **end**

Hállense la eficiencia y la eficiencia asintótica del algoritmo.

SOLUCIÓN

Eficiencia: $5n$. Eficiencia asintótica: $O(n)$

NOTA

En los dos algoritmos anteriores hemos supuesto que la operación de potenciación, $x**i$, se realiza meramente a través de i multiplicaciones. En la práctica, los computadores utilizan subrutinas más eficientes para calcular x^i, cuya eficiencia asintótica es $O(\log i)$, para $i \geq 2$.

3. Considérese el siguiente algoritmo:

DATOS: n$\in \mathbb{N}^*$, $(a_{ij})_{i=1,\ldots,n;j=1,\ldots,n}$
Introdúzcanse n, a; s:=0;
for i:=1 **to** n **do** s:=s+a_{ii};
escríbase s; **end**

¿Qué problema resuelve este algoritmo?

4. Encuéntrese una función $g(n)$ adecuada, tal que $f(n)$ sea $O(g(n))$, en cada uno de estos casos:

(a) $f(n) = \binom{n}{3}$
(b) $f(n) = \frac{5n^3+6}{n+2}$
(c) $f(n) = \frac{(n^2+7)3^n}{2^n}$
(d) $f(n) = n + \log(n)$

5. Sea un algoritmo para resolver cierto problema, siendo n el tamaño de la instancia. Si, en el peor de los casos, el algoritmo desarrolla n iteraciones, y la iteración i-ésima require $(i + 5)$ operaciones, ¿cuál es su eficiencia?

6. Cuando se dice que una expresión $f(n)$ es $O(\log_a n)$, ¿por qué es innecesario especificar la base a del logaritmo?

SOLUCIÓN

¿Qué relación hay entre $\log_a n$ y $\log_b n$ para dos bases distintas, a y b?

7. Escríbase un algoritmo para hallar las raíces reales de una ecuación de segundo grado.

SOLUCIÓN

Hay que discutir primero cuándo existen raíces reales y cuándo no.

8. Escríbase un algoritmo para hallar los cuadrados de todos los números naturales impares menores o iguales que $n \in \mathbb{N}^*$. Hállense la eficiencia y la eficiencia asintótica del algoritmo.

SOLUCIÓN

DATOS: n$\in \mathbb{N}^*$
Introdúzcase n; i:=1;
while i$\leq$n **do**
begin
y:=i$*$i;
escríbase y;
i:=i+2
end;
end

9. Escríbase un algoritmo que halle todos los números naturales de tres cifras tales que sean iguales a la suma de los cubos de las tres cifras.

10. Escríbase un algoritmo que calcule el número combinatorio $\binom{n}{m}$, donde $n, m \in \mathbb{N}^*$, $n \geq m$.

11. Una bañera tiene un grifo y un sumidero. La bañera tiene una capacidad de l litros, el grifo echa g litros por minuto y el sumidero desagua s litros por minuto, donde $g \neq s$. Si la operación empezó cuando la bañera tenía l_0 litros, $l_0 < l$, escríbase un algoritmo que calcule cuánto tarda en vaciarse o llenarse, según el caso; hágase, no resolviendo el problema mediante ecuaciones, sino viendo de minuto en minuto el estado de la bañera. Supóngase que $l, l_0, g, s \in \mathbb{N}^*$.

SOLUCIÓN

Hay que discutir primero cuándo la bañera se va a vaciar o a llenar.

12. La raíz cuadrada (positiva) de un número $q \in \mathbb{N}^*$ puede ser aproximada mediante la sucesión:

$$\begin{aligned} x_1 &:= 1 \\ x_{n+1} &:= \frac{x_n + q/x_n}{2}, \text{ si } n \in \mathbb{N}^* \end{aligned}$$

Se considera que un término de la sucesión $(x_n)_{n=1}^{\infty}$ es una buena aproximación de $\sqrt{q}$ cuando $|x_n - x_{n-1}| < 10^{-4}$. Escríbase un algoritmo que calcule de esta forma el valor aproximado de la raíz cuadrada de q.

NOTA

Obsérvese que, tal como se ha definido el problema, el algoritmo termina en un número finito de etapas.

2. SEGUNDA PARTE: MÉTODOS DE MEJORA

1. Generalidades sobre optimización

1.1. Introducción

Mínimos y máximos

Sea D un conjunto, y sea $f : D \to \mathbb{R}$ una función. Se dice que $\overline{x} \in D$ es un mínimo de f si

$$f(\overline{x}) \leq f(x),\ \forall x \in D$$

y que es un máximo de f si

$$f(\overline{x}) \geq f(x),\ \forall x \in D$$

Si $D \subseteq \mathbb{R}^n$, se dice que $\overline{x} \in D$ es un mínimo local de f si

$$\exists \varepsilon > 0 \text{ tal que } f(\overline{x}) \leq f(x),\ \forall x \in D \cap B_\varepsilon(\overline{x})$$

y que es un máximo local de f si

$$\exists \varepsilon > 0 \text{ tal que } f(\overline{x}) \geq f(x),\ \forall x \in D \cap B_\varepsilon(\overline{x})$$

Problemas de optimización

Consideramos el problema de optimización:

$$\begin{array}{lll} \text{(P)} & Min & f(x) \\ & s.a & x \in S \\ & & [x \in X] \end{array}$$

donde:

$f : X \to \mathbb{R}$
$S \subseteq X$

En él aparecen tres elementos:

- El conjunto de referencia X, que es el ámbito en el que trabajamos (a nivel elemental será normalmente $\mathbb{R}^n$ o un conjunto finito)

- El conjunto factible S, que incluye los puntos que, en principio, son aceptables como solución del problema. Sus elementos se llaman *soluciones factibles* (un término desafortunado, pero consagrado por el uso).

- La función objetivo f, que mide el menor o mayor valor que para nosotros tienen los puntos factibles.

Son soluciones (óptimas) del problema (P) los mínimos de la función restringida $f_{|S}$. Las soluciones óptimas forman el conjunto óptimo:

$$\mathcal{O} := \{z \in S : f(z) \leq f(x) \ \forall x \in S\}$$

Son soluciones (óptimas) locales de (P) los mínimos locales de la función restringida $f_{|S}$.

Si $S = \emptyset$ se dice que el problema es no factible; entonces $\mathcal{O} = \emptyset$, y el problema no tiene solución. Si $\mathcal{O} \neq \emptyset$, se define el *valor óptimo* como el valor de la función objetivo sobre cualquier solución óptima; obsérvese que el valor óptimo es único.

Alternativamente, el problema de optimización se puede formular como problema de maximización. Para pasar de un problema de maximización a uno de minimización, o viceversa, basta cambiar el signo de la función objetivo.

El conjunto factible S viene frecuentemente dado implícitamente mediante condiciones impuestas a los elementos de X. En particular, el conjunto factible puede ser $S := \{x \in X : g_i(x) \leq 0 \ , \ i \in I\}$, donde $g_i : X \to \mathbb{R} \ , \ i \in I$. En los casos que nosotros vamos a considerar, $I := \{1, ..., m\}$, y el problema de optimización se escribe:

$$\begin{array}{lll} \text{(PR)} & Min & f(x) \\ & s.a & g_i(x) \leq 0 \ , \ i = 1, ..., m \\ & & [x \in X] \end{array}$$

donde:
$f : X \to \mathbb{R}$
$g_i : X \to \mathbb{R} \ , \ i = 1, ..., m$

Dado que las restricciones de tipo "$\geq$" se pueden escribir con "$\leq$", y que una restricción de igualdad se puede escribir mediante dos de desigualdad, el problema con restricciones (PR) tiene gran generalidad.

Como ejemplo de (PR), con $X = \mathbb{R}^3$, tenemos:

$$\begin{array}{lllll} Min & -x_1^2 + 2x_2 & & & \\ & x_1 & -4x_2 & +x_3 & \leq 1 \\ s.a & -3x_1 & +2x_2 & & = 6 \end{array}$$

Un tipo especial del problema (P) se da cuando X es un conjunto finito: es el *problema de optimización combinatorial*. El *problema de optimización lineal* aparece cuando en el problema (PR) se tiene que $X = \mathbb{R}^n$ y las funciones que aparecen (f, $g_i, i = 1, ..., m$) son afines (lineales, en sentido amplio). Si $X = \mathbb{Z}^n$ y las funciones que aparecen son afines, se tiene el *problema de optimización lineal entera*.

1.2. Resultados generales

Existencia

Sea D un conjunto finito y no vacío, y sea $f : D \to \mathbb{R}$ una función. Entonces existen mínimo y máximo de f. Esto se prueba inmediatamente considerando en $\mathbb{R}$ el conjunto finito y no vacío $\{f(x) : x \in D\}$. El siguiente importante resultado generaliza la anterior observación:

Teorema de Weierstrass. Sea D un conjunto compacto y no vacío, y sea $f : D \to \mathbb{R}$ continua. Entonces existen mínimo y máximo de f.

En el caso elemental $D \subseteq \mathbb{R}^n$, "compacto" equivale a "cerrado y acotado".

Unicidad

Teorema. Sea $D \subseteq \mathbb{R}^n$ convexo, y sea $f : D \to \mathbb{R}$ estrictamente convexa (cóncava). Entonces, si existe mínimo (máximo) de f, es único.

Este resultado se satisface también si "estrictamente convexa" se sustituye por "estrictamente cóncava" y "mínimo" se sustituye por "máximo".

Local y global

Todo óptimo (global) es local. En presencia de convexidad se cumple el recíproco:

Teorema. Sea $D \subseteq \mathbb{R}^n$ convexo, y sea $f : D \to \mathbb{R}$ convexa. Dado $\overline{x} \in D$, se tiene:

$$\overline{x} \text{ es mínimo de } f \text{ si, y sólo si, } \overline{x} \text{ es mínimo local de } f$$

Este resultado se satisface también si se sustituye: "convexa" por "cóncava", "mínimo" por "máximo" y "mínimo local" por "máximo local".

2. Métodos de mejora

2.1. Métodos de mejora

Sea el problema (P).

Rasgos principales de los métodos de mejora

1. Son métodos iterativos tales que *en cada iteración se dispone de una solución factible* ("óptimo provisional"). Así, en la iteración h-ésima se tiene $y^h \in S$.

2. El método está dotado con un *criterio de optimalidad*, esto es, una condición suficiente de optimalidad. Así, en la iteración h-ésima, y^h se somete al criterio de optimalidad; si se satisface el criterio, y^h es solución óptima, y se finaliza; caso contrario, se continúa.

3. Se procede entonces a la *mejora*: se halla $y^{h+1} \in S$ tal que $f(y^{h+1}) \leq f(y^h)$. (Se distingue entre "mejorar" ($\leq$) y "mejorar estrictamente" ($<$).)

Comentario

El que se disponga en cada iteración de una solución factible tiene la ventaja de que, si el método se interrumpe antes de encontrar una solución óptima, al menos se obtiene una solución factible que puede estar bastante cerca del óptimo. Por otra parte, tiene el inconveniente de que hay que inicializar con una solución óptima, lo cual puede ser difícil a veces.

Esquema general de los métodos de mejora

Definimos los métodos de mejora mediante el siguiente esquema (expresado no en pseudocódigo, sino mediante pasos):

PASO 1 (inicialización)

Elíjase $y^1 \in S$;

$h \leftarrow 1$

PASO 2 (criterio de optimalidad)

Si se satisface el criterio de optimalidad: y^h solución óptima; FIN

PASO 3 (mejora)

Elíjase $y^{h+1} \in S$ tal que $f(y^{h+1}) \leq f(y^h)$;

$h \leftarrow h + 1$;

váyase al paso 2

El esquema anterior parte de que el problema tiene solución óptima. Si el problema no tiene por qué ser factible, hay que incorporar en el paso 1 un detector de problemas no factibles (a veces la elección de solución factible inicial o, en su caso, la detección de no factibilidad, se realizan mediante un algoritmo previo). Si el problema, aun siendo factible, no tiene por qué tener solución óptima, hay que incorporar en el paso 3 un detector de problemas sin solución.

Para obtener a partir del esquema general un método concreto, hay que determinar:

1. ¿Cómo se elige la solución factible inicial (y cómo se detecta que no existe, si el problema no es factible)?
2. ¿Qué criterio de optimalidad se utiliza?
3. ¿Cómo se mejora (o sea, cómo se elije la nueva y^{h+1})?
4. Si el problema es factible, pero no tiene por qué tener solución óptima, ¿qué detector de problemas sin solución se utiliza?

2.2. Algoritmo de mejora simple

Sea el problema (P), en el que $X = S = \{x_1, x_2, ..., x_n\}$, con $n \in \mathbb{N}^*$. (El conjunto factible es finito y está enumerado.) Este problema tiene siempre solución.

Definición

El algoritmo sigue la idea del óptimo provisional, que al final del proceso se convierte en óptimo definitivo (ejemplo: chica que busca marido, y tiene sucesivos novios). En cada iteración se compara el óptimo provisional (novio) con un nuevo candidato, y el mejor de ambos pasa a ser el nuevo óptimo provisional.

PASO 1 (inicialización)

$y^1 \leftarrow x_1$;

$h \leftarrow 1$

PASO 2 (criterio de optimalidad)

Si $h = n$: y^h solución óptima; FIN

PASO 3 (mejora)

$$y^{h+1} \leftarrow \begin{cases} y^h \ , \text{ si } f(y^h) \leq f(x_{h+1}) \\ x_{h+1}, \text{ si } f(y^h) > f(x_{h+1}) \end{cases}$$

$h \leftarrow h + 1$;

váyase al paso 2

Ejercicio 1. Escríbase el algoritmo anterior en pseudocódigo.
Solución
Una primera versión sería la siguiente:

DATOS: n$\in \mathbb{N}^*$, $(x_i)_{i=1,\ldots,n}$
Introdúzcanse n, x;
y$\leftarrow x_1$;
h$\leftarrow$1;
while h<n **do**
begin **if** f(y)>f(x_{h+1}) **then** y$\leftarrow x_{h+1}$; h$\leftarrow$h+1 end
escríbase y;
end

En este caso el algoritmo se puede reescribir de manera más compacta:

DATOS: n$\in \mathbb{N}^*$, $(x_i)_{i=1,\ldots,n}$
Introdúzcanse n, x;
y:=x_1;
for h:=1 **to** n-1 **do**

if f(y)>f(x_{h+1}) **then** y←x_{h+1};
escríbase y;
end

Ejercicio 2. El problema de hallar el mínimo de n números dados es un caso particular del problema general anterior (¿cuál es ahora la función objetivo?). Particularícese el algoritmo escrito al final del Ejercicio 1 para este caso, y compárese con el obtenido en el Tema 2 (¿qué ha pasado con la variable contadora?).

Ejercicio 3. Escríbase mediante pasos el algoritmo obtenido en el Tema 2 para el problema de hallar el mínimo de n números dados, en vez de en pseudocódigo.

Reglas de parada

El inconveniente del algoritmo de mejora simple es que el criterio de optimalidad consiste simplemente en constatar el agotamiento de todo el conjunto factible. Se pueden a veces considerar criterios adicionales de optimalidad (esto es, condiciones suficientes de optimalidad), llamadas *reglas de parada*. Así, por ejemplo, la chica que busca marido puede considerar que un caballero que tenga las características de ser listo, fiel y buen cocinero, es óptimo. Con ello, si apareciera un individuo con tales características, se le declararía óptimo y se pararía el proceso de búsqueda, sin analizar los candidatos todavía pendientes de examen.

2.3. Métodos de direcciones de mejora

Sea el problema (P), en el que $X \subseteq \mathbb{R}^n$.

Direcciones de mejora

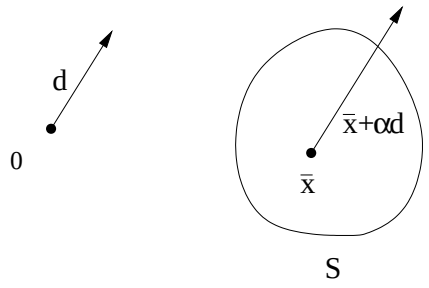
En un espacio vectorial, una dirección es un vector no nulo. (Denotamos por $\mathbb{R}^n_*$ el conjunto de vectores no nulos de $\mathbb{R}^n$.)

Sean $\overline{x} \in X$ y $d \in \mathbb{R}^n_*$. Se dice que d es una dirección de mejora en $\overline{x}$ si cumple:

$$\exists \alpha > 0 \Big/ \begin{array}{c} (\overline{x} + \alpha d) \in X \\ f(\overline{x} + \alpha d) \leq f(\overline{x}) \end{array}$$

Sean $\overline{x} \in X$ y $d \in \mathbb{R}^n_*$. Se dice que d es una dirección factible de mejora en $\overline{x}$ si cumple:

$$\exists \alpha > 0 \Big/ \begin{array}{c} (\overline{x} + \alpha d) \in S \\ f(\overline{x} + \alpha d) \leq f(\overline{x}) \end{array}$$



Definición de método de direcciones de mejora

Recordemos que en la mejora de los métodos de mejora se pasa de una solución factible a otra mejor:

$$y^h \rightsquigarrow y^{h+1} \in S / f(y^{h+1}) \leq f(y^h)$$

Los métodos de direcciones de mejora son métodos de mejora en los que la mejora se realiza de una manera especial. A saber, el proceso para elegir y^{h+1} se realiza en dos pasos:

1. Se elije una dirección de mejora d^h en y^h.
2. Se elige cuánto se avanza en esa dirección, esto es, se elige $\alpha_h \geq 0$ y se hace

$$y^{h+1} := y^h + \alpha_h d^h \tag{2.1}$$

Puesta en práctica

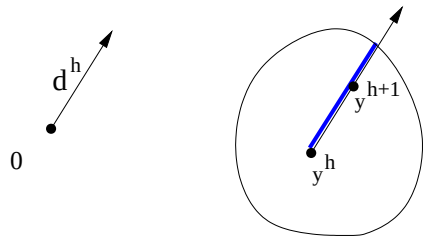
La elección de d^h depende de cada método de direcciones de mejora concreto. En cambio, la elección de α_h viene dada por la resolución, exacta o aproximada, del siguiente problema de optimización:

$$\begin{array}{ll} Min & f(y^h + \alpha d^h) \\ s.a & y^h + \alpha d^h \in S \\ & \alpha \geq 0 \end{array} \tag{2.2}$$

Es un problema unidimensional cuya variable es α. Si este problema (que siempre es factible: considérese $\alpha = 0$) no tiene solución óptima, el problema (P) no tiene tampoco solución óptima (es éste normalmente el "detector de problemas sin solución" que se utiliza en el método; en particular, lo es en el método del

símplex, que veremos enseguida). Si (2.2) tiene solución óptima, sea α_h una de sus soluciones óptimas.

A través de (2.2) se determina un punto óptimo de entre aquellos que sean factibles de la semirrecta que parte de y^h en la dirección de mejora previamente elegida.



¿Qué ocurre si d^h, siendo como es dirección de mejora, no es dirección factible de mejora en y^h? Entonces no hay en la semirrecta que parte de y^h en la dirección de mejora ningún punto distinto de y^h que mejore el valor en y^h. En consecuencia, resolviendo (2.2), resulta $\alpha_h = 0$, y nos quedamos en el punto en el que estamos: $y^{h+1} = y^h$. (Ya consideraremos más adelante este caso en el algoritmo del símplex.)

Comentario

Los métodos de direcciones de mejora permiten reducir la resolución de un problema de optimización multidimensioal (con varias variables) a la resolución sucesiva de problemas de optimización unidimensionales (con una variable), más sencillos en principio de resolver.

3. El algoritmo del símplex

El algoritmo del símplex permite resolver "eficientemente" (ya precisaremos esto) cualquier problema de optimización lineal. Es el método de optimizacióm más utilizado en la práctica empresarial.

3.1. El problema de optimización lineal

Repaso de álgebra

Sea una aplicación lineal $p : \mathbb{R}^n \to \mathbb{R}$. Entonces

$$p(x) = (v_1, v_2, ..., v_n) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = v_1x_1 + v_2x_2 + ... + v_nx_n$$

donde $(v_1, v_2, ..., v_n)$ es la matriz en las bases canónicas de p (de dimensiones $1 \times n$).

Notación y terminología

Como es habitual, consideraremos que todos los vectores de $\mathbb{R}^n$ se representan mediante vectores columna, salvo advertencia en contrario.

Convenimos en llamar a un número α *positivo* si es $\alpha \geq 0$, y *estrictamente positivo* si es $\alpha > 0$. (Esta terminología no es la más frecuente, ya que se llama habitualmente *positivo* a un número si es estrictamente mayor que cero.)

El problema de optimización lineal en forma general

Tal como lo hemos definido arriba, el problema de optimización lineal es:

$$\begin{array}{ll} Min & a_1x_1 + a_2x_2 + ... + a_nx_n \\ s.a & c_{11}x_1 + c_{12}x_2 + ... + c_{1n}x_n - d_1 \leq 0 \\ & c_{21}x_1 + c_{22}x_2 + ... + c_{2n}x_n - d_2 \leq 0 \\ & ..... \\ & c_{m1}x_1 + c_{m2}x_2 + ... + c_{mn}x_n - d_m \leq 0 \\ & [x \in \mathbb{R}^n] \end{array}$$

La función objetivo se puede considerar que es una función lineal (en sentido estricto), ya que la constante de la función afín puede ser eliminada. En forma matricial queda

$$\begin{array}{ll} Min & ax \\ s.a & Cx \leq d \\ & [x \in \mathbb{R}^n] \end{array} \tag{3.1}$$

donde:

$a = (a_1, a_2, ..., a_n)$ es el vector de costes

$C = (c_{ij})_{\substack{i=1,\dots,m \\ j=1,\dots,n}}$ es la matriz del sistema de desigualdades

$d = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \end{pmatrix}$ es el vector de términos independientes

$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ es el vector de variables

(Convenimos en que todos los vectores de costes se representarán mediante matrices fila.)

Diremos que (3.1) es la forma general del problema de optimización lineal.

Formas del problema de optimización lineal

El problema de optimización lineal aparece en la práctica en diferentes *formas*. Las principales son:

Forma general		Forma canónica		Forma estándar	
Min	ax	Min	ax	Min	ax
$s.a$	$Cx \leq d$	$s.a$	$Cx \leq d$	$s.a$	$Cx = d$
	$[x \in \mathbb{R}^n]$		$x \geq 0$		$x \geq 0$
			$[x \in \mathbb{R}^n]$		$[x \in \mathbb{R}^n]$

Evidentemente, la forma canónica es un caso particular de la forma general (considérese la restricción $-Ix \leq 0$, donde I es la matriz identidad $n \times n$), y la forma estándar un caso particular de la forma canónica.

Paso a la forma estándar

El método del símplex habitual necesita que el problema esté expresado en foma estándar. A pesar de que la forma estándar es sólo un caso particular de la forma general, podemos reducir la resolución de un problema en forma general a la de otro en forma estándar, en el sentido que precisaremos enseguida.

Para realizar esta reducción a la forma estándar, procedemos en dos etapas:

1. Paso de la forma general a la forma canonica. Si tenemos una variable x_j irrestringida en signo, la descomponemos en la diferencia de dos variables positivas:

$$x_j = x'_j - x''_j, \text{ con } x'_j \geq 0, x''_j \geq 0$$

y reemplazamos x_j allí donde aparezca por $(x'_j - x''_j)$.

(La idea subyacente es que todo número real se puede poner como diferencia de dos números positivos, aunque ciertamente no de forma única.)

2. Paso de la forma canónica a la forma estándar. Introducimos para cada desigualdad

$$c_{i1}x_1 + c_{i2}x_2 + ... + c_{in}x_n \leq d_i$$

una *variable de holgura* $s_i \geq 0$, y reemplazamos la desigualdad por la igualdad

$$c_{i1}x_1 + c_{i2}x_2 + ... + c_{in}x_n + s_i = d_i$$

(Si la desigualdad es del tipo $\geq$, la variable de holgura aparecerá restando.)

Vamos a ilustrar el procedimiento mediante un ejemplo. Sea el problema de optimización lineal:

$$\begin{array}{lll} Max & z_1 - 4z_2 & \\ s.a & -3z_1 + z_2 & \leq 9 \\ & z_1 + 2z_2 & \leq -1 \\ & z_2 \geq 0 & \\ & [(z_1, z_{2)} \in \mathbb{R}^2] & \end{array} \tag{3.2}$$

Como labor previa, lo escribimos en la forma general, convirtiéndolo en un problema de minimización:

$$\begin{array}{lll} Min & -z_1 + 4z_2 & \\ s.a & -3z_1 + z_2 & \leq 9 \\ & z_1 + 2z_2 & \leq -1 \\ & z_2 \geq 0 & \end{array}$$

Haciendo el cambio $z_1 = z'_1 - z''_1$, lo pasamos a la forma canónica:

$$\begin{array}{lll} Min & -z'_1 + z''_1 + 4z_2 & \\ s.a & -3z'_1 + 3z''_1 + z_2 & \leq 9 \\ & z'_1 - z''_1 + 2z_2 & \leq -1 \\ & z'_1, z''_1, z_2 \geq 0 & \end{array}$$

Introduciendo las variables de holgura s_1 y s_2, lo transformamos a la forma estándar:

$$\begin{array}{lll} Min & -z'_1 + z''_1 + 4z_2 & \\ s.a & -3z'_1 + 3z''_1 + z_2 + s_1 & = 9 \\ & z'_1 - z''_1 + 2z_2 + s_2 & = -1 \\ & z'_1, z''_1, z_2, s_1, s_2 \geq 0 & \\ & [(z'_1, z''_1, z_2, s_1, s_2) \in \mathbb{R}^5] & \end{array} \tag{3.3}$$

Si queremos podemos, por razones estéticas, rebautizar las variables: $x_1 = z_1'$, $x_2 = z_1''$, $x_3 = z_2$, $x_4 = s_1$, $x_5 = s_2$.

¿Es (3.2) "el mismo" problema que (3.3)?

En general, se dice que dos problemas de optimización son *equivalentes* si sus conjuntos optimos coinciden. ¿Son (3.2) y (3.3) equivalentes? No. Si llamamos $\mathcal{O}_1$ al conjunto óptimo de (3.2) y $\mathcal{O}_2$ al de (3.3), es imposible que $\mathcal{O}_1 = \mathcal{O}_2$, puesto que $\mathcal{O}_1 \subseteq \mathbb{R}^2$ y $\mathcal{O}_2 \subseteq \mathbb{R}^5$.

Sin embargo, supongamos que obtenemos la siguiente solución optima de (3.3):

$$\begin{array}{ll} z_1' & \longrightarrow \\ z_1'' & \longrightarrow \\ z_2 & \longrightarrow \\ s_1 & \longrightarrow \\ s_2 & \longrightarrow \end{array} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 6 \\ 0 \end{pmatrix}$$

(Se ha señalado el lugar que ocupa en el vector cada variable del problema de optimización.) Vamos a hallar una solución óptima del problema (3.2); ésta tendrá dos componentes, correspondientes a las variables z_1 y z_2. Recordando que $z_1 = z_1' - z_1''$, y olvidándonos de las variables de holgura, se obtiene

$$\begin{array}{ll} z_1 & \longrightarrow \\ z_2 & \longrightarrow \end{array} \begin{pmatrix} 0-1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

que efectivamente es una solución óptima de (3.2).

El procedimiento anterior funciona siempre. Se puede demostrar que, si partimos de un problema original en forma general, y lo transformamos en un problema en forma estándar mediante las dos etapas señaladas arriba, entonces toda solución óptima del problema transformado se convierte en una solucón óptima del problema original deshaciendo los cambios de variable y eliminando las variables de holgura, tal como se ha ilustrado en el ejemplo.

En suma, podemos reducir la resolución de problemas en forma general a la resolución de problemas en forma estándar. Así pues, basta saber resolver problemas en forma estándar para lograr resolver también problemas en forma general.

Ejercicio

Sea el problema de optimización lineal:

$$\begin{array}{lll} Min & z_1 - 2z_2 & \\ s.a & z_1 + 7z_2 & \leq 5 \\ & 5z_1 - 2z_2 & = 2 \\ & [(z_1, z_{2)} \in \mathbb{R}^2] & \end{array}$$

Transfórmese a la forma estándar.

Problema de partida para el método del símplex

En todo lo que sigue sobre el método del símplex, partimos de un problema de optimización lineal en forma estándar:

$$\begin{array}{lll} \text{(PL)} & Min & cx \\ & s.a & Ax = b \\ & & x \geq 0 \\ & & [x \in \mathbb{R}^n] \end{array}$$

donde:

$c = (c_1, c_2, ..., c_n)$ es el vector de costes

$A = (a_{ij})_{\substack{i=1,...,m \\ j=1,...,n}}$ es la matriz del sistema

$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$ es el vector de términos independientes

$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ es el vector de variables

Asumiremos que (PL) satisface dos hipótesis:

(1) $m < n$

(2) $rg(A) = m$

El conjunto factible es $S := \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. Las restricciones que lo definen son, por un lado, $Ax = b$ (llamado simplemente "el sistema"), y, por otro, $x \geq 0$ (restricciones de positividad).

Las dos hipótesis anteriores (1) y (2) se refieren al sistema $Ax = b$, y se puede demostrar fácilmente que equivalen a las dos siguientes:

(i) En el sistema no hay ecuaciones redundantes

(ii) El sistema es compatible, y tiene más de una solución

Tanto (i) como (ii) parecen hipótesis asequibles (si hay ecuaciones redundantes, se suprimen; si el sistema no es compatible, el problema de optimización es no factible; si el sistema tiene una sola solución, o bien el problema de optimización

es no factible, o bien S se reduce a un solo punto, y no hay mucho que optimizar). Sin embargo, ya veremos más adelante que las cosas no son tan sencillas.

3.2. Preliminares

Base

El concepto de *base de una matriz* está muy relacionado con el de base de un espacio vectorial. A la hora de definir esta última, se considera a veces que los vectores del sistema vienen dados en cierto orden, de manera que cambia la base si cambia el orden en que son dados sus elementos; nosotros siempre consideraremos "base" de un espacio vectorial en el sentido de "base ordenada".

Puesto que $rg(A) = m$ y $m < n$, tenemos, por un lado, que todas las filas de la matriz A son linealmente independientes, y , por otro, que en ella existen m columnas linealmente independientes. La columnas de A son vectores de $\mathbb{R}^m$.

Se llama *base de* A (o, simplemente, *base*) a una base de $\mathbb{R}^m$ formada por columnas de A. Así pues, una base es un sistema formado por m columnas linealmente independientes de A (dadas en cierto orden).

Equivalentemente, se considera que una base es una submatriz invertible de A con dimensiones $m \times m$. (La submatriz indica m columnas de A en un cierto orden.) Consideraremos indiferentemente la base como sistema de columnas o como matriz, según nos interese en cada caso.

Consideremos por ejemplo que A es de dimensiones 3×5. Una base estará formada por tres columnas de A. Supongamos que las columnas 1, 4 y 5 son linealmente independientes (etiquetamos las columas según el lugar en que están en la matriz A):

$$A = \left(\boxed{1} \; \boxed{2} \; \boxed{3} \; \boxed{4} \; \boxed{5} \right)$$

y así una base sería:

$$\boxed{1}, \boxed{4}, \boxed{5}$$

y otra distinta:

$$\boxed{1}, \boxed{5}, \boxed{4}$$

Matricialmente la primera base sería:

$$B = \left(\boxed{1} \; \boxed{4} \; \boxed{5} \right)$$

Se llaman *columnas básicas* a las que pertenecen a la base, y *columnas no básicas* a las restantes. En nuestro ejemplo, las columnas 1, 4 y 5 son básicas, y la 2 y la 3 son no básicas. Análogamente se llaman *variables básicas* a las asociadas a las columnas básicas, y *variables no básicas* a las otras. En nuestro ejemplo, x_1, x_4 y x_5 son básicas, y x_2 y x_3 son no básicas. De manera obvia se definen componentes básicas y no básicas de un vector de $\mathbb{R}^n$, etc.

Dada una base, el resto de de las columnas, dadas en cierto orden, forman una *no base*. Como en el caso de las bases, también una no base se puede dar matricialmente. En nuestro ejemplo tomamos

$$N = \left(\boxed{2} \; \boxed{3} \right)$$

Así una no base tiene matricialmente dimensiones $m \times (n - m)$.

Bases y no bases se pueden enunciar de manera compacta dando, en su orden, las etiquetas de las columnas que las forman. En nuestro ejemplo tenemos la base $I = (1, 4, 5)$ y la no base $J = (2, 3)$. Esta manera de describir bases y no bases requiere en el ordenador menos memoria que la descripción matricial.

Notación

En cada momento tendremos en el método del símplex una base y una no base determinadas. Dado entonces un vector con n componentes, formado por variables o números, podremos descomponerlo en dos vectores, que llamaremos su parte básica y su parte no básica. Sea por ejemplo que tenemos la base $I = (1, 4, 5)$ y la no base $J = (2, 3)$. Consideremos el vector de variables:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

Definimos ahora su *parte básica*, con m componentes:

$$x^B = \begin{pmatrix} x_1 \\ x_4 \\ x_5 \end{pmatrix}$$

y su *parte no basica*, con $(n-m)$ componentes:

$$x^N = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}$$

La parte básica x^B de x está formada por las componentes básicas de x en el mismo orden en que están en la base. La parte no básica x^N de x está formada por las componentes no básicas de x en el mismo orden en que están en la no base. Analogamente, si en nuestro caso el vector de costes fuera $c = (1, 1, 13, 10, 0)$, entonces $c^B = (1, 10, 0)$ y $c^N = (1, 13)$.

Ejemplo ilustrativo

Ilustraremos la exposición teórica del método del símplex mediante el siguiente ejemplo. Sea el problema de optimización lineal:

$$\begin{array}{lllllll} Min & x_1 + x_2 + 13x_3 + 10x_4 & & & & & \\ & x_1 & +x_2 & +x_3 & +x_4 & & = 5 \\ s.a & -x_1 & +x_2 & & & +x_5 & = 2 \\ & & -x_2 & +x_3 & -x_4 & & = -5 \\ & x \geq 0 & & & & & \\ & [x \in \mathbb{R}^5] & & & & & \end{array}$$

Se tiene que $m = 3$ y $n = 5$. Las matrices de definen el problema serían, con la notación de (PL):

$c = (1, 1, 13, 10, 0)$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & -1 & 0 \end{pmatrix}$$

$$b = \begin{pmatrix} 5 \\ 2 \\ -5 \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

Determinamos la base $I = (1, 4, 5)$ y la no base $J = (2, 3)$. Matricialmente.

$$B = \begin{pmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}, \; N = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix}$$

Como hemos visto, se tiene ahora que

$$x^B = \begin{pmatrix} x_1 \\ x_4 \\ x_5 \end{pmatrix}, \; x^N = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}$$

Repaso de álgebra

Sea W una matriz de dimensiones $m \times n$ y v un vector columna $n \times 1$. Denotamos $W = (W_1, W_2, ..., W_n)$, donde W_j es la j-ésima columna de W (y, por tanto, $W_j \in \mathbb{R}^m$). Entonces:

$$Wv = v_1 W_1 + v_2 W_2 + ... + v_n W_n$$

Dicho en palabras: el producto de una matriz por un vector (columna) es una combinación lineal de las columnas de la matriz, cuyos coeficientes son las componentes del vector, de manera que cada componente va con su columna correspondiente (la primera componente de v con la primera columna de W, la segunda componente de v con la segunda columna de W,...: "cada oveja con su pareja").

Una fórmula sencilla, pero fundamental

Dada una base B y una no base N, consideramos el sistema

$$Ax = b \tag{3.4}$$

Denotamos mediante $A_1, A_2, ..., A_n$ las columnas de A, mediante $B_1, B_2, ..., B_m$ las columnas de B, y mediante $N_1, N_2, ..., N_{n-m}$ las columnas de N. Aplicando el "repaso de álgebra" anterior,

$$Ax = x_1 A_1 + x_2 A_2 + ... + x_n A_n \tag{3.5}$$

Por otra parte, por la misma razón,

$$Bx^B + Nx^N = x_1^B B_1 + ... + x_m^B B_m + x_1^N N_1 + ... + x_{n-m}^N N_{n-m} \qquad (3.6)$$

Observemos ahora los segundos miembros de (3.5) y (3.6). En el primer caso, aparece una combinación lineal de todas las columnas de A (n sumandos), cuyos coeficientes son las componentes de x (y "cada oveja con su pareja"). En el segundo caso, también aparece una combinación lineal de todas las columnas de A (n sumandos); el coeficiente con el que aparece cada columna es el mismo que en el caso anterior ("cada oveja va con su pareja"). En consecuencia:

$$Ax = Bx^B + Nx^N$$

Hemos obtenido que el sistema (3.4) es equivalente al sistema

$$Bx^B + Nx^N = b$$

en el cual despejamos:

$$Bx^B = b - Nx^N$$
$$x^B = B^{-1}b - B^{-1}Nx^N$$

Hemos obtenido el siguiente resultado:

$$\begin{array}{l} \text{el sistema} \\ Ax = b \\ \text{es equivalente al sistema} \\ x^B = B^{-1}b - B^{-1}Nx^N \end{array} \qquad (3.7)$$

Esto quiere decir que un vector $x \in \mathbb{R}^n$ satisface $Ax = b$ si, y sólo si, $x^B = B^{-1}b - B^{-1}Nx^N$. Obsérvese que en $x^B = B^{-1}b - B^{-1}Nx^N$ las variables básicas están despejadas a partir de las no básicas. Así pues, el que se cumpla el sistema $Ax = b$ equivale a esta dependencia de las variables básicas de las no básicas: las variables no básicas son "libres", pero las básicas son sus "esclavas". (Así que se podría llamar a $x^B = B^{-1}b - B^{-1}Nx^N$ la "ecuación de la esclavitud": las básicas son esclavas de las no básicas.)

Ejemplo

Veamos las fórmulas (3.6) y (3.5) en nuestro ejemplo ilustrativo. Como $I = (1, 4, 5)$ y $J = (2, 3)$, se tiene que $B = (A_1, A_4, A_5)$ y $N = (A_2, A_3)$; así $B_1 = A_1$, $B_2 = A_4$, etc. Luego

$$Bx^B + Nx^N = x_1^B B_1 + x_2^B B_2 + x_3^B B_3 + x_1^N N_1 + x_2^N N_2 = x_1 A_1 + x_4 A_4 + x_5 A_5 + x_2 A_2 + x_3 A_3$$

que es igual al segundo miembro de (3.6) con estos datos.

Reformulación del problema de optimización

Sean una base B y una no base N. Consideremos $x \in \mathbb{R}^n$ tal que $Ax = b$. Vamos a reescribir el valor cx de la función objetivo. Separando la parte básica de la no básica, como antes,

$$cx = c^B x^B + c^N x^N$$

Aplicamos ahora (3.7):

$$cx = c^B(B^{-1}b - B^{-1}Nx^N) + c^N x^N$$

Por la propiedad distributiva,

$$cx = c^B B^{-1}b - c^B B^{-1}Nx^N + c^N x^N$$

$$cx = c^B B^{-1}b + (c^N - c^B B^{-1}N)x^N$$

Hemos obtenido:

$$\begin{array}{l} \text{Si } Ax = b, \\ \text{entonces } cx = c^B B^{-1}b + (c^N - c^B B^{-1}N)x^N \end{array} \tag{3.8}$$

Podemos ahora reformular de manera equivalente el problema (PL), de manera que las componentes básicas estén despejadas a partir de las no básicas:

$$\begin{array}{lll} \text{(PLR)} & Min & c^B B^{-1}b + (c^N - c^B B^{-1}N)x^N \\ & s.a & x^B = B^{-1}b - B^{-1}Nx^N \\ & & B^{-1}b - B^{-1}Nx^N \geq 0 \\ & & x^N \geq 0 \\ & & [x \in \mathbb{R}^n] \end{array}$$

Solución básica

Sea una base B. Una *solución básica* respecto a la base B es un vector $\overline{x} \in \mathbb{R}^n$ que satisface las dos siguientes condiciones:

(1) Satisface el sistema: $A\overline{x} = b$

(2) Todas sus componentes no básicas son nulas: $\overline{x}^N = 0$

En virtud de (3.7), la condición (1) equivale a

(1') $\overline{x}^B = B^{-1}b - B^{-1}N\overline{x}^N$

Si se cumple (2), esto equivale a su vez a

(1") $\overline{x}^B = B^{-1}b$

En consecuencia, las fórmulas $\overline{x}^N = 0$ y $\overline{x}^B = B^{-1}b$ determinan una solución básica, y para toda base existe una, y una sola, solución básica. Podemos pues definir de manera equivalente "solución básica": la *solución básica* respecto a la base B es el vector $\overline{x} \in \mathbb{R}^n$ definido mediante: $\overline{x}^B = B^{-1}b, \overline{x}^N = 0$.

Dada una base B, ¿su solución básica será siempre factible? No. La solución básica $\overline{x}$ respecto a B será factible precisamente cuando $\overline{x}^B \geq 0$. En efecto,

$$\overline{x} \in S \Leftrightarrow \overline{x} \geq 0 \Leftrightarrow \overline{x}^B \geq 0$$

Si la solución básica es factible, se dice que es una *solución factible básica* (y la base correspondiente se dice que es *factible*).

Ejemplo

Vamos a hallar la solución básica en nuestro ejemplo ilustrativo. La base es $I = (1, 4, 5)$. La solución básica viene dada por

$$\overline{x}^B = B^{-1}b = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ -5 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \\ 2 \end{pmatrix}$$

$$\overline{x}^N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

En consecuencia,

$$\overline{x} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \\ 2 \end{pmatrix}$$

Se tiene que $\overline{x}$ es una solución factible básica.

Si hubiéramos considerado la base $I = (1, 2, 3)$, su inversa sería

$$B^{-1} = \begin{pmatrix} 1/3 & -2/3 & -1/3 \\ 1/3 & 1/3 & -1/3 \\ 1/3 & 1/3 & 2/3 \end{pmatrix}$$

con lo que $\overline{x}^B = \begin{pmatrix} 2 \\ 4 \\ -1 \end{pmatrix}$, y la solución básica no sería factible.

3.3. Fundamentos del método del símplex

El método del símplex en su contexto

El método del símplex fue creado por el matemático americano George Dantzig.

Podemos considerar tres rasgos importantes del método del simplex:

(i) Es un método de direcciones de mejora.

(ii) Si el procedimiento no se detiene en una iteración, pasamos en ella de una solución factible básica a otra solución factible básica.

(iii) El criterio de optimalidad del símplex tiene la siguiente forma: en cada iteración se analiza un número finito de candidatas a direcciones de mejora; si ninguna de estas candidatas es efectivamente una dirección de mejora, entonces estamos en un óptimo. (En su momento explicaremos esto.)

Recordando lo que hemos dicho al hablar en general de los métodos de mejora (y de los métodos de direcciones de mejora), para desarrollar el método del símplex habrá que determinar:

(1) ¿Cómo se elige la solución factible básica inicial (y cómo se detecta que no existe, si el problema no es factible)?

(2) ¿Qué criterio de optimalidad se utiliza?

(3) ¿Cómo se hallan las direcciones de mejora?

En el método del símplex, (1) se aborda mediante un algoritmo previo, que comentaremos en su momento. En cuanto a (2) y (3), se les da solución mediante los dos teoremas principales en que se basa el símplex: el *Teorema de direcciones de mejora* (para (3)) y el *Teorema de optimalidad* (para (2)).

Geométricamente, el conjunto factible de (PL) es un poliedro (no necesariamente acotado). Las soluciones factibles básicas corresponden a vértices del poliedro. Las direcciones de mejora nos hacen recorrer aristas del poliedro, a así pasamos a través de una arista de un vértice a otro. No entramos en la justificación rigurosa de estos aspectos geométricos.

Teorema de direcciones de mejora

Idea intuitiva

El Teorema de direcciones de mejora proporciona las direcciones de mejora utilizadas en el algoritmo del simplex.

Sean una base (factible) B, una no base N, y la correspondiente solución factible básica $\overline{x}$. Descomponiendo $\overline{x}$ en su parte básica y en su parte no básica, lo podemos representar gráficamente (con los vectores "tumbados"):

$$\overline{x}^N = (\ 0\ ,\ 0\ ,\ 0\ ,...,\ 0\ ) \qquad\qquad \overline{x}^B = (\ \ ,\ \ ,\ 0\ ,...,\ \ ,\ \ )$$

Todas las componentes no básicas son nulas, y las básicas, al satisfacer $\overline{x}$ el sistema de ecuaciones, vienen dadas por la fórmula $\overline{x}^B = B^{-1}b - B^{-1}N\overline{x}^N$, que en este caso da $\overline{x}^B = B^{-1}b$; obsérvese que todas las componentes son positivas.

Veamos cómo se mejora en el símplex. Al ser un método de direcciones de mejora, se aplica la fórmula general (2.1), y pasamos de la actual solución factible $y^h = \overline{x}$ a una nueva solución factible $y^{h+1} = \overline{x} + \Delta d$, donde por conveniencia hemos denotado $\Delta = \alpha_h$ y $d = d^h$. Consideramos por separado la parte básica y la parte no básica de la nueva solución factible $\overline{x} + \Delta d$. La idea principal es la siguiente. En la parte no básica se dejan todas las componentes como estaban (en su valor cero), salvo una de ellas (digamos que la que ocupa el lugar k-ésimo), que se hace crecer. Gráficamente:

$$(\overline{x} + \Delta d)^N = \quad (\ 0\ ,\ \underset{k}{\uparrow}\ ,\ 0\ ,...,\ 0\ )$$

¿Cómo se expresa esta idea en términos de la dirección de mejora? Se tiene que

$$(\overline{x} + \Delta d)^N = \overline{x}^N + \Delta d^N = \Delta d^N$$

En consecuencia, hacemos d^N igual a un vector que tiene todas sus componentes cero, salvo la k-ésima, que es un 1. Lo denotamos así:

$$d^N = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \llcorner k$$

Obtenemos que

$$(\overline{x} + \Delta d)^N = \begin{pmatrix} 0 \\ \vdots \\ \Delta \ \llcorner k \\ \vdots \\ 0 \end{pmatrix} \tag{3.9}$$

En su momento veremos cómo se determina $\Delta \geq 0$, ó, lo que es lo mismo, cuánto se crece en la componente k-ésima.

¿Qué ocurre con la parte básica $(\overline{x}+\Delta d)^B$ de la nueva solución factible? Puesto que $\overline{x}+\Delta d$ ha de ser factible, cumple el sistema, o, lo que es lo mismo, la ecuación $x^B = B^{-1}b - B^{-1}Nx^N$. Luego $(\overline{x}+\Delta d)^B = B^{-1}b - B^{-1}N(\overline{x}+\Delta d)^N$, y los valores de la parte básica de $\overline{x} + \Delta d$ quedan unívocamente determinados por los de la parte no básica. Ya veremos en la demostración del teorema como ello determina d^B, la parte básica de la dirección de mejora.

Teorema de direcciones de mejora. Considérese el problema (PL). Sean una base B, una no base N, y la correspondiente solución factible básica $\overline{x}$. Sea $k \in \left\{j \in \overline{n-m}/(c^N - c^B B^{-1}N)_j < 0\right\}$. Sea $d \in \mathbb{R}^n$ definida mediante $d^N = \begin{pmatrix} 0 \\ \vdots \\ 1 \ \llcorner k \\ \vdots \\ 0 \end{pmatrix}$, $d^B = -B^{-1}N_k$.

Entonces:

(1) d es una dirección de mejora en $\overline{x}$

(2) $(\overline{x} + \alpha d)$ satisface el sistema $Ax = b$, $\forall \alpha \geq 0$

Comentario

(Recordemos que $(c^N - c^B B^{-1}N)_j$ es la j-ésima componente del vector $(c^N - c^B B^{-1}N)$, y N_k es la k-ésima columna de la matriz N).

¿Cómo selecciono la posición k en que voy a hacer crecer el vector $\overline{x}^N$? Gráfi-

camente, observemos los dos siguientes vectores, ambos con $(n-m)$ componentes:

$$\overline{x}^N = \qquad\qquad \overset{\uparrow}{(\ -\ ,\ -\ ,\ -\ ,...,\ -\ )}$$

$$c^N - c^B B^{-1} N = \quad (\quad ,\quad ,\quad ,...,\quad)$$

Si en el vector de abajo alguna componente k es estrictamente negativa, entonces esa componente se puede hacer crecer en el vector de arriba para mejorar. (Si hay varias estrictamente negativas, vale cualquiera de ellas; en el gráfico se toma $k = 2$).

Demostración del Teorema de direcciones de mejora. Probaremos primero (2), y después (1).

(2) En virtud de (3.7), la pregunta que hemos de responder es:

¿ $(\overline{x} + \alpha d)$ satisface $x^B = B^{-1}b - B^{-1}Nx^N$, $\forall \alpha \geq 0$?

O, lo que es lo mismo,

¿ $(\overline{x} + \alpha d)^B = B^{-1}b - B^{-1}N(\overline{x} + \alpha d)^N$, $\forall \alpha \geq 0$?

O sea,

¿ $\overline{x}^B + \alpha d^B = B^{-1}b - B^{-1}N(\overline{x}^N + \alpha d^N)$, $\forall \alpha \geq 0$?

Puesto que $\overline{x}$ es solución básica, se tiene que $\overline{x}^N = 0$ y $\overline{x}^B = B^{-1}b$, y la línea anterior es equivalente a

¿ $\alpha d^B = -\alpha B^{-1} N d^N$, $\forall \alpha \geq 0$?

Ahora bien, $Nd^N = N_k$, ya que todas las componentes de d^N son ceros, salvo la k-ésima, que es un 1. Así la línea anterior queda

¿ $\alpha d^B = -\alpha B^{-1} N_k$, $\forall \alpha \geq 0$?

Consideramos ahora dos casos:

Caso 1: si $\alpha = 0$. Entonces la igualdad es trivial.

Caso 2: si $\alpha > 0$. Entonces podemos dividir por α en ambos miembros de la igualdad, y resulta

¿ $d^B = -B^{-1}N_k$?

lo cual es cierto por hipótesis.

(1) Recordando la definición de dirección de mejora, la pregunta que hemos de responder ahora es:

¿ $\exists \alpha > 0$ / $c(\overline{x} + \alpha d) \leq c\overline{x}$?

Vamos a demostrar algo más fuerte, a saber:

¿ $c(\overline{x} + \alpha d) \leq c\overline{x}$, $\forall \alpha \geq 0$?

Puesto que $(\overline{x} + \alpha d)$ satisface el sistema (por el apartado (2), ya demostrado), y también lo hace $\overline{x}$, podemos aplicar (3.8) a ambos vectores, resultando:

¿ $c^B B^{-1}b + (c^N - c^B B^{-1}N)(\overline{x} + \alpha d)^N \leq c^B B^{-1}b + (c^N - c^B B^{-1}N)\overline{x}^N$, $\forall \alpha \geq 0$?

O sea,

¿ $(c^N - c^B B^{-1}N)(\overline{x}^N + \alpha d^N) \leq (c^N - c^B B^{-1}N)\overline{x}^N$, $\forall \alpha \geq 0$?

Ya que $\overline{x}^N = 0$, hemos pues de demostrar:

¿ $\alpha(c^N - c^B B^{-1}N)d^N \leq 0$, $\forall \alpha \geq 0$?

En el primer miembro de la desigualdad hay un producto escalar; puesto que todas las componentes de d^N son ceros, salvo la k-ésima, que es un 1, nos queda

¿ $\alpha(c^N - c^B B^{-1}N)_k \leq 0$, $\forall \alpha \geq 0$?

lo cual es cierto, considerando que $(c^N - c^B B^{-1}N)_k < 0$ por hipótesis.

Q.E.D.

Teorema de optimalidad

Teorema de optimalidad. Considérese el problema (PL). Sean una base B, una no base N, y la correspondiente solución factible básica $\overline{x}$. Sea $\left\{ j \in \overline{n-m} / (c^N - c^B B^{-1}N)_j < 0 \right\} = \emptyset$.

Entonces $\overline{x}$ es solución óptima.

Comentario

Recordemos la idea intuitiva sobre cómo se realiza la mejora: se dejan todas las componentes de $\overline{x}^N$ como estaban (en su valor cero), salvo una de ellas, la que ocupa el lugar k-ésimo, que se hace crecer; donde k es una posición del vector $(c^N - c^B B^{-1}N)$ que es estrictamente negativa. Pero, ¿qué ocurre si ninguna posición del vector $(c^N - c^B B^{-1}N)$ es estrictamente negativa? El Teorema de optimalidad nos dice que entonces estamos en un óptimo.

Ahora vemos que se verifica el rasgo (iii) del método del simplex que consideramos al principio de la subsección "Fundamentos del método del símplex", en relación al criterio de optimalidad del método. En cada iteración se analiza un número finito de candidatas a direcciones de mejora: $(n-m)$ candidatas, correspondientes a las $(n-m)$ componentes de $\overline{x}^N$; haciendo crecer cada una de estas componentes, dejando las demás en su valor cero, se obtiene una candidata a dirección de mejora. Si ninguna de estas candidatas es efectivamente una dirección de mejora (para lo cual hace falta que la correspondiente posición del vector $(c^N - c^B B^{-1} N)$ sea estrictamente negativa, según el Teorema de direcciones de mejora), entonces estamos en un óptimo (según el Teorema de optimalidad).

Obsérvese cómo el Teorema de direcciones de mejora y el Teorema de optimalidad "encajan" perfectamente.

Demostración del Teorema de optimalidad.

Obviamente $\left\{ j \in \overline{n-m} / (c^N - c^B B^{-1} N)_j < 0 \right\} = \emptyset$ es equivalente a que

$$(c^N - c^B B^{-1} N) \geq 0$$

Puesto que ya sabemos que $\overline{x}$ es factible, será solución óptima si es mejor que las otras soluciones factibles. La pregunta que hemos de responder es:

¿ $c\overline{x} \leq cx$, $\forall x \in S$?

Puesto que $\overline{x}$ satisface el sistema (al ser factible), y también lo hace $x \in S$, podemos aplicar (3.8) a ambos vectores, resultando:

¿ $c^B B^{-1} b + (c^N - c^B B^{-1} N)\overline{x}^N \leq c^B B^{-1} b + (c^N - c^B B^{-1} N) x^N$, $\forall x \in S$?

Ya que $\overline{x}^N = 0$, hemos pues de demostrar:

¿ $0 \leq (c^N - c^B B^{-1} N) x^N$, $\forall x \in S$?

Ahora bien, todas las componentes de x^N son positivas (al ser x factible), y también son positivas todas las componentes de $(c^N - c^B B^{-1} N)$; en consecuencia el producto escalar de ambos vectores $(c^N - c^B B^{-1} N) x^N$ es un número positivo, y es cierto lo que queríamos ver.

Q.E.D.

3.4. Ejecución de la mejora

Resolución del problema unidimensional

Sean una base B, una no base N, y la correspondiente solución factible básica $\overline{x}$. Supongamos que no se satisface el criterio de optimalidad, e intentamos mejorar. Al ser el símplex un método de direcciones de mejora, se aplica la fórmula general

(2.1), y pasamos de la actual solución factible $y^h = \overline{x}$ a una nueva solución factible $y^{h+1} = \overline{x} + \Delta d$, donde d se obtiene ahora mediante el Teorema de direcciones de mejora:

$$d^N = \begin{pmatrix} 0 \\ \vdots \\ 1 \; \llcorner k \\ \vdots \\ 0 \end{pmatrix}, \; d^B = -B^{-1}N_k$$

siendo $(c^N - c^B B^{-1} N)_k < 0$. En lo sucesivo denotaremos $p := d^N$, $q := d^B$.

¿Cómo determinamos Δ? De acuerdo a la regla general de los métodos de direcciones de mejora, habremos de resolver el problema (2.2). En nuestro caso queda:

$$\begin{array}{ll} Min & c(\overline{x} + \alpha d) \\ s.a & A(\overline{x} + \alpha d) = b \\ & (\overline{x} + \alpha d) \geq 0 \\ & \alpha \geq 0 \end{array} \tag{3.10}$$

que es un problema unidimensional cuya variable es α.

Para resolver (3.10), vamos a hallar un problema equivalente extremadamente simple, y cuya resolución resultará trivial.

Empezaremos por simplificar la función objetivo de (3.10). Para cualquier α factible de (3.10), se tiene que $A(\overline{x} + \alpha d) = b$, y podemos aplicar (3.8) a $\overline{x} + \alpha d$, resultando la función objetivo:

$Min \qquad c^B B^{-1} b + (c^N - c^B B^{-1} N)(\overline{x} + \alpha d)^N$

Eliminando la constante $c^B B^{-1} b$ de la función objetivo, y recordando que $p := d^N$, queda

$Min \qquad (c^N - c^B B^{-1} N)(\overline{x}^N + \alpha p)$

Ya que $\overline{x}^N = 0$, resulta

$Min \qquad \alpha (c^N - c^B B^{-1} N) p$

Puesto que todas las componentes de p son ceros, salvo la k-ésima, que es un 1, nos queda la función objetivo

$Min \qquad \alpha (c^N - c^B B^{-1} N)_k$

Cambiando de signo, pasamos a un problema de maximización:

$Max \qquad [-(c^N - c^B B^{-1} N)_k] \alpha$

Ahora bien, $[-(c^N - c^B B^{-1} N)_k]$ es una constante estrictamente positiva, y podemos dividir por ella la función objetivo, que así resulta

$Max \qquad \alpha$

Consideramos ahora las restricciones de (3.10). Sea $\alpha \geq 0$. En virtud del Teorema de direcciones de mejora, apartado (2), se tiene que la restricción $A(\overline{x} + \alpha d) = b$ se cumple siempre, y no hace falta pues exigirla. Por otra parte, la restricción $(\overline{x} + \alpha d) \geq 0$ se puede desdoblar en dos:

$$\begin{array}{l} \text{(i) } (\overline{x} + \alpha d)^B \geq 0 \leftrightarrow \overline{x}^B + \alpha q \geq 0 \\ \text{(ii) } (\overline{x} + \alpha d)^N \geq 0 \leftrightarrow \overline{x}^N + \alpha p \geq 0 \leftrightarrow \alpha p \geq 0 \end{array}$$

Es obvio que $\alpha p \geq 0$ se cumple siempre (puesto que $\alpha \geq 0$ y $p \geq 0$), y no hace falta pues exigir (ii). En consecuencia, el problema (3.10) es equivalente a

$$\begin{array}{ll} Max & \alpha \\ s.a & \overline{x}^B + \alpha q \geq 0 \\ & \alpha \geq 0 \end{array}$$

Vamos a considerar la restricción $\overline{x}^B + \alpha q \geq 0$. Es una restricción vectorial, y la podemos escribir componente a componente:

$$\overline{x}_i^B + \alpha q_i \geq 0, \, \forall i = 1, ..., m$$

Desdoblamos estas restricciones en dos grupos:

$$\begin{array}{l} \text{(i) } \overline{x}_i^B + \alpha q_i \geq 0, \, \forall i / q_i \geq 0 \\ \text{(ii) } \overline{x}_i^B + \alpha q_i \geq 0, \, \forall i / q_i < 0 \end{array}$$

Es obvio que (i) se cumple siempre (puesto que $\overline{x}_i^B \geq 0$, al ser $\overline{x} \in S$), y no hace falta pues exigirlo. En cuanto a (ii), despejamos:

$$\alpha q_i \geq -\overline{x}_i^B, \, \forall i / q_i < 0$$

$$\alpha \leq \frac{-\overline{x}_i^B}{q_i}, \, \forall i / q_i < 0$$

(obsérvese el paso de $\geq$ a $\leq$ cuando dividimos por un número estrictamente negativo ambos miembros). Distinguimos ahora dos casos:

Caso 1: si $\exists i / q_i < 0$. Entonces definimos

$$\Delta := \text{mín} \left\{ \frac{\overline{x}_i^B}{-q_i} / q_i < 0 \right\}$$

(Δ es un número real bien definido), con lo que (ii) equivale a

$$\alpha \leq \Delta$$

Obsérvese que $\Delta \geq 0$.

Caso 2: si $\nexists i / q_i < 0$. Entonces definimos $\Delta := \infty$, con lo que el que no haya ninguna desigualdad en (ii) se puede también expresar simbólicamente $\alpha \leq \Delta$.

Hemos obtenido que el problema (3.10) es equivalente a

$$\begin{array}{ll} Max & \alpha \\ s.a & 0 \leq \alpha \leq \Delta \end{array}$$

Consideramos dos casos:

Caso 1: si $\Delta < \infty$ (Δ es un número real). Entonces el problema unidimensional (3.10) tiene una, y una sola solución óptima, para el valor $\alpha = \Delta$.

Caso 2: si $\Delta = \infty$. Entonces el problema unidimensional (3.10) no tiene solución óptima. Se puede demostrar que en este caso el problema original (PL) tampoco tiene solución óptima. Se dice entonces que el problema (PL) es *no acotado* (es factible, pero la función objetivo decrece en S cuanto queramos).

Carácter básico de la nueva solución factible

Resumamos lo visto hasta ahora sobre la mejora en el método del símplex. Sean una base B, una no base N, y la correspondiente solución factible básica $\overline{x}$. Supongamos que no se satisface el criterio de optimalidad. El Teorema de direcciones de mejora nos proporciona una dirección de mejora d. Ahora hay dos posibilidades. Si $\Delta = \infty$, entonces el problema (PL) es no acotado, y finalizamos. Si $\Delta < \infty$, entonces mejoramos, y la nueva solución factible es $\overline{x} + \Delta d$.

¿Hemos ya concluido la descripción del proceso de mejora? Nos queda solo una cuestión pendiente: demostrar que $\overline{x} + \Delta d$ es solución básica. De esta manera justificaremos que hemos pasado de solución factible básica a solución factible básica, y que así estamos en condiciones de iniciar una nueva iteración.

Es obvio que en condiciones normales $(\overline{x} + \Delta d)_k^N \neq 0$, así que $\overline{x} + \Delta d$ no será solución básica respecto a la base de partida B. Vamos a probar que efectivamente $\overline{x}+\Delta d$ es una solución básica, pero respecto a una base distinta de la inicial. Esta base se obtendrá cambiando una sola columna de la base inicial. ¿Qué tenemos en mente? En la nueva no base, la columna que ocupaba la posición k-ésima en la no base antigua ya no nos sirve, pues $(\overline{x} + \Delta d)_k^N$ ya no es nula. Necesitamos encontrar una columna que la sustituya, que corresponda a una componente del vector $(\overline{x} + \Delta d)$ que sea nula.

En nuestra búsqueda de esa columna, vamos a ver cómo se alteran las componentes de $\overline{x}$ al avanzarse en la dirección de mejora d proporcionada por el Teorema de direcciones de mejora. Se van recorriendo puntos de la forma $(\overline{x}+\alpha d)$, con $\alpha \geq 0$, hasta que nos detenemos cuando $\alpha = \Delta$. Recordemos que en la parte no básica se quedan todas las componentes como estaban (en su valor cero), salvo una de ellas, la que ocupa el lugar k-ésimo, que crece. Y qué pasa en las componentes básicas? Para la componente básica que ocupa la posición i-ésima se cumple que

$$(\overline{x} + \alpha d)_i^B = \overline{x}_i^B + \alpha q_i$$

o sea que esa componente crece si $q_i > 0$, decrece si $q_i < 0$, y no varía si $q_i = 0$. Gráficamente:

$$\overline{x}^N + \alpha p = \underset{\uparrow\ k}{(\ 0\ ,\ 0\ ,\ 0\ ,...,\ 0\ )} \quad , \quad \overline{x}^B + \alpha q = \underset{\uparrow \quad \downarrow \quad \updownarrow \quad \downarrow}{(\quad ,\quad ,\ 0\ ,...,\quad ,\quad)}$$

¿Hasta cuánto puede aumentar α? Hasta que una de las componentes básicas que decrecen ($q_i < 0$) alcance el valor cero (si se siguiera avanzando esa componente tomaría valores estrictamente negativos); en ese valor de α nos detenemos (es el valor $\alpha = \Delta$), precisamente cuando por vez primera una de las componentes que decrece toma el valor cero.

¿En qué componente ocurrirá esto? Consideremos la definición de Δ. El mínimo se tomará en una de las posiciones i tal que $q_i < 0$; digamos que sea en la posición l. Se tiene que

$$\Delta := \text{mín}\left\{ \frac{\overline{x}_i^B}{-q_i} / q_i < 0 \right\} = \frac{\overline{x}_l^B}{-q_l}$$

(Si hay empate y el mínimo se toma en varias posiciones, sea l una cualquiera de ellas).

Definimos la nueva base (y la nueva no base). La nueva base B_{nueva} se obtendrá sustituyendo la columna que estaba en la posición l de la base antigua B por la columna que estaba en la posición k de la no base antigua N; la nueva no base N_{nueva} se obtendrá sustituyendo la columna que estaba en la posición k de la no base antigua N por la columna que estaba en la posición l de la base antigua B. En pocas palabras, permutamos la columna que ocupa la posición l de la base con la que ocupa la posición k de la no base.

Debemos ahora responder dos preguntas:

1. ¿Es B_{nueva} una base? O sea, ¿son sus columnas linealmente independientes?

La respuesta es afirmativa, aunque no vamos a demostrarlo aquí.

2. ¿Es $\overline{x}+\Delta d$ solución básica respecto a B_{nueva}? Según la definición de solución básica, para demostrarlo hace falta verificar dos cosas:

2.1. ¿Satisface $\overline{x}+\Delta d$ el sistema $Ax=b$? Sí, puesto que $\overline{x}+\Delta d$ es factible.

2.2. ¿Son nulas todas las componentes de $\overline{x}+\Delta d$ que son no básicas respecto a B_{nueva}? Las componentes de $\overline{x}+\Delta d$ que son no básicas respecto a B_{nueva}, son las siguientes (formuladas con la notación "antigua" de B y N):

- $(\overline{x}+\Delta d)^N_j$, para $j=1,...,(n-m), j\neq k$. Estas posiciones eran no básicas respecto a la base antigua, y siguen siéndolo respecto a la nueva.
- $(\overline{x}+\Delta d)^B_l$. Esta posición era básica respecto a la base antigua, pero ahora es no básica respecto a la nueva.

Para ver que $(\overline{x}+\Delta d)^N_j=0$ si $j\neq k$, basta considerar (3.9).

Veamos ahora que $(\overline{x}+\Delta d)^B_l$ es nulo:

$$(\overline{x}+\Delta d)^B_l=\overline{x}^B_l+\Delta q_l=\overline{x}^B_l+\left(\frac{\overline{x}^B_l}{-q_l}\right)q_l=0$$

Hemos pues demostrado que $(\overline{x}+\Delta d)$ es solución básica respecto a la nueva base B_{nueva}.

Escolio: el caso $\Delta=0$

Nunca es posible que $\Delta<0$ ¿Es posible que $\Delta=0$? Sí. En este caso $(\overline{x}+\Delta d)=\overline{x}$, y permanecemos en la misma solución factible básica, pero cambia la base (así $\overline{x}$ resulta ser básica respecto a dos bases distintas).

Se dice que una solución básica es degenerada si alguna de sus componntes básicas es nula. Observando la definición de Δ, el caso $\Delta = 0$ solo se podrá producir si $\overline{x}$ es degenerada (respecto a B). Pero es posible que $\overline{x}$ sea degenerada y sin embargo $\Delta > 0$.

En el caso $\Delta = 0$, se tiene que d, aunque es dirección de mejora, no es dirección factible de mejora. Al no poderse avanzar en la dirección d sin abandonar el conjunto factible, nos quedamos donde estamos, en $\overline{x}$.

3.5. Método del símplex

El algoritmo y la Fase I

Estamos ya en condiciones de escribir el algoritmo del símplex.

ALGORITMO DEL SÍMPLEX

DATOS: m, n, c, A, b.

PASO 1 (inicialización)

Elíjanse: una base B, N, una solución factible básica $\overline{x}$ (así: I, J)

PASO 2 (criterio de optimalidad)

Si $\left\{j \in \overline{n-m} / (c^N - c^B B^{-1} N)_j < 0\right\} = \emptyset$: $\overline{x}$ óptimo; FIN

PASO 3 (mejora)

Elíjase $k \in \left\{j \in \overline{n-m} / (c^N - c^B B^{-1} N)_j < 0\right\}$;

$q \leftarrow -B^{-1} N_k$;

$\Delta \leftarrow \text{mín} \left\{ \frac{\overline{x}_i^B}{-q_i} / q_i < 0 \right\}$;

Si $\Delta = \infty$: problema no acotado; FIN

Si $\Delta < \infty$: Elíjase l de entre los i en que se ha tomado el mínimo;
tróquense J_k, I_l;
váyase al paso 2

Obsérvese que a la hora de elegir k puede haber empate, esto es, haber varias posiciones j en que $(c^N - c^B B^{-1} N)_j < 0$. En tal caso se puede elegir arbitrariamente una de ellas. Análogamente a la hora de elegir l puede haber empate, esto es, haber varias posiciones i en que se tome el mínimo de $\left\{ \frac{\overline{x}_i^B}{-q_i} / q_i < 0 \right\}$. En tal caso también se puede elegir arbitrariamente una de ellas. En la práctica, los paquetes informáticos comerciales aplican reglas empíricas para elegir k y l en caso de empate. En el caso de k, la regla habitual es elegir la posición j en que $\left|(c^N - c^B B^{-1} N)_j\right|$ es mayor.

En el algoritmo se parte del problema (PL), y se supone que es factible; por otra parte, no se detalla cómo se obtiene la base factible inicial (fijada ésta, es trivial escoger una no base). La llamada *Fase I del símplex* es un algoritmo que se aplica para inicializar el algoritmo del símplex. Realiza dos misiones:

> (i) Partiendo de un problema de optimización lineal en forma estándar, construye un problema transformado que cumple las hipótesis de (PL), a cuya resolución se reduce la del problema original.
>
> (ii) Comprueba si el nuevo problema es factible. Si no es así, el proceso finaliza. Si es factible, halla una base factible inicial. (Todo problema (PL) factible tiene al menos una base que es factible).

¿Cómo realiza su trabajo la Fase I del símplex? En cuanto a (i), se añaden columnas a la matriz del sistema (el procedimiento de suprimir ecuaciones redundantes es difícil). En cuanto a (ii), se aplica el mismo algoritmo del símplex a un problema auxiliar.

La versión que aquí se da del algoritmo del simplex está estilizada, en relación a la que llevan los paquetes comerciales. El mayor esfuerzo computacional en el algoritmo estriba en la inversión de la base que hay que realizar en cada iteración. Dado que una base difiere de la base siguiente solamente en una columna, se utilizan procedimientos que permiten pasar de la inversa de una base a la de la base siguiente con un esfuerzo computacional razonable.

Ejemplo

En nuestro ejemplo ilustrativo, hemos partido de la base $I = (1, 4, 5)$, $J = (2, 3)$. En el Paso 2,

$$(c^N - c^B B^{-1} N) = (1, 13) - (1, 10, 0) B^{-1} N = (-9, 3)$$

No estamos en un óptimo, al ser la segunda componente estrictamente negativa. Así que aplicamos el Teorema de direcciones de mejora, con $k = 1$. Ahora:

$$q = -B^{-1}N_k = -B^{-1}\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ -1 \end{pmatrix}$$

Pasamos ahora a calcular Δ:

$$\Delta = \text{mín}\left\{\frac{\overline{x}_i^B}{-q_i}/q_i < 0\right\} = \text{mín}\left\{\frac{5}{1}, \frac{2}{1}\right\} = 2$$

El mínimo se ha tomado para $i = 3$, y, en consecuencia, $l = 3$. Podemos realizar el cambio de base:

ANTES	DESPUÉS
$I = (1, 4, 5)$	$I = (1, 4, 2)$
$J = (2, 3)$	$J = (5, 3)$

Con ello concluye la iteración. La nueva base y la nueva no base son, matricialmente,

$$B = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 0 & -1 & -1 \end{pmatrix},\ N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Vamos a hallar la nueva solución factible básica:

$$\overline{x}^B = B^{-1}b = \begin{pmatrix} 1 & 0 & 1 \\ -1 & -1 & -2 \\ 1 & 1 & 1 \end{pmatrix}\begin{pmatrix} 5 \\ 2 \\ -5 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 2 \end{pmatrix}$$

Por supuesto,

$$\overline{x}^N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

En consecuencia, la nueva solución factible básica es:

$$\overline{x} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 3 \\ 0 \end{pmatrix}$$

Hallemos la dirección de mejora d que hemos utilizado. Su parte básica $d^B = q$ la hemos ya calculado. La parte no básica será:

$$d^N = p = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Luego

$$d = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \text{-1} \\ \text{-1} \end{pmatrix}$$

Podemos ahora comprobar que efectivamente se verifica que

$$\overline{x}_{\text{nueva}} = \overline{x}_{\text{antigua}} + \Delta d = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \\ 2 \end{pmatrix} + 2 \begin{pmatrix} 0 \\ 1 \\ 0 \\ \text{-1} \\ \text{-1} \end{pmatrix}$$

Convergencia del método del símplex

Decimos que un método es convergente cuando termina en un número finito de etapas para todas las instancias del problema. Si un método no es convergente, no se le puede llamar algoritmo. Sin embargo, el llamado "algoritmo" del símplex, tal como lo hemos enunciado (y tal como aparece en los paquetes informáticos comerciales) no es convergente, en sentido estricto.

Para darnos cuenta de por qué el método del símplex puede no converger, vamos a "demostrar" (erróneamente) que el símplex converge. El razonamiento tiene tres pasos:

(1) Dada una instancia del problema (PL), existe un número finito de bases.

(2) Si no finalizamos en cierta iteración, en ella se pasa de una base a otra distinta (aquí entendemos por "base" el vector I con las etiquetas de sus columnas).

(3) Como hay un número finito de bases, y en cada iteración en la que no finalizamos se pasa de una base a otra distinta, habremos de

finalizar en un número finito de iteraciones (etapas), porque si no se acabarían las bases.

Los pasos (1) y (2) son correctos. Pero el paso (3) es erróneo; veamos por qué. Podríamos pasar de una base I a otra distinta I', de esta a otra distinta I'', y de ésta a otra distinta I'''..., que coincidiera con I ($I''' = I$). A este fenómeno se le llama *ciclaje*.

Así pues, el método del símplex puede no converger, en teoría, debido al posible ciclaje. Sin embargo, en la práctica (en que se utilizan computadores), el fenómeno del ciclaje es irrelevante. Ello tiene una justificación, cuya idea fundamental es la siguiente. El ciclaje se produce por la arbitrariedad con que se elijen k y l cuando hay empate. Ahora bien, el mismo error de redondeo de la máquina hace que se produzcan pequeñas perturbaciones que impiden que se vuelva a una base que ya se había visitado antes.

Existe una variante del método del símplex, llamada *variante lexicográfica*, en la que el ciclaje es imposible (no sólo en la práctica computacional, sino en teoría). La idea de esta variante, en cuyos detalles no entramos, es evitar la arbitrariedad con que se elije l cuando hay empate, mediante una regla que determina la elección. Tenemos pues el siguiente resultado:

Teorema. Sea el problema (PL). Entonces el método del símplex, variante lexicográfica, es convergente.

Este resultado es importante, aunque no se conozca cómo es la variante lexicográfica (la cual, por cierto, no se utiliza en la práctica computacional). Vamos a demostrar de manera muy sencilla el siguiente teorema a partir del teorema anterior.

Teorema. Si (PL) tiene solución óptima, entonces tendrá al menos una solución óptima básica.

Demostración. Aplicamos a (PL) la variante lexicográfica del símplex. Como (PL) tiene solución óptima, entonces, por el teorema anterior, la variante lexicográfica nos dará una solución óptima en un número finito de etapas. Pero esta solución óptima que hemos obtenido, como todas las del símplex, será básica.

El que el método del símplex resulte computacionalmente "convergente en la práctica" y, sobre todo, que la variante lexicográfica sea convergente, conduce a que se hable del *algoritmo del símplex*. Hay en ello cierto abuso del lenguaje, porque, en sentido estricto, solo la variante lexicográfica es un algoritmo.

3.6. Cuestiones de eficiencia y complejidad

Eficiencia del algoritmo del símplex

El algoritmo del símplex es exponencial.

Sin embargo, su funcionamiento es muy bueno en casi todas las instancias, salvo en algunas "patológicas" (que son las que motivan que su eficiencia sea exponencial). En la práctica, se puede decir que el algoritmo del símplex es, en la gran mayoría de las instancias, *aproximadamente* lineal.

El algoritmo del símplex representa pues una excepción en cuanto al significado práctico de su eficiencia. Hay para ello razones matemáticas, en las que no vamos a entrar.

Complejidad del problema de optimización lineal

A pesar de lo que pudiera pensarse tras lo dicho en el epígrafe anterior:

Teorema. El problema de optimización lineal es polinomial.

El algoritmo polinomial diseñado por Khachiyan para demostrar el teorema, no resultó ser eficiente en la práctica, a pesar de su interés teórico. Karmarkar encontró un nuevo algoritmo polinomial, y los algoritmos basados en su idea, llamados *algoritmos de punto interior*, han sido perfeccionados desde entonces por los matemáticos.

La situación actual es la siguiente:

(i) El algoritmo del símplex y los mejores algoritmos de punto interior son de similar eficiencia práctica para los problemas de optimización lineal sin una estructura especial. El algoritmo del símplex es, con mucho, el que más se utiliza.

(ii) En cuanto a los problemas de optimización lineal especiales, el símplex resulta ser mejor para algunos de ellos, y los algoritmos de punto interior para otros.

El tiempo dirá qué algoritmos acabarán prevaleciendo en los problemas de optimización líneal. De momento el algoritmo del símplex sigue manteniendo una posición dominante. En todo caso, los elementos fundamentales del símplex forman parte de los conocimientos necesarios para estudiar los algoritmos para los problemas de optimización líneal.

4. Flujos en redes

4.1. Conceptos generales

Red

Una red es un grafo en cuyos arcos hay una restricción de capacidad. Formalmente:

Sea $G = (X, U)$ un grafo. Sea una función

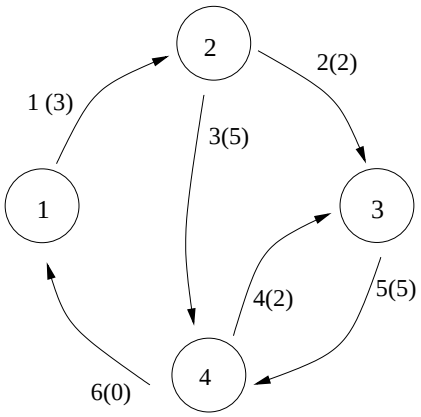
$$\begin{array}{rl} u: & U \to \mathbb{R}_+ \cup \{\infty\} \\ & j \to u(j) \end{array}$$

Se llama *red* al par $(G = (X, U), u)$. A u se le llama la *función de capacidades* de la red (y $u(j)$ es la *capacidad* del arco j).

Dada una red $(G = (X, U), u)$, en esta sección supondremos que $X = \{1, 2, ..., m\}$, $U = \{1, 2, ..., n\}$, $m, n \in \mathbb{N}^*$, y el grafo G es irreflexivo. En consecuencia, G tendrá una matriz de incidencia $A = (a_{ij})_{\substack{i=1,...,m \\ j=1,...,n}}$. Denotaremos normalmente u_j en vez de $u(j)$, y utilizaremos la notación vectorial para u:

$$u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$$

En la figura denotamos la etiqueta de cada arco mediante un número junto al arco, y las capacidades mediante números (o el símbolo ∞) entre paréntesis. En este caso $u = \begin{pmatrix} 3 \\ 2 \\ 5 \\ 2 \\ 5 \\ 0 \end{pmatrix}$.



Si los arcos describen vías de transporte (de material, electricidad, información...), las capacidades de los arcos se pueden interpretar como cantidades máximas que pueden circular por el arco en cierto periodo de tiempo (si un arco tiene capacidad ∞, quiere decir que no hay restricciones para la cantidad que puede circular por el arco).

Flujos

Sea una red $(G = (X, U), u)$. Se llama *flujo* a un vector $x \in \mathbb{R}^n_+$. Un flujo se puede interpretar como descripción de la circulación de material por la red en cierto periodo de tiempo: x_j es la cantidad de material que va por el arco j. Dado el flujo x, a x_j se le llama el flujo por el arco j.

Se dice que un flujo x es *admisible* si $x \leq u$.

Por ejemplo, el flujo $x = \begin{pmatrix} 3 \\ 0 \\ 5 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ en la red de la figura es admisible.

Saldos

Un flujo en una red producirá un *saldo* en cada vértice: la suma de los flujos de los arcos incidentes con el vértice, considerando los flujos de los arcos que salen

como positivos y los flujos de los arcos que entran como negativos. (Recordemos la regla mnemotéctica: "dar es positivo").

Veamos cuáles serían los saldos para el flujo $x = \begin{pmatrix} 3 \\ 0 \\ 5 \\ 1 \\ 0 \\ 0 \end{pmatrix}$.

Vértice 1:

Sale:	(lo que sale por el arco 1)	$+3$
Entra:	(lo que entra por el arco 6)	-0
		$+3$

Vértice 2:

Sale:	(lo que sale por los arcos $2, 3$)	$+5$
Entra:	(lo que entra por el arco 1)	-3
		$+2$

Vértice 3:

Sale:	(lo que sale por el arco 5)	$+0$
Entra:	(lo que entra por los arcos $2, 4$)	-1
		-1

Vértice 4:

Sale:	(lo que sale por los arcos $4, 6$)	$+1$
Entra:	(lo que entra por los arcos $3, 5$)	-5
		-4

Podemos decir que en el vértice 1 el flujo produce una "salida neta" de 3 unidades, o en el vértice 4 una "entrada neta" de 4 unidades.

Denotaremos los saldos producidos por un flujo x mediante un vector de $\mathbb{R}^m$, cuya i-ésima componente es el saldo producido por x en el vértice i: es el *vector de saldos de* x. En este ejemplo el vector de saldos sería $\begin{pmatrix} 3 \\ 2 \\ -1 \\ -4 \end{pmatrix}$.

Relación entre un flujo y sus saldos

Multipliquemos la matriz de incidencia del grafo de la figura por el flujo anterior:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \\ 5 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ -1 \\ -4 \end{pmatrix}$$

Lo que hemos obtenido no es una coincidencia.

Teorema. Sea una red $(G = (X, U), u)$. El producto de la matriz de incidencia del grafo por un flujo x, es el vector de saldos de x.

La demostración del siguiente corolario a partir del teorema queda como ejercicio. (Sugerencia: considérese la suma de las filas de una matriz de incidencia.)

Corolario. Sean una red $(G = (X, U), u)$ y un flujo x. La suma de las componentes del vector de saldos de x es cero.

4.2. Problemas clásicos

El problema de transbordo

El problema de transbordo desempeña un papel central en el estudio de flujos en redes, por su gran generalidad, y porque a su resolución se puede reducir la de muchos otros problemas.

Consideremos una red que describe un sistema de transporte entre ciertos centros (vértices) unidos por vías (arcos). Dado un periodo de tiempo, de algunos centros se ha de sacar material (v.g., centros de producción), y en otros se ha introducir (v.g., centros de venta). Por otra parte, el transporte tendrá costos, que dependerán de la via que se utilice. Los datos del problema son pues:

(i) Una red $(G = (X, U), u)$.

(ii) Un vector de saldos requeridos $r = \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix}$, donde r_i es el saldo requerido en el vértice i.

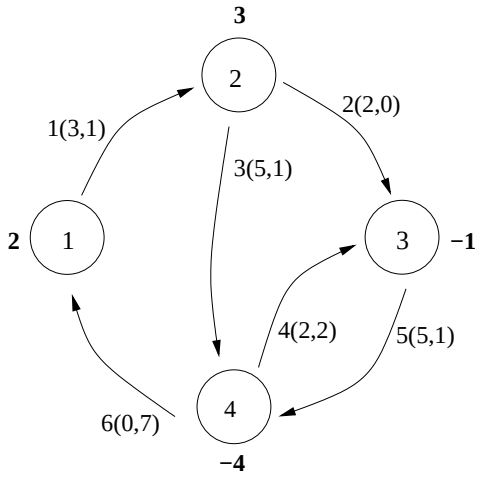
(iii) Un vector de costes $c = (c_1, c_2, ..., c_n)$, donde c_j es el coste unitario por el arco j.

La circulación de material por la red vendrá representada por un flujo x. Los flujos que satisfacen nuestras restricciones son aquellos flujos admisibles tales que $Ax = r$. De entre ellos nos interesa escoger uno de coste mínimo. Nuestro problema se puede formular así matemáticamente:

$$\begin{array}{lll} \text{(PT)} & Min & cx \\ & s.a & Ax = r \\ & & 0 \leq x \leq u \\ & & [x \in \mathbb{R}^n] \end{array}$$

A un problema matemático del tipo (PT), donde A es la matriz de incidencia de un grafo, se le denomina *problema de transbordo.*

Como ejemplo, consideremos la red de la figura anterior, con vector de saldos requeridos $r = \begin{pmatrix} 2 \\ 3 \\ -1 \\ -4 \end{pmatrix}$ y vector de costes $c = (1, 0, 1, 2, 1, 7)$. En la figura denotamos la etiqueta de cada arco mediante un número junto al arco, las capacidades y costes mediante un par ordenado (u_j, c_j) junto al arco, y los saldos requeridos mediante un número en negrita al lado de cada vértice. Los vértices 1 y 2 podrían representar fábricas, y los vértices 3 y 4 puntos de venta.



En este ejemplo el problema matemático (PT) sería:

$$
\begin{array}{ll}
Min & x_1 + x_3 + 2x_4 + x_5 + 7x_6 \\
s.a & x_1 - x_6 = 2 \\
 & -x_1 + x_2 + x_3 = 3 \\
 & -x_2 - x_4 + x_5 = -1 \\
 & -x_3 + x_4 - x_5 + x_6 = -4 \\
 & 0 \leq x_1 \leq 3 \\
 & 0 \leq x_2 \leq 2 \\
 & 0 \leq x_3 \leq 5 \\
 & 0 \leq x_4 \leq 2 \\
 & 0 \leq x_5 \leq 5 \\
 & 0 \leq x_6 \leq 0
\end{array}
$$

El problema de transbordo (PT) es un problema de optimización lineal en forma estándar ordinario, salvo por dos particularidades:

(1) Las variables no sólo tienen cota inferior (cero), sino también pueden tener cota superior ($0 \leq x \leq u$). (Si $u_j = \infty$, para cierto arco j, entonces x_j no tiene cota superior.)

(2) La matriz del sistema A es la matriz de incidencia de un grafo.

Los problemas de optimización lineal que cumplen (1) se dice que tienen *variables acotadas*. Ciertamente las restricciones de la forma $x_j \leq u_j$ se pueden convertir en igualdades, cara a obtener un problema de optimización lineal en forma estándar, al cual se pueda aplicar el algoritmo del símplex. Sin embargo, existe una variante del símplex que trabaja directamente con estas restricciones: se llama el *algoritmo del símplex con variables acotadas*. La principal diferencia con el símplex ordinario estriba en la definición de solución básica. Dada una base, las componentes no básicas de una "solución básica" pueden valer ahora, o bien cero, o bien la cota superior de la variable ($\overline{x}_j^N = 0$ ó $\overline{x}_j^N = u_j^N$).

Para resolver el problema de transbordo se utiliza normalmente una variante del símplex con variables acotadas especializada al hecho de que la matriz del sistema es una matriz de incidencia. A esta variante se le llama *algoritmo de transbordo*.

El problema de transporte

Considérese un problema de transbordo, con la notación del epígrafe anterior. Sea un vértice i. Si $r_i > 0$, se dice que i es un vertice de oferta; si $r_i < 0$, se dice que i es un vertice de demanda; si $r_i = 0$, se dice que i es un vertice de transbordo.

El problema de transporte es un caso especial de problema de transbordo. Se llama problema de transporte a un problema de transbordo que cumple dos condiciones:

(i) No hay vértices de transbordo.

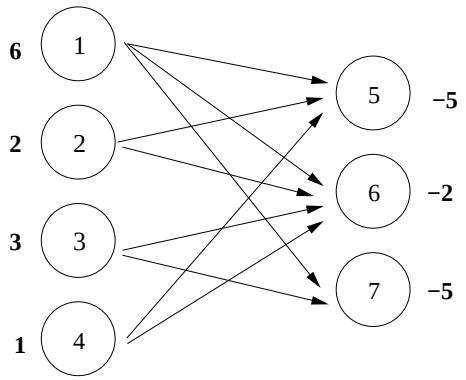
(ii) Sólo hay arcos desde vertices de oferta a vértices de demanda.

Gráficamente, los problemas de transporte se suelen representar como en la figura, con los vértices de oferta a la izquierda y los de demanda a la derecha (no se han indicado etiquetas, capacidades o costes de los arcos, por claridad).

Aunque hay un algoritmos específicos para el problema de transporte, lo más frecuente es que el problema de transporte se resuelva aplicando el algoritmo de transbordo.

El problema de flujo máximo

El problema de flujo máximo no es un caso particular del problema de transbordo, pero reduciremos su resolución a la del problema de transbordo.



Intuitivamente, el *problema de flujo máximo* consiste en lo siguiente. Dada una red y dos vértices distinguidos s y t, encontrar un flujo que lleve la mayor cantidad posible de material desde s hasta t. Vamos ahora a precisar esta idea definiendo formalmente el problema.

Los datos del problema de flujo máximo son:

(i) Una red $(G = (X, U), u)$.

(ii) Dos vértices $s, t \in X, s \neq t$. (Al vértice s se le llama *fuente*, y al vértice t, *sumidero*.)

Por la naturaleza del problema, podemos suponer que no hay ningún arco desde el sumidero a la fuente.

Sea

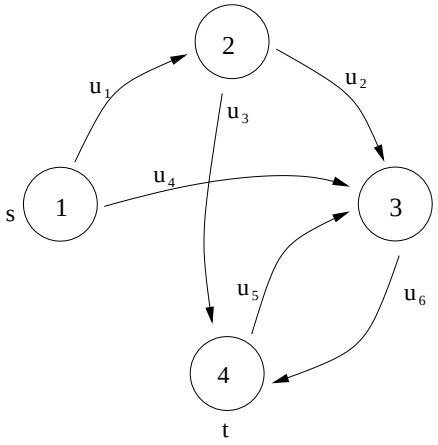
$$B := \{x \in \mathbb{R}^n : 0 \leq x \leq u \text{ y el saldo de } x \text{ en todos los vértices, salvo } s \text{ y } t, \text{ es cero}\}$$

Si $x \in B$, puesto que la suma de las componentes del vector de saldos de cualquier flujo es cero, se tiene que

$$(\text{saldo de } x \text{ en } s) = -(\text{saldo de } x \text{ en } t)$$

Intuitivamente, los flujos de B corresponden a la idea de "llevar material de s a t" (no se queda nada en los otros vértices). Podemos ya definir el problema. El *problema de flujo máximo* consiste en encontrar un flujo de B cuyo saldo en la fuente sea máximo (alternativamente, cuyo saldo en el sumidero sea mínimo).

En la figura aparece un problema de flujo máximo. Hemos denotado la capacidad del arco j por u_j; la fuente es el vértice 1, y el sumidero el vértice 4.



Dado un problema de flujo máximo (PFM), vamos a definir su *problema de transbordo asociado* (PTA), de manera que se reduzca la resolución del (PFM) a la de su (PTA). Sea pues un (PFM) con la notación anterior; su *problema de transbordo asociado* se define por sus elementos:

(1) El grafo es el mismo que el del (PFM), pero añadiendo un arco más desde el sumidero a la fuente (es el llamado *arco auxiliar*).

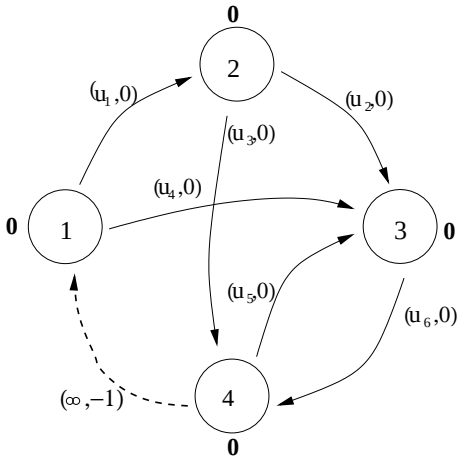
(2) Las capacidades de los arcos son las del (PFM), y al arco auxiliar se le asigna capacidad ∞.

(3) Los saldos requeridos son cero en todos los vértices.

(4) Los costos son cero en todos los arcos, salvo en el arco auxiliar, que es -1.

El (PTA) del (PFM) de la figura anterior sería el siguiente.

Teorema. Sea un (PFM). Si $\begin{pmatrix} \overline{x}_1 \\ \overline{x}_2 \\ \vdots \\ \overline{x}_n \\ \overline{x}_{n+1} \end{pmatrix}$ es una solución óptima de su (PTA) (donde la última componente corresponde al arco auxiliar), entonces $\begin{pmatrix} \overline{x}_1 \\ \overline{x}_2 \\ \vdots \\ \overline{x}_n \end{pmatrix}$ es una solución óptima del (PFM).



Podemos pues resolver el problema de flujo máximo aplicando el algoritmo de transbordo a su (PTA).

¿Cuál es la idea intuitiva subyacente al teorema? Al hacer que el saldo requerido en s y t sea también cero, obligamos a que todo el material que "de verdad" (en el grafo original) vaya de s a t, vuelva "virtualmente" de t a s a través del arco auxiliar. De esta manera, podemos medir la totalidad del material que va de s a t, haciéndolo pasar "virtualmente" por el arco auxiliar. Basta ahora maximizar la cantidad de flujo por el arco auxiliar (que por ello tiene costo -1).

Considerable interés teórico tiene un algoritmo específico para resolver el problema de flujo máximo: el *algoritmo de Ford-Fulkerson*. Sin embargo, en los paquetes informáticos comerciales se suele aplicar el algoritmo de transbordo, mediante la reducción del teorema anterior.

5. Planificación de actividades

5.1. Introducción

Idea del problema

Supongamos que tenemos que realizar una tarea compleja, a la que se denomina *proyecto*. Para ello la descomponemos en tareas sencillas, llamadas *actividades*. En *planificación de actividades* se trata de distribuir en el tiempo la ejecución de las actividades para la óptima realización del proyecto.

Sea el siguiente ejemplo ilustrativo. El proyecto es la construcción de un edificio. Consideramos las siguientes actividades:

(1) Empezar

(2) Explanar

(3) Vallar

(4) Montar la grúa

(5) Hacer los cimientos

(6) Hacer la caseta de obra

(7) Levantar la estructura

(8) Realizar las conexiones con el exterior (agua, electricidad, alcantarillado...)

(9) Hacer las paredes

(10) Terminar

Los meros hechos de *empezar* y *terminar* hay que considerarlos siempre como actividades; son las llamadas *actividades ficticias*.

A partir de la lista de actividades, se elabora la *tabla de actividades*, en la que figura para cada actividad i: su duración y las actividades que la han de preceder para poder ejecutarse i. En nuestro ejemplo (duraciones en semanas):

actividades	duración	predecesoras
1	0	–
2	4	1
3	10	1
4	6	2
5	2	2
6	11	2
7	22	$4, 5$
8	3	5
9	17	$3, 6, 8$
10	0	$7, 9$

(El lector deberá tomarse los datos de la tabla con un poco de humor.)

En toda tabla de actividades se han de observar las siguientes reglas:

(i) La duración de las actividades ficticias es siempre cero.

(ii) En planificación de actividades, cuando se habla de "actividades predecesoras" hay que entender "actividades predecesoras inmediatas". Esto quiere decir que si se cataloga la actividad i como predecesora de la actividad j, ya no se pueden catalogar como predecesoras de j las predecesoras de i, ni las predecesoras de las predecesoras de i, etc. Así, en el ejemplo, la actividad 5 figura como predecesora de 8, pero ya no se catalogan como predecesoras de 8 las predecesoras de 5 (la 2), ni las predecesoras de las predecesoras de 5 (la 1), pese a que, evidentemente, las actividades 2 y 1 también han de realizarse antes de poder ejecutarse la 8.

(iii) Las predecesoras de la actividad "terminar" se determinan mediante una regla matemática, que veremos más adelante.

Grafo de actividades

En lo sucesivo suponemos que tenemos un proyecto que consta de n actividades, que etiquetamos $1, 2, ..., n$. La actividad 1 es la de "comenzar", y la actividad n la de "terminar". Denotamos la duración (medida en cierta unidad de tiempo) de la actividad i por τ_i; evidentemente $\tau_i \geq 0$.

A partir de la tabla de actividades construimos el *grafo de actividades*, que es un grafo ponderado $G = (X, U, p)$. Lo definimos a continuación por sus elementos:

(1) Vértices: hay un vértice por cada una de las actividades. (Se identifican vértices y actividades.)

(2) Arcos: $(i, j) \in U$ cuando la actividad i precede a la actividad j. (La relación "técnica" de precedencia de actividades se identifica con la relación "matemática" de precedencia de vértices en un grafo.)

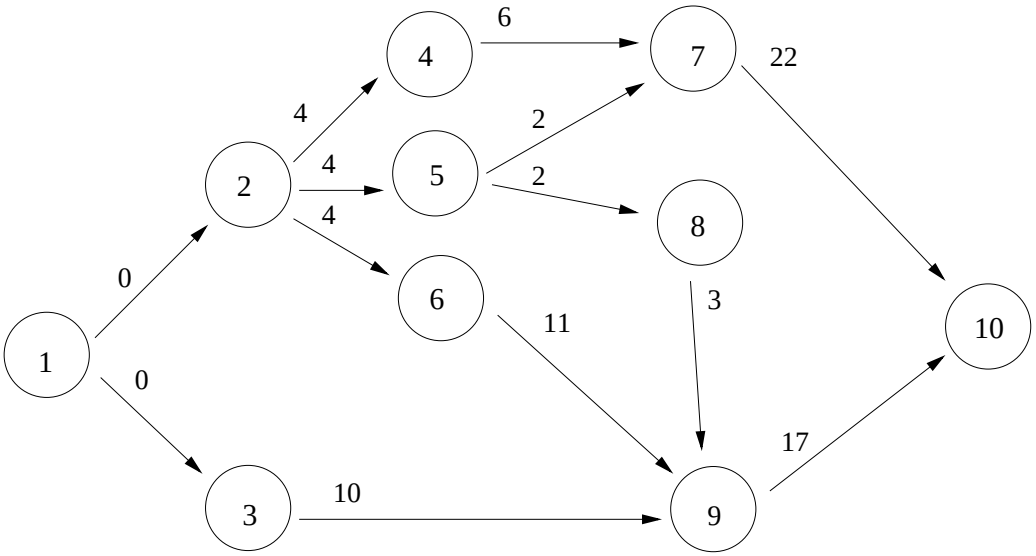
(3) Pesos: todos los arcos que salen del mismo vértice tienen el mismo peso, que es la duración del vértice (o sea, de la actividad correspondiente al vértice).

En la figura aparece el grafo de actividades del ejemplo ilustrativo anterior.

Teniendo en cuenta su procedencia, asumimos que el grafo de actividades satisface:

(i) Es conexo y acíclico.

(ii) Dada una actividad i, existe al menos un camino desde el vértice 1 (o vértice origen) hasta el vértice i, y existe al menos un camino desde el vértice i hasta el vértice n (o vértice final).



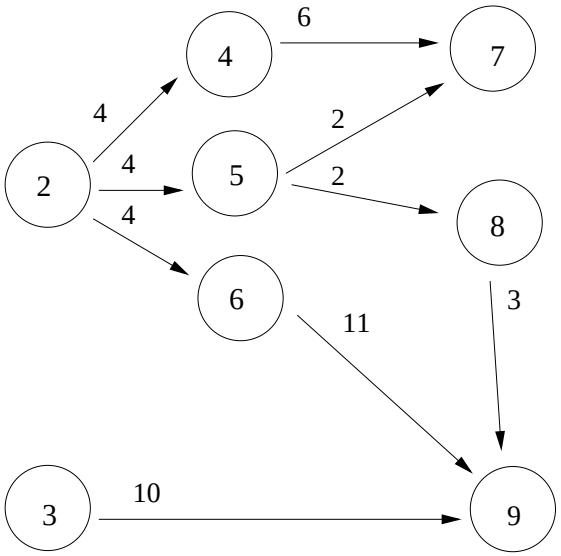
(Si $i = 1$, convenimos en que hay un camino de longitud cero desde el vértice 1 hasta i; análogamente, si $i = n$, convenimos en que hay un camino de longitud cero desde el vértice i hasta el vértice n.)

El que exista al menos un camino desde el vértice origen hasta el vértice i, equivale a que, si se toman los predecesores de i, luego los predecesores de los predecesores de i, etc., entonces se acaba llegando al vértice origen (que representa el inicio del proyecto). El que exista al menos un camino desde el vértice i hasta el vértice final, se puede formular análogamente en términos de sucesores.

Frecuentemente, en la tabla de actividades no se dan las actividades ficticias. Así, en el ejemplo, faltarían la primera y la última fila de la tabla. Si entonces dibujáramos el grafo de actividades, el resultado sería el siguiente

¿Cómo nos damos cuenta de que faltan las actividades ficticias? La regla es:

> (i) Todo grafo de actividades ha de tener uno y un solo vértice sin predecesores, cuya duración ha de ser cero. Caso contrario hay que añadir un vértice origen.
>
> (ii) Todo grafo de actividades ha de tener uno y un solo vértice sin sucesores, cuya duración ha de ser cero. Caso contrario hay que añadir un vértice final.



Una vez que sabemos que hay que añadir un vértice origen o un vértice final, ¿cómo completamos el grafo? La regla es:

> (i) Si hay que añadir un vértice origen, se le hace predecesor de los vértices que no tuvieran predecesor.
>
> (ii) Si hay que añadir un vértice final, se le hace sucesor de los vértices que no tuvieran sucesor.

En cuanto a la asignación de peso a los arcos que se han añadido, se aplica el apartado (3) de la definición del grafo de actividades.

En la figura anterior hay dos vértices sin predecesores, el 2 y el 3, con lo que hay que añadir un vértice origen. Por otra parte, hay dos vértices sin sucesores, el 7 y el 9, con lo que hay que añadir un vértice final. Aplicando la regla anterior, se obtiene el grafo de actividades completo que teníamos al principio.

5.2. El CPM

Los dos métodos básicos de planificación de actividades son el *CPM* (*critical path method*, *método del camino crítico*) y el *CPS* (*critical project scheduling*).

La terminología en planificación de actividades es un tanto caótica, pero, independientemente de las denominaciones utilizadas, hay que distinguir los enfoques diferentes de los dos métodos.

Tiempos de comienzo de las actividades

Al iniciarse el proyecto empieza a correr el tiempo. La pregunta que se plantea es: ¿cuándo se puede, y debe, comenzar cada una de las actividades?

Sea una actividad i. Definimos:

- t_i es el tiempo más temprano en que se puede comenzar la actividad i.

Para clarificar el concepto, consideremos la actividad 7 en el grafo de actividades de la figura. ¿Qué actividades han de haber sido completadas antes de poder comenzar la actividad 7? Respuesta: las predecesoras de 7, las predecesoras de las predecesoras de 7, etc. O sea, todas las actividades que estén en un camino desde el vértice origen hasta el vértice 7. En nuestro caso, las actividades 1, 2, 4 y 5. Sólo cuando haya transcurrido el tiempo necesario para realizar estas actividades se puede comenzar la actividad 7. La suma de las duraciones de estas actividades es 0+4+6+2=12 semanas. Sin embargo, la realización de 4 y 5 se pueden realizar a la vez ("en paralelo"), y el lector quedará fácilmente convencido de que la actividad 7 se puede comenzar después de haber transcurrido 10 semanas, y no antes. En consecuencia, t_7 =10. De manera inmediata se puede ver que t_4 =4 y t_5 =4.

En particular, t_n es el tiempo más temprano en que se puede terminar el proyecto. Se denota también $\tau := t_n$. En el ejemplo, como veremos, τ =32 semanas.

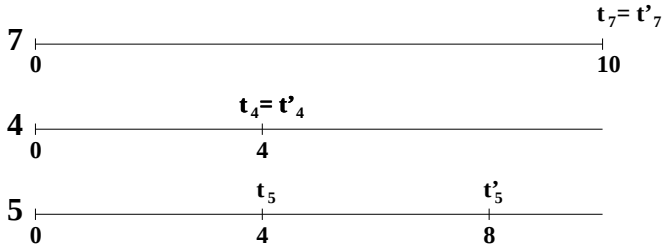
Vamos a definir otro valor temporal para la actividad i.

- t'_i es el tiempo más tardío en que se debe comenzar la actividad i para que no se produzcan retrasos en el tiempo de ejecución del proyecto.

Evidentemente, $t'_i \geq t_i$.

Consideremos los siguientes gráficos de tiempo para las actividades 7, 4 y 5 del ejemplo.

Como veremos, t'_7 =10. Esto quiere decir que, no solo no podemos comenzar la actividad 7 antes (del fin) de la décima semana (t_7 =10), sino que tampoco debemos hacerlo después de la décima semana, porque si no el proyecto no podría concluirse en 32 semanas. (Cuando hablemos de que una actividad se realiza en cierto periodo (o antes de cierto periodo), entenderemos que se realiza *al final* del periodo (o antes *del final* del periodo)).



Consideremos ahora la actividad 5; si en vez de comenzarla en la cuarta semana, la comenzamos en la sexta semana, ¿se producirían retrasos en el inicio de la actividad 7? (No olvidemos que si la actividad 7 se retrasa, se retrasa todo el proyecto, pues $t_7 = t'_7$). La respuesta es negativa: 6+2=8, y la actividad 7 estaría completada antes de la décima semana. Se podría comenzar la actividad 5 hasta en la octava semana (pero no más tarde), sin que se produzcan retrasos en el inicio de la actividad 7. No solo se cumple esto, sino que, como veremos, se puede comenzar la actividad 5 hasta en la octava semana, sin que se produzcan retrasos en la ejecución del proyecto. En consecuencia, t'_5 =8.

En cuanto a la actividad 4, si se comienza más tarde de la cuarta semana, ya no se puede comenzar la actividad 7 en la décima semana, y se retrasaría todo el proyecto. Luego t'_4 =4.

Definimos, por último, la *holgura* de la actividad i, que denotamos por s_i.

- $s_i := t'_i - t_i$

Evidentemente, $s_i \geq 0$.

Se llaman *actividades críticas* aquellas cuya holgura es nula. Es evidente el interés económico que tiene identificarlas. En el ejemplo, ya hemos visto que las actividades 4 y 7 son críticas, y que la 5 no lo es.

El siguiente teorema permite calcular t_i y t'_i hallando caminos más largos desde el vértice origen hasta el vértice i y desde el vértice i hasta el vértice final.

Teorema. Sea un grafo de actividades $G = (X, U, p)$, y sea una actividad $i \in X$. Entonces:

(i) t_i es igual a la longitud de un camino más largo desde el vértice origen hasta el vértice i.

(ii) $t'_i = \tau - l_i$, donde l_i es la longitud de un camino más largo desde el vértice i hasta el vértice final.

El Principio de Bellman

Existen algoritmos eficientes para hallar caminos más largos (o más cortos) en un grafo ponderado, asumiendo hipótesis adecuadas. Uno de los resultados en que se pueden basar estos algoritmos es el llamado *Principio de Bellman*. Este "principio" es una idea intuitiva que cristaliza en teoremas adaptados a diferentes contextos. Su formulación en el caso de caminos más largos en grafos es la siguiente.

Teorema (Principio de Bellman). Sean dos vértices distintos h y j en un grafo ponderado. Supongamos que un camino más largo P desde h hasta j pasa por el vértice i. Entonces el tramo de P entre h e i es un camino más largo desde h hasta i. (Análogamente, el tramo de P entre i y j es un camino más largo desde i hasta j).

Demostración. La prueba es inmediata, por absurdo. Sea P_1 el tramo de P entre h e i, y P_2 el tramo de P entre i y j. Si P_1 no fuera un camino más largo desde h hasta i, existiría un camino P_1' desde h hasta i más largo que P_1. Pero entonces la unión de P_1' y P_2 sería un camino desde h hasta j más largo que P, lo cual va contra la hipotesis.

(Un enunciado análogo se cumple para caminos más cortos). Obsérvese que para aplicar el Principio de Bellman hace falta partir de que existe un camino más largo desde h hasta j.

Tabla del CPM

A partir de los dos teoremas anteriores, se pueden calcular los t_i, t_i' y s_i. Suponemos que los vértices del grafo están etiquetados de manera que todos los arcos vayan de un vértice de etiqueta menor a otro de etiqueta mayor (recordamos que el grafo de actividades es acíclico). Los resultados se suelen disponer en la llamada *tabla del CPM*, o *tabla del camino crítico*, que ilustramos a continuación en el caso del ejemplo.

	1	2	3	4	5	6	7	8	9	10
t_i	0	0	0	4	4	4	10	6	15	32
l_i	32	32	27	28	24	28	22	20	17	0
t_i'	0	0	5	4	8	4	10	12	15	32
s_i	0	0	5	0	4	0	0	6	0	0

Para el cálculo de los t_i, hemos empezado por los vértices más cercanos al origen. En casi todos los casos, hay solo uno, o dos, caminos desde el origen al

vértice i. Supuestos calculados $t_1, t_2, ..., t_8$, para calcular t_9 se razona de la siguiente forma, aplicando el Principio de Bellman. Hay tres posibilidades para los caminos más largos desde el origen hasta el vértice 9:

> (i) Entran en el vértice 9 por el vértice 8 (o sea, el último arco del camino es $(8, 9)$). En este caso la longitud de un camino más largo desde el origen hasta el vértice 9, será la longitud de un camino más largo desde el origen hasta el vértice 8 más la longitud del arco $(8, 9)$ (por el Principio de Bellman). Pero la longitud de un camino más largo desde el origen hasta el vértice 8 ya está calculada (es t_8), y concluimos que la longitud de un camino más largo desde el origen hasta el vértice 9 *que entre por el vértice* 8 es 6+3=9.
>
> (ii) Entran en el vértice 9 por el vértice 6. En este caso la longitud de un camino más largo desde el origen hasta el vértice 9, será la longitud de un camino más largo desde el origen hasta el vértice 6 más la longitud del arco $(6, 9)$. Concluimos que la longitud de un camino más largo desde el origen hasta el vértice 9 *que entre por el vértice* 6 es 4+11=15.
>
> (iii) Entran en el vértice 9 por el vértice 3. En este caso la longitud de un camino más largo desde el origen hasta el vértice 9, será la longitud de un camino más largo desde el origen hasta el vértice 3 más la longitud del arco $(3, 9)$. Concluimos que la longitud de un camino más largo desde el origen hasta el vértice 9 *que entre por el vértice* 3 es 0+10=10.

Sabemos que existe algún camino más largo desde el origen hasta el vértice 9 (queda como ejercicio justificar esto, teniendo en cuenta que estamos en un grafo de actividades). En consecuencia, la longitud de un camino más largo desde el origen hasta el vértice 9 es $t_9 = \text{máx}\{9, 15, 10\} = 15$. Para calcular t_{10} vamos a razonar de forma análoga. Hay dos posibilidades para los caminos más largos desde el origen hasta el vértice 10:

> (i) Entran en el vértice 10 por el vértice 7. En este caso la longitud de un camino más largo desde el origen hasta el vértice 10, será la longitud de un camino más largo desde el origen hasta el vértice 7 más la longitud del arco $(7, 10)$. Concluimos que la longitud de un camino más largo desde el origen hasta el vértice 10 *que entre por el vértice* 7 es t_7+22=32.

(ii) Entran en el vértice 10 por el vértice 9. En este caso la longitud de un camino más largo desde el origen hasta el vértice 10, será la longitud de un camino más largo desde el origen hasta el vértice 9 más la longitud del arco $(9, 10)$. Concluimos que la longitud de un camino más largo desde el origen hasta el vértice 10 *que entre por el vértice* 9 es 15+17=32.

En consecuencia, la longitud de un camino más largo desde el origen hasta el vértice 10 es $t_{10} = 32$.

Para el cálculo de los l_i, empezamos por los vértices más cercanos al vértice final. En casi todos los casos, hay solo uno, o dos, caminos desde el vértice i al final. Supuestos calculados $l_{10}, l_9, ..., l_3$, para calcular l_2 se razona de la siguiente forma, aplicando el Principio de Bellman. Hay tres posibilidades para los caminos más largos desde vértice 2 hasta el final:

(i) Salen del vértice 2 por el vértice 4 (o sea, el primer arco del camino es $(2, 4)$). En este caso la longitud de un camino más largo desde vértice 2 hasta el final, será la longitud del arco $(2, 4)$ más la longitud de un camino más largo desde el vértice 4 hasta el final (por el Principio de Bellman). Pero la longitud de un camino más largo desde el vértice 4 hasta el final ya está calculada (es l_4), y concluimos que la longitud de un camino más largo desde el vértice 2 hasta el final *que salga por el vértice* 4 es $4+l_4$=32.

(ii) Salen del vértice 2 por el vértice 5. En este caso la longitud de un camino más largo desde vértice 2 hasta el final, será la longitud del arco $(2, 5)$ más la longitud de un camino más largo desde el vértice 5 hasta el final. Concluimos que la longitud de un camino más largo desde el vértice 2 hasta el final *que salga por el vértice* 5 es 4+24=28.

(iii) Salen del vértice 2 por el vértice 6. En este caso la longitud de un camino más largo desde vértice 2 hasta el final, será la longitud del arco $(2, 6)$ más la longitud de un camino más largo desde el vértice 6 hasta el final. Concluimos que la longitud de un camino más largo desde el vértice 2 hasta el final *que salga por el vértice* 6 es 4+28=32.

Sabemos que existe algún camino más largo desde el vértice 2 hasta el final. En consecuencia, la longitud de un camino más largo desde el vértice 2 hasta el final es $l_2 = \text{máx}\{32, 28\} = 32$. Para calcular l_1 se razona de forma análoga.

Caminos críticos

Se llama *camino crítico* a un camino más largo desde el vértice origen al vértice final.

En todo grafo de actividades existe al menos un camino crítico, aunque puede existir más de uno. Todo camino crítico tiene longitud τ. Los caminos críticos están estrechamente ligados a las actividades críticas, como indica el siguiente teorema.

Teorema. Sea un grafo de actividades $G = (X, U, p)$. Entonces una actividad es crítica si, y sólo si, está en algún camino crítico.

El *análisis del CPM*, o *análisis del camino crítico*, consiste en hallar la tabla del CPM y todos los caminos críticos.

Vamos a hallar todos los caminos críticos en el ejemplo, con ayuda del teorema anterior. Las actividades críticas son: 1, 2, 4, 6, 7, 9 y 10. Puesto que los caminos críticos solo pueden atravesar actividades críticas, todos los caminos críticos han de empezar por

$$1, 2, 4$$

o, en una segunda posibilidad, por

$$1, 2, 6$$

En la primera posibilidad obtenemos como candidato

$$1, 2, 4, 7, 10$$

que efectivamente es un camino crítico, pues su longitud es 32. De acuerdo al teorema, todavía nos faltan caminos críticos, pues las actividades críticas 6 y 9 han de estar en algún camino crítico. En la segunda posibilidad resulta

$$1, 2, 6, 9, 10$$

cuya longitud es 32, y es, por tanto, otro camino crítico. Ahora todas las actividades críticas están ya en algún camino crítico, y damos por concluida la búsqueda (si hubiera otros caminos críticos, que no es el caso, ya no nos interesarían, ya que están cubiertas todas las actividades críticas).

El CPM como problema de optimización lineal

El resultado del CPM son $(t_1, t_2, ..., t_n)$ y $(t'_1, t'_2, ..., t'_n)$, que corresponden a *tiempos óptimos* de comienzo de las actividades. Esto es, si T_i es el tiempo de

comienzo de la actividad i, $i = 1, 2, ..., n$, entonces $t_i \leq T_i \leq t'_i$ cuando el tiempo de terminación de todo el proyecto es el óptimo (el menor posible), τ.

Vamos a formular el CPM como problema de optimización, cuyas variables son las T_i. La restricción de inicialización es obviamente $T_1 = 0$ ¿Cómo escribimos las restricciones de precedencia? Supongamos que $(i, j) \in U$, o sea que la actividad i preceda a la actividad j. En términos de tiempos esto se puede formular:

$$T_j \geq T_i + \tau_i$$

En palabras: el tiempo de comienzo de la actividad j ha de ser posterior al tiempo de comienzo de la actividad i más la duración de la actividad i. (Por ejemplo, el que la actividad 4 preceda a la actividad 7 se escribiría $T_7 \geq T_4 + 6$) ¿Qué hemos de minimizar? El tiempo de ejecución de todo el proyecto, T_n.

La formulación del CPM como problema de optimización es:

$$\begin{array}{lll} \text{(CPM)} & Min & T_n \\ & s.a & T_j \geq T_i + \tau_i \ , \ \forall (i,j) \in U \\ & & T_1 = 0 \\ & & [(T_1, T_2, ..., T_n) \in \mathbb{R}^n] \end{array}$$

El problema de optimización (CPM) siempre tiene solución, y su conjunto óptimo resulta ser $\mathcal{O} = \{(T_1, T_2, ..., T_n) \in \mathbb{R}^n : t_i \leq T_i \leq t'_i, \ i = 1, 2, ..., n\} = [t_1, t'_1] \times [t_2, t'_2] \times ... \times [t_n, t'_n]$.

Evidentemente (CPM) es un problema de optimización lineal, que se resuelve mediante una variante especializada del algoritmo del símplex, que además no proporciona una mera solución óptima, sino precisamente $(t_1, t_2, ..., t_n)$ y $(t'_1, t'_2, ..., t'_n)$. La resolución de (CPM) como problema de optimización lineal es un procedimiento alternativo al ya visto de obtención de caminos más largos para resolver el CPM.

5.3. El CPS

El problema y su resolución

Es importante comparar el CPM con el CPS. Recordemos los elementos del CPM:

Datos. (X, U): grafo
τ_i: duración de actividad i, $i = 1, 2, ..., n$

Variables. T_i: tiempo de comienzo de la actividad i, $i = 1, 2, ..., n$

Objetivo. Minimizar el tiempo de ejecución del proyecto

Problema del CPM. $Min \quad T_n$

$s.a \quad T_j \geq T_i + \tau_i \ , \ \forall (i,j) \in U$

$T_1 = 0$

El CPS supone un planteamiento del problema distinto al del CPM. Las dos diferencias principales son:

(i) En el CPM la duración de las actividades es un dato, mientras que en el CPS es una variable del problema. Para cada actividad i, hay una cota inferior m_i y una cota superior M_i de la duración de la actividad, con $0 \leq m_i \leq M_i$.

(ii) En el CPM el objetivo es minimizar el tiempo de ejecución del proyecto, mientras que en el CPS el tiempo de ejecución del proyecto está meramente sujeto a una cota superior T, con $T \geq 0$. En el CPS aparece un nuevo elemento, el costo de cada actividad, y el objetivo es minimizar el costo total. El costo de cada actividad depende de su duración τ_i (que es ahora variable), y es $(\alpha_i - \beta_i \tau_i)$, siendo los coeficientes de costo dados $\alpha_i, \beta_i \in \mathbb{R}$, $\beta_i > 0$. (Cuanto más rápido se ejecuta la actividad, resulta más cara.)

Podemos ya definir el CPS:

Datos. (X, U): grafo
m_i, M_i: cotas de la duración de actividad i, $i = 1, 2, ..., n$
α_i, β_i: coeficientes de costo de la actividad i, $i = 1, 2, ..., n$
T: cota superior de la duración del proyecto

Variables. T_i: tiempo de comienzo de la actividad i, $i = 1, 2, ..., n$
τ_i: duración de actividad i, $i = 1, 2, ..., n$

Objetivo. Minimizar el costo total

Problema del CPS. $Min \quad \sum_{i=1}^{n} (\alpha_i - \beta_i \tau_i)$

$s.a \quad T_j \geq T_i + \tau_i \ , \ \forall (i,j) \in U$

$T_1 = 0$

$m_i \leq \tau_i \leq M_i \ , \ i = 1, 2, ..., n$

$T_n \leq T$

El problema del CPS es un problema de optimización lineal, que se resuelve mediante una variante especializada del algoritmo del símplex. No tiene por qué ser factible, aunque si es factible tiene solución óptima. Si tiene una solución óptima $(\overline{T}_1, \overline{T}_2, ..., \overline{T}_n, \overline{\tau}_1, \overline{\tau}_2, ..., \overline{\tau}_n)$, entonces $\overline{\tau}_1, \overline{\tau}_2, ..., \overline{\tau}_n$ son duraciones óptimas de las actividades.

Aplicación interactiva del CPM y el CPS

En la práctica, es frecuente aplicar tanto el CPM como el CPS, a través de un "diálogo" entre el decisor y la máquina (esto es, de manera interactiva).

Partimos de los datos del CPS, y fijamos las duraciones de las actividades en el nivel "ahorrativo" $\tau_i = M_i$. Aplicamos con estos valores fijos el CPM, y obtenemos un tiempo de ejecución del proyecto τ. Si el decisor considera aceptable este valor de τ, el proceso termina (y utilizamos los valores de $(t_1, t_2, ..., t_n)$ y $(t'_1, t'_2, ..., t'_n)$ obtenidos en el CPM). Si el decisor considera excesivo este valor de τ, se procede al *acortamiento* ("*crushing*"): el decisor fija prudencialmente una cota superior T del tiempo de ejecución del proyecto, con $T < \tau$, y se aplica el CPS (obviamente con duraciones variables de las actividades). Si el CPS es factible y el costo que da es aceptable (a juicio del decisor), el proceso termina: el CPS nos ha dado las duraciones óptimas, y con ellas se aplica el CPM para obtener los correspondientes $(t_1, t_2, ..., t_n)$ y $(t'_1, t'_2, ..., t'_n)$. Caso contrario, si el CPS no es factible o el costo que da no es aceptable, el decisor fija prudencialmente una nueva cota superior T' del tiempo de ejecución del proyecto, con $T' > T$, y se aplica el CPS... El proceso se continúa hasta que el decisor quede satisfecho.

6. Ejercicios sobre optimización

1. Sea el problema de optimización lineal:

$$\begin{array}{lrrrrrl} Min & -45x_1 & -30x_2 & & & & \\ & 2x_1 & +10x_2 & +x_3 & & & = 60 \\ s.a & 6x_1 & +6x_2 & & +x_4 & & = 60 \\ & 10x_1 & +5x_2 & & & +x_5 & = 85 \\ & x \geq 0 & & & & & \end{array}$$

¿Es factible la base $I = (3, 4, 5)$, $J = (1, 2)$? Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (3, 4, 1)$, $J = (5, 2)$. ¿Qué vector de $\mathbb{R}^5$ se obtiene, en su caso, como solución factible básica tras la iteración anterior? ¿Qué dirección de mejora se ha utilizado?

SOLUCIÓN

Sí es factible $I = (3, 4, 5)$, $J = (1, 2)$. A partir de $I = (3, 4, 1)$, $J = (5, 2)$:
$c^N - c^B B^{-1} N = (4{,}5, -7{,}5)$; $k = 2$; $\Delta = 3$; $l = 2$;

$$\overline{x} = \begin{pmatrix} 7 \\ 3 \\ 16 \\ 0 \\ 0 \end{pmatrix}, d = \begin{pmatrix} -0{,}5 \\ 1 \\ -9 \\ -3 \\ 0 \end{pmatrix}$$

2. Sea el problema de optimización lineal:

$$\begin{array}{lllllll} Min & -x_1 + 2x_2 & & & & & \\ s.a & x_1 & -4x_2 & +x_3 & & = 1 \\ & -3x_1 & +2x_2 & & +x_4 & = 6 \\ & x \geq 0 & & & & & \end{array}$$

Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (3, 4)$, $J = (1, 2)$. ¿Qué vector de $\mathbb{R}^4$ se obtiene, en su caso, como solución factible básica tras la iteración anterior?

SOLUCIÓN

$c^N - c^B B^{-1} N = (-1, 2)$; $k = 1$; $q = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$; $\Delta = 1$; $l = 1$; $\overline{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 9 \end{pmatrix}$

3. Sea el problema de optimización lineal:

$$\begin{array}{lllllll} Min & -2x_3 - 2x_4 - 2x_5 & & & & & \\ & x_1 & & -x_4 & +3x_5 & = 21 \\ s.a & & x_2 & +2x_4 & -3x_5 & = 6 \\ & & & x_3 & +x_4 & +x_5 & = 1 \\ & x \geq 0 & & & & & \end{array}$$

Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (1, 2, 5)$, $J = (4, 3)$. ¿Qué vector de $\mathbb{R}^5$ se obtiene, en su caso, como solución factible básica tras la iteración anterior?

SOLUCIÓN

$c^N - c^B B^{-1} N = (0,0)$; $\overline{x} = \begin{pmatrix} 18 \\ 9 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ es solución óptima

4. Sea el problema de optimización lineal:

$$\begin{array}{lrrrrrl} Min & & -3x_3 & -2x_4 & -2x_5 & & \\ & x_1 & & +3x_3 & -x_4 & +3x_5 & = 21 \\ s.a & & x_2 & & +2x_4 & -3x_5 & = -3 \\ & & & x_3 & +x_4 & +x_5 & = 1 \\ & x \geq 0 & & & & & \end{array}$$

Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (1,2,5)$, $J = (4,3)$. ¿Qué vector de $\mathbb{R}^5$ se obtiene, en su caso, como solución factible básica tras la iteración anterior?

SOLUCIÓN

$\overline{x} = \begin{pmatrix} 18 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ (inicial); $c^N - c^B B^{-1} N = (0,-1)$; $k = 2$; $q = \begin{pmatrix} 0 \\ -3 \\ -1 \end{pmatrix}$; $\Delta = 0$; $l = 2$; $\overline{x} = \begin{pmatrix} 18 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ (final)

5. Sea el problema de optimización lineal:

$$\begin{array}{lrrrl} Min & -2x_1 & -x_2 & -2x_3 & \\ & 4x_1 & +x_2 & & = 12 \\ s.a & -2x_1 & & +x_3 & = 4 \\ & x \geq 0 & & & \end{array}$$

Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (2,3)$, $J = 1$. ¿Qué vector de $\mathbb{R}^3$ se obtiene, en su caso, como solución factible básica tras la iteración anterior? Exprésese el problema de optimización con las

variables básicas despejadas a partir de las no básicas, respecto a la base inicial; idem respecto a la base siguiente: $I = (2, 1)$, $J = 3$.

SOLUCIÓN

$c^N - c^B B^{-1} N = -2$; $k = 1$; $q = \begin{pmatrix} -4 \\ 2 \end{pmatrix}$; $\Delta = 3$; $l = 1$; $\overline{x} = \begin{pmatrix} 3 \\ 0 \\ 10 \end{pmatrix}$

La reformulación pedida del problema (respecto a $I = (2, 1)$, $J = 3$) es:

$$\begin{array}{lll} Min & -16 - x_3 & \\ s.a & x_2 = 20 - 2x_3 & \\ & x_1 = -2 + \frac{1}{2}x_3 & \\ & x_3 \leq 10 & \\ & x_3 \geq 4 & \end{array}$$

6. Sea el problema de optimización lineal:

$$\begin{array}{lllll} Min & -2x_1 - x_2 - 2x_3 & & & \\ s.a & -4x_1 + x_2 & & = 12 & \\ & -2x_1 & + x_3 & = 4 & \\ & x \geq 0 & & & \end{array}$$

Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (2, 3)$, $J = 1$. ¿Qué vector de $\mathbb{R}^3$ se obtiene, en su caso, como solución factible básica tras la iteración anterior?

SOLUCIÓN

$c^N - c^B B^{-1} N = -10$; $k = 1$; $q = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$; $\Delta = \infty$; problema no acotado

7. Sea el problema de optimización lineal:

$$\begin{array}{lllllll} Min & -x_4 - 2x_5 & & & & & \\ & x_1 & & -x_4 & +3x_5 & = 21 & \\ s.a & & x_2 & +2x_4 & -3x_5 & = 6 & \\ & & & x_3 + x_4 & +x_5 & = 1 & \\ & x \geq 0 & & & & & \end{array}$$

Realícese una iteración completa del algoritmo del símplex, partiendo de la base $I = (1, 2, 3)$, $J = (4, 5)$. ¿Qué vector de $\mathbb{R}^5$ se obtiene, en su caso, como solución factible básica tras la iteración anterior?

SOLUCIÓN

$c^N - c^B B^{-1} N = (-1, -2)$; $k = 2$ (elegimos); $q = \begin{pmatrix} -3 \\ 3 \\ -1 \end{pmatrix}$; $\Delta = 1$; $l = 3$;

$$\overline{x} = \begin{pmatrix} 18 \\ 9 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

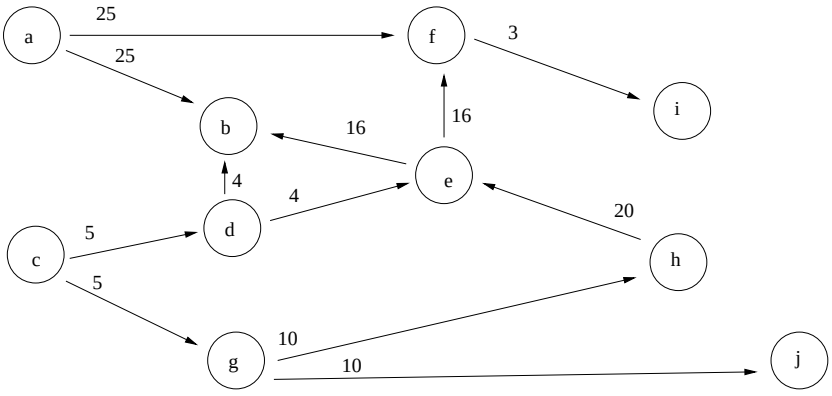
8. Considérese un proyecto con la siguiente tabla de actividades:

actividad	a	b	c	d	e	f	g	h	i	j
duración	25	14	5	4	16	3	10	20	2	6
predecesores	-	a,d,e	-	c	d,h	a,e	c	g	f	g

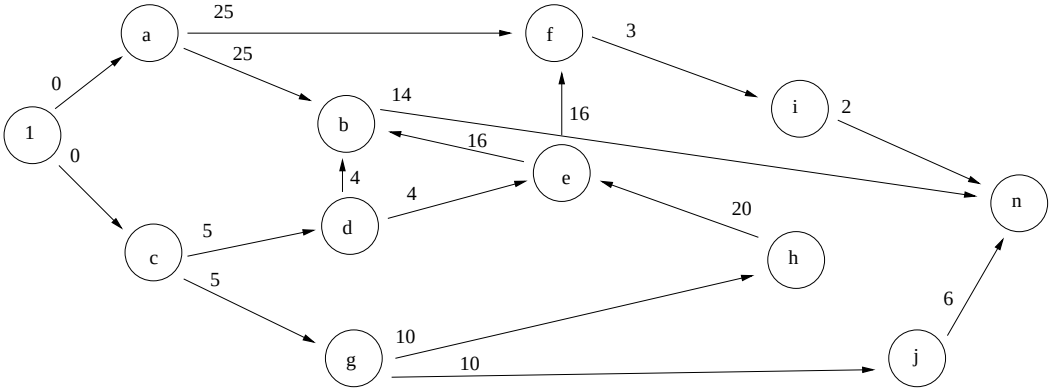
Dibújese el grafo de actividades y hágase el análisis completo del camino crítico.

SOLUCIÓN

Hay que añadir las actividades de "empezar" (1) y "terminar" (n). Antes de añadirlas el grafo aparece así:



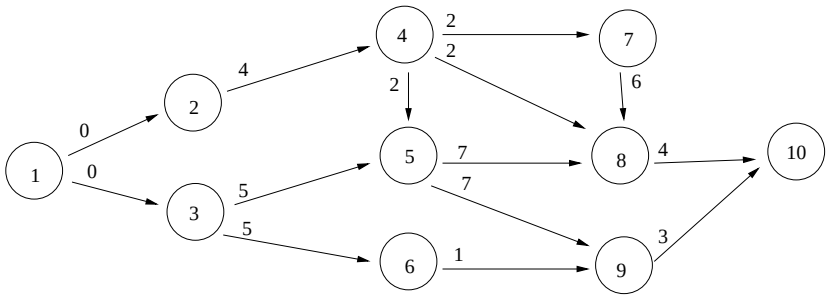
Después de añadirlas, el grafo de actividades es:



actividad	1	a	b	c	d	e	f	g	h	i	j	n
t_i	0	0	51	0	5	35	51	5	15	54	15	65
t'_i	0	26	51	0	31	35	60	5	15	63	59	65
s_i	0	26	0	0	26	0	9	0	0	9	44	0

Hay 1 camino crítico: 1, c, g, h, e, b, n

9. Considérese el siguiente grafo de actividades:



Hállese un camino crítico. El coste extra por unidad de tiempo del acortamiento de la duración de las actividades 2,...,9 es, respectivamente: 5.1, 2, 5.3, 4, 7, 3.2, 2.1, 3 (en miles de euros). Plantéese el problema matemático de cómo acortar óptimamente la duración del proyecto en 3 unidades de tiempo, suponiendo que la duración de las actividades puede ser reducida, como máximo, el 25 %.

SOLUCIÓN

Un camino crítico: 1, 2, 4, 5, 8, 10. En consecuencia τ=17.

Problema de optimización:

$$
\begin{array}{ll}
Min & -(5{,}1\tau_2 + 2\tau_3 + 5{,}3\tau_4 + 4\tau_5 + 7\tau_6 + 3{,}2\tau_7 + 2{,}1\tau_8 + 3\tau_9) \\
s.a & T_2 \geq T_1 + \tau_1 \\
 & T_3 \geq T_1 + \tau_1 \\
 & T_4 \geq T_2 + \tau_2 \\
 & T_5 \geq T_3 + \tau_3 \\
 & T_5 \geq T_4 + \tau_4 \\
 & T_6 \geq T_3 + \tau_3 \\
 & T_7 \geq T_4 + \tau_4 \\
 & T_8 \geq T_4 + \tau_4 \\
 & T_8 \geq T_5 + \tau_5 \\
 & T_8 \geq T_7 + \tau_7 \\
 & T_9 \geq T_5 + \tau_5 \\
 & T_9 \geq T_6 + \tau_6 \\
 & T_{10} \geq T_8 + \tau_8 \\
 & T_{10} \geq T_9 + \tau_9 \\
 & T_1 = 0
\end{array}
\qquad
\begin{array}{l}
\\ \\
\tau_1 = 0 \\
3 \leq \tau_2 \leq 4 \\
3{,}75 \leq \tau_3 \leq 5 \\
1{,}5 \leq \tau_4 \leq 2 \\
5{,}25 \leq \tau_5 \leq 7 \\
0{,}75 \leq \tau_6 \leq 1 \\
4{,}5 \leq \tau_7 \leq 6 \\
3 \leq \tau_8 \leq 4 \\
2{,}25 \leq \tau_9 \leq 3 \\
\tau_{10} = 0 \\
T_{10} \leq 14
\end{array}
$$

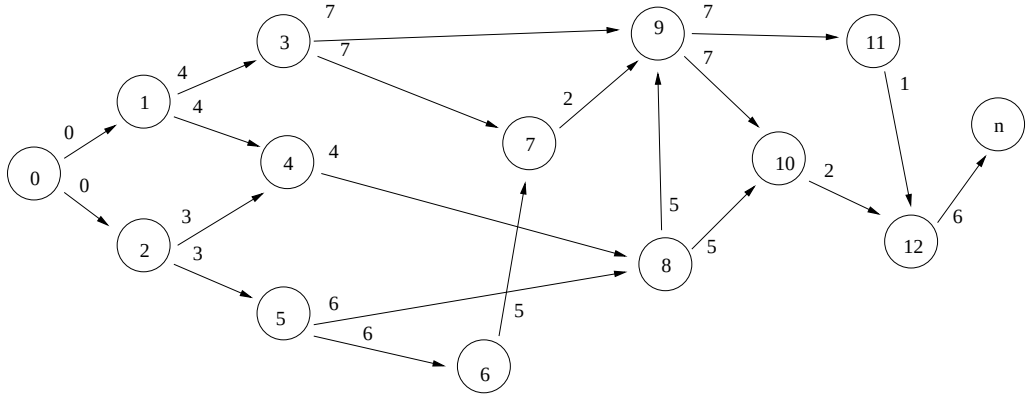
10. Considérese un proyecto con la siguiente tabla de actividades:

actividad	1	2	3	4	5	6	7	8	9	10	11	12
duración	4	3	7	4	6	5	2	5	7	2	1	6
predecesores	-	-	1	1,2	2	5	3,6	4,5	3,7,8	8,9	9	10,11

Dibújese el grafo de actividades y hágase el análisis completo del camino crítico.

SOLUCIÓN

Hay que añadir las actividades de "empezar" (0) y "terminar" (n).



actividad	0	1	2	3	4	5	6	7	8	9	10	11	12	n
t_i	0	0	0	4	4	3	9	14	9	16	23	23	25	31
t'_i	0	3	0	7	7	3	9	14	11	16	23	24	25	31
s_i	0	3	0	3	3	0	0	0	2	0	0	1	0	0

Hay 1 camino crítico: 0, 2, 5, 6, 7, 9, 10, 12, n

11. Considérese un proyecto con la siguiente tabla de actividades:

actividad	1	2	3	4	5	6	7	8	9	10	11
duración	6	2	10	1	4	2	7	7	9	2	4
predecesores	-	-	1	1	1	5	2,4	3,6	2,4	7	8,10

Hállese el tiempo necesario para realizar el proyecto. Si la duración de cada actividad puede acortarse como máximo en dos unidades de tiempo (la cuarta en una unidad), y el coste extra por unidad acortada es de 5 euros, plantéese el problema matemático de cómo acortar óptimamente la duración del proyecto en un 30 %.

A. APÉNDICE: RECORDATORIO DE MATEMÁTICA DISCRETA

1. Algo de Aritmética

1.1. Números

En este apéndice bosquejaremos, de manera muy esquemática, algunos elementos básicos de matemática discreta.

Conjuntos de números:

Números naturales $\mathbb{N} := \{0, 1, 2, 3, ...\}$

(Notación: $\mathbb{N}^* := \{1, 2, 3, ...\}$)

Números enteros $\mathbb{Z} := \{0, 1, -1, 2, -2, 3, -3, ...\}$

Números racionales $\mathbb{Q} := \left\{ \frac{p}{q} : p \in \mathbb{Z}, q \in \mathbb{N}^* \right\}$

Números reales $\mathbb{R}$

Números complejos $\mathbb{C}$

Recta real:

Si representamos $\mathbb{N}$, $\mathbb{Z}$ y $\mathbb{Q}$ en una recta, resulta que los números racionales no la llenan. Los números reales, sí. La matemática de los números reales es una "matemática del continuo", y ello permite el cálculo infinitesimal habitual. En contraste, los números naturales (y los enteros) están separados por saltos regulares. La "matemática discreta" trabaja solo con números enteros. (Del verbo latino *discerno, discernere, discrevi, discretum.*)

Todas las propiedades de los números naturales se pueden deducir a partir de unos pocos enunciados llamados "axiomas de Peano". Hoy existen axiomas generales para toda la matemática, los axiomas de la teoría de conjuntos, y los axiomas de Peano ya no son "axiomas", ya que se pueden deducir de estos axiomas generales. A partir de $\mathbb{N}$, se pueden construir los demás conjuntos de números, hasta $\mathbb{R}$ y $\mathbb{C}$, y deducir sus propiedades. Así pues, a partir de los axiomas de Peano se puede deducir todo el cálculo infinitesimal. ¿Por qué es esto posible?

Uno de los axiomas de la teoría de conjuntos es el "axioma de infinitud", que postula que existe un conjunto infinito; el poder de este axioma es enorme.

1.2. Combinatoria

La combinatoria es el arte de contar. Al contar se establece una correspondencia biyectiva con un conjunto patrón. Los niños toman como conjuntos patrones subconjuntos del conjunto de los dedos de las manos (sistema decimal: diez dedos), un conjunto material y finito. Nosotros tomamos como conjuntos patrones subconjuntos de $\mathbb{N}$, un conjunto infinito y abstracto.

(Notación: $\overline{n} := \{1, 2, ..., n\}$ para $n \in \mathbb{N}^*$)

Sea A un conjunto. Hay tres posibilidades excluyentes:

(1) $A = \emptyset$

(2) $\exists f : \overline{n} \to A$ biyectiva, para cierto $n \in \mathbb{N}^*$

(3) ninguna de las anteriores

En cada uno de estos casos definimos el numero de elementos de A, denotado $|A|$:

(1) $|A| := 0$

(2) $|A| := n$

(3) $|A| := \infty$

En los casos (1) y (2) se dice que el conjunto es finito.

Para contar, como muchas veces en matemáticas, acudiremos frecuentemente al método analítico: reduciremos la resolución de un problema complejo, que no sabemos resolver en principio, a la resolución de problemas más sencillos, que sí sabemos resolver.

Sean A y B conjuntos finitos. Entonces se tiene:

(i) $A \cap B = \emptyset \Rightarrow |A \cup B| = |A| + |B|$

(ii) $|A \times B| = |A|\,|B|$

Sea A un conjunto con $|A| = n, n \in \mathbb{N}^*$, y $p \in \mathbb{N}^*$.

Una *p-tupla*, o tupla de p elementos, de A es, intuitivamente, una lista (ordenada) de p elementos de A (se pueden repetir elementos). Formalmente, es una aplicación $f : \overline{p} \to A$. La función

$$\begin{aligned} f \;\; : \;\; & \overline{p} \to A \\ & 1 \to a_1 \\ & 2 \to a_2 \end{aligned}$$

$$\cdots\cdots\cdots$$
$$p \to a_p$$

se denota $(a_1, a_2, ..., a_p)$.

Una *p-permutación* de A, o *variación* de elementos de A tomados de p en p, es una p-tupla inyectiva. Intuitivamente, es una lista (ordenada) de p elementos de A en la que no se repite ningún elemento. Evidentemente, ha de ser $p \leq n$. Si $p = n$ se habla simplemente de *permutación*.

Una *p-combinación* de A, o combinación de elementos de A tomados de p en p, es un subconjunto de A con p elementos. Intuitivamente, es una lista no ordenada de p elementos de A (en la que no se repite ningún elemento). Evidentemente, ha de ser $p \leq n$. La notación será del tipo $\{a_1, a_2, ..., a_p\}$.

En las tuplas y variaciones afecta el orden; en las combinaciones, no. En las tuplas puede haber repeticiones; en las variaciones y combinaciones, no.

Denotamos A^p el conjunto de todas las p-tuplas de A. El número de todas las p-tuplas de elementos de A es

$$|A^p| = n^p$$

Si $p \leq n$, el número de todas las p-permutaciones de elementos de A es

$$n(n-1) \cdot ... \cdot (n-p+1) = \frac{n!}{(n-p)!}$$

En particular, el número de permutaciones es $n!$

Si $p \leq n$, el número de todas las p-combinaciones de elementos de A es el de p-permutaciones partido por el factorial de p, o sea

$$\frac{n(n-1) \cdot ... \cdot (n-p+1)}{p!} = \frac{n!}{(n-p)!p!} =: \binom{n}{p}$$

1.3. Enumeración

Intuitivamente, una *enumeración* de los elementos de un conjunto A es una lista (ordenada) de todos sus elementos (sin repetir ninguno). O sea, que enumeración y permutación es lo mismo. Cuando hablamos de enumerar un conjunto, se trata de encontrar alguna enumeración suya.

Vamos a intentar enumerar el conjunto A^p, suponiendo que A está ya enumerado, $A = \{a_1, a_2, ..., a_n\}$, siendo $n \in \mathbb{N}^*$ y $p \in \mathbb{N}^*$. Lo vamos a hacer en el llamado orden lexicográfico.

Intuitivamente, el orden lexicográfico es el que utilizamos para encontrar palabras en el diccionario. Sea p la longitud de la palabra más larga en castellano. Entonces las palabras son p-tuplas del conjunto $A = \{\square, a, b, c, ..., z\}$. Aquí $\square$ representa el espacio en blanco, y lo colocamos en el primer lugar de la enumeración de A. Así, en orden lexicográfico, aparecerán en este orden las tres palabras siguientes:

(1) PEPE
(2) PEPITO
(3) PEPOTE

¿Cuál es la regla que estamos aplicando?

Aplicando por analogía la misma regla, enumeramos todas las ternas de A, siendo ahora $A = \{1, 2, 3, 4, 5\}$ (con la enumeración natural), en orden lexicográfico:

$(1,1,1)$	$(1,2,5)$	$(2,5,5)$
$(1,1,2)$	$(1,3,1)$	$(3,1,1)$
$(1,1,3)$	$(1,3,2)$	$(3,1,2)$
.....	.....	.....
$(1,1,5)$	$(1,5,5)$	$(4,5,5)$
$(1,2,1)$	$(2,1,1)$	$(5,1,1)$
$(1,2,2)$	$(2,1,2)$	.....
.....	.....	$(5,5,5)$

En el tema siguiente volveremos a la enumeración lexicográfica, de manera más rigurosa.

2. Teoría de grafos

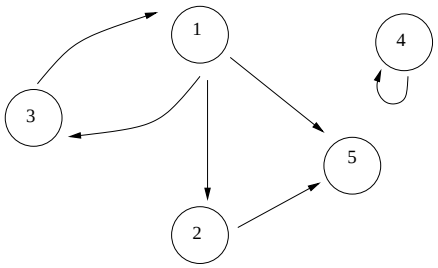
2.1. Grafos

Intuitivamente, un *grafo* es un dibujo en el que aparecen puntos y flechas que los unen.

A los puntos se les llama *nodos* o *vertices*; a las flechas, *arcos*. Matemáticamente, un grafo G está formado por un conjunto X, que corresponde a los vértices, y un subconjunto U del producto cartesiano $X \times X$, que corresponde a los arcos; se denota $G = (X, U)$. Normalmente se asume además que X es finito, cosa que haremos nosotros.

En el ejemplo de la figura,

$$X = \{1, 2, 3, 4, 5\}\,,\ U = \{(1,3), (3,1), (1,2), (1,5), (2,5), (4,4)\}\,.$$



Todo arco va desde un vértice inicial a un vértice final. Los arcos de la forma (x, x), como aquí lo es el $(4, 4)$, se llaman *bucles*.

Las definiciones de "Grafo" y "relación binaria" coinciden, al menos si se consideran conjuntos finitos. En consecuencia, todas las definiciones sobre relaciones binarias son aplicables a los grafos. Por ejemplo, tenemos las definiciones:

Grafo reflexivo (relación binaria reflexiva), cuando $(x, x) \in U$, $\forall x \in X$ (o sea, todos los vértices tienen un bucle).

Grafo irreflexivo (relación binaria irreflexiva), cuando $(x, x) \notin U$, $\forall x \in X$ (o sea, no hay ningún bucle).

Grafo simétrico (relación binaria simétrica), cuando
$(x, y) \in U \Rightarrow (y, x) \in U$, $\forall x, y \in X$.

2.2. Definiciones

Sea un grafo $G = (X, U)$.

Un *camino* es una sucesión finita de arcos de manera que el vértice final de cada arco coincida con el vértice inicial del arco siguiente en la sucesión (con excepción del último arco).

Los caminos quedan unívocamente determinados si se dan los vértices que atraviesan, en su orden. Por ejemplo, en el grafo de la figura, $(3, 1), (1, 2), (2, 5)$ es un camino; se puede alternativamente hablar del camino $3, 1, 2, 5$. Los caminos van desde un vértice inicial (aquí, el 3) hasta un vértice final (el 5). Un *ciclo* es un camino en el que coinciden el vértice inicial y el vértice final.

Un camino es *simple* si en él no se repite ningún arco, y *elemental* si no se repite ningún vértice. Todo camino elemental es simple, pero el recíproco no es cierto.

Sea un vértice $x \in X$. Los *sucesores* de x son aquellos vértices que son vértices finales de los arcos que salen de x. El conjunto de todos los sucesores de x es:

$$\Gamma^+(x) := \{z \in X : (x, z) \in U\}$$

Los *predecesores* de x son aquellos vértices que son vértices iniciales de los arcos que entran en x. El conjunto de todos los predecesores de x es:

$$\Gamma^-(x) := \{z \in X : (z, x) \in U\}$$

En el ejemplo de la figura, $\Gamma^+(1) = \{2, 3, 5\}$, $\Gamma^+(5) = \emptyset$, $\Gamma^-(1) = \{3\}$.

El *extragrado* de x es el número de sus sucesores. Lo denotamos $\rho^+(x) := |\Gamma^+(x)|$. El *intragrado* de x es el número de sus predecesores. Lo denotamos $\rho^-(x) := |\Gamma^-(x)|$. En el ejemplo, $\rho^+(1) = 3$, $\rho^+(5) = 0$, $\rho^-(1) = 1$.

Incidencia. Un vértice y un arco son *incidentes* si el vértice es uno de los extremos del arco.

Adyacencia. Dos *vértices* son *adyacentes* si son extremos del mismo arco. Dos *arcos* son *adyacentes* si tienen un extremo en común. En el ejemplo, los arcos $(1, 2)$ y $(2, 5)$ son adyacentes, y también los son $(1, 3)$ y $(1, 5)$.

Un *grafo parcial* de $G = (X, U)$ es otro grafo $G' = (X, U')$ tal que $U' \subseteq U$. O sea, G' se obtiene suprimiento arcos de G (manteniéndose el mismo conjunto de vértices). Todo grafo es grafo parcial de sí mismo.

Un *subgrafo* de $G = (X, U)$ es otro grafo $G' = (X', U')$ tal que $X' \subseteq X$ y que $U' \subseteq U$ es el resultado de suprimir de U los arcos incidentes con vértices de $(X \sim X')$. Todo grafo es subgrafo de sí mismo.

¿Sería formalizable matemáticamente mediante el concepto de grafo un dibujo como el de la figura, pero en el que hubiera además una segunda flecha desde 2 hasta 5? Ciertamente no; hace falta introducir un nuevo concepto matemático, el de *multigrafo*, para formalizar el dibujo. No lo haremos aquí.

2.3. Grafos simples

Un *grafo simple* es un grafo simétrico e irreflexivo.

Los grafos simples se utilizan normalmente para formalizar conceptos *no dirigidos*: Jorgito y Jaimito son hermanos, y esta relación no es direccional, no va desde Jorgito hasta Jaimito, en ese orden, sino que es *entre* Jorgito y Jaimito.

Sea $G = (X, U)$ un grafo simple. Entonces se tiene:

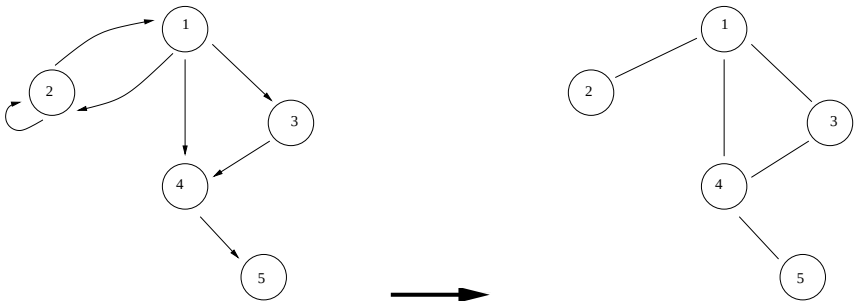
$$(a, b) \in U \Rightarrow (b, a) \in U$$

Definimos entonces la *arista* $[a, b] := \{(a, b), (b, a)\}$. Una arista no tiene vértice inicial ni vértice final, solo extremos: $[a, b]$ es la arista entre a y b. En un grafo simple lo que nos interesa normalmente son las aristas, no los arcos en sí; éstas se suelen dibujar mediante una linea, sin puntas de flecha, entre sus extremos. Si G es un grafo simple, lo enunciaremos habitualmente $G = (X, E)$, donde E es el conjunto de aristas. Si U es su conjunto de arcos, se tendrá que $|U| = 2\,|E|$.

Sea un grafo $G = (X, U)$. Su *simplificación* es un nuevo grafo simple $G' = (X, E')$ con el mismo conjunto de vértices, y cuyas aristas vienen dadas por

$$[a, b] \in E' \Leftrightarrow (a, b) \in U \text{ y } a \neq b$$

La simplificación de un grafo corresponde a la idea intuitiva de "olvidarnos" del sentido de los arcos. En la figura tenemos un grafo (izquierda) y su simplificación (derecha).



Sea un grafo simple $G = (X, E)$.

El concepto de cadena cumple en los grafos simples una función análoga al de camino.

Una *cadena* es una sucesión finita de aristas de manera que un extremo de cada arista sea también extremo de la arista siguiente en la sucesión (con excepción de la última arista).

Las cadenas quedan unívocamente determinadas si se dan los vértices que atraviesan, en su orden. Por ejemplo, en el grafo de la figura (derecha), $[2, 1], [1, 3], [3, 4], [4, 5]$ es una cadena; se puede alternativamente hablar de la cadena $2, 1, 3, 4, 5$. Nótese que la cadena $[1, 2], [3, 1], [4, 3], [5, 4]$ es la misma que la anterior. Las cadenas $2, 1, 3, 4, 5$ y $5, 4, 3, 1, 2$ se identifican (aún siendo distintas en sentido estricto), y así no se habla del vertice inicial y el vértice inicial de una cadena, sino sólo de sus *extremos*: la cadena anterior está *entre* los vértices 2 y 5.

Una cadena es *simple* si en ella no se repite ningúna arista, y *elemental* si no se repite ningún vértice. Toda cadena elemental es simple, pero el recíproco no es cierto.

Análogamente al concepto de ciclo, en un grafo simple un *circuito* es una cadena simple en la que coinciden sus extremos. Por conveniencia, ahora se exige que no se repitan aristas (las cadenas de la forma $2, 1, 2$ no interesa que circuitos).

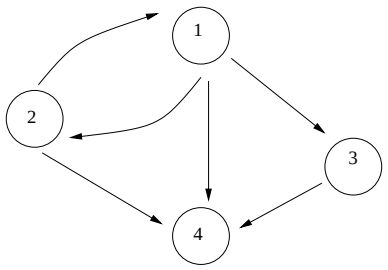
Sea un vértice $x \in X$. En los grafos simples el intragrado y el extragrado de x siempre coinciden; al valor común se le llama grado, y se le denota $\rho(x)$.

Conexión. Intuitivamente, un grafo es conexo cuando no está separado en diferentes "islas". Aunque la conexión se aplique a todo tipo de grafos, simples o no, es un concepto no dirigido, que exige "olvidarse" de los sentidos de los arcos. Matemáticamente, un grafo $G = (X, U)$ es *conexo* cuando, *en su simplificación*, dados dos vértices cualesquiera distintos existe siempre una cadena entre ellos.

Teorema. Sea $G = (X, U)$ un grafo. Entonces

$$\sum_{x \in X} \rho^+(x) = \sum_{x \in X} \rho^-(x) = |U|$$

No lo demostramos, pero lo podemos comprobar en el siguiente ejemplo.



La versión para grafos simples se puede demostrar como corolario del anterior.

Corolario. Sea $G = (X, E)$ un grafo simple. Entonces

$$\sum_{x \in X} \rho(x) = 2\,|E|$$

Ejercicio de modelización.

Teorema. Sea $G = (X, U)$ un grafo, y sean $x, y \in X$, $x \neq y$. Si existe un camino desde x hasta y, entonces existe un camino elemental desde x hasta y.

La demostración se basa en una sencilla idea fácilmente visualizable. La versión "natural" para grafos simples resulta ser cierta:

Teorema. Sea $G = (X, E)$ un grafo simple, y sean $x, y \in X$, $x \neq y$. Si existe una cadena entre x e y, entonces existe una cadena elemental entre x e y.

2.4. Grafos en el ordenador

Sea un grafo $G = (X, U)$, con $X = \{1, 2, ..., m\}$, $m \in \mathbb{N}^*$. Consideramos tres posibilidades para introducir el grafo en el ordenador.

1. La definición

Los arcos son pares de números (i, j), y éstos se introducen en el ordenador.

2. Matriz de adyacencia

La matriz de adyacencia $A = (a_{ij})_{\substack{i=1,...,m \\ j=1,...,m}}$ es una matriz cuadrada $m \times m$, cuyo término general es $a_{ij} = \begin{cases} 1, \text{ si } (i,j) \in U \\ 0, \text{ si no} \end{cases}$

En el ejemplo de la figura anterior sería:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

3. Matriz de incidencia

Para que se pueda escribir la matriz de incidencia, necesitamos dos hipótesis adicionales:

(1) Los arcos deben de estar etiquetados: $U = \{1, 2, ..., n\}$, $n \in \mathbb{N}^*$

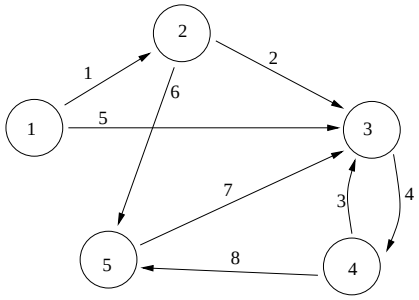
(2) El grafo ha de ser irreflexivo

La matriz de incidencia $A = (a_{ij})_{\substack{i=1,...,m \\ j=1,...,n}}$ es una matriz $m \times n$, cuyo término general es $a_{ij} = \begin{cases} 1, \text{ si el arco } j \text{ sale del vértice } i \\ -1, \text{ si el arco } j \text{ entra en el vértice } i \\ 0, \text{ si no ocurren los anteriores} \end{cases}$

(Regla mnemotécnica: "dar es positivo".)

Así pues, a cada fila le corresponde un vértice, y a cada columna un arco.

En el caso de la figura sería:



$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 \end{pmatrix}$$

Notese que siempre en todas las columnas hay un 1, un -1, y el resto ceros.

Ejercicio. ¿Puede tener la matriz de incidencia rango m?

Teorema. Sea un grafo conexo $G = (X, U)$ con matriz de incidencia A. Entonces $rg(A) = m - 1$.

2.5. Grafos acíclicos

Un grafo es *acíclico* cuando no tiene ningún ciclo.

Teorema. Sea $G = (X, U)$ un grafo. Denotamos $m := |X|$. Entonces las siguientes condiciones son equivalentes:

(a) G es acíclico.

(b) Existe $f : X \to \overline{m}$ biyectiva tal que:

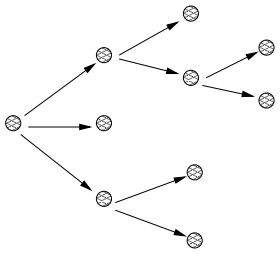
$$(i, j) \in U \Rightarrow f(i) < f(j), \quad \forall i, j \in X$$

Intuitivamente, el teorema dice que un grafo es acíclico si, y sólo si, podemos reetiquetar sus vértices de manera que todos los arcos vayan de un vértice de etiqueta menor a otro de etiqueta mayor.

Un tipo muy sencillo de grafos acíclicos son las arborescencias. Un grafo $G = (X, U)$ es una *arborescencia* cuando:

$$\exists x_r \in X : \forall x \in (X \sim \{x_r\}) \exists^* \text{ camino desde } x_r \text{ hasta } x$$

En una arborescencia, se pude probar fácilmente que el vértice x_r es el único con la propiedad de la definición. Se le llama la *raíz* de la arborescencia.

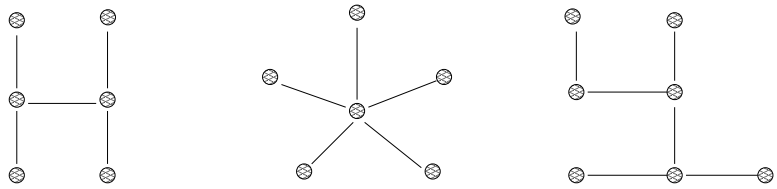


Las arborescencias corresponden a la idea intuitiva de los árboles de clasificación.

Teorema. Sea $G = (X, U)$ una arborescencia. Entonces G es acíclico y conexo.

2.6. Árboles

Los árboles son los análogos "no dirigidos" de las arborescencias. Compárese el teorema anterior con la siguiente definición. Un *árbol* es un grafo simple que es conexo y sin circuitos.



Teorema. Sea $G = (X, E)$ un grafo simple. Entonces las siguientes condiciones son equivalentes:

(a) G es un árbol.

(b) Para todo $x, y \in X$, tal que $x \neq y$, existe una, y una sola, cadena simple entre x e y.

(c) G es conexo y $|E| = |X| - 1$.

(d) G no tiene circuitos y además cumple: si se añade una nueva arista entre dos vértices distintos se forma un circuito.

Veamos el significado del anterior teorema de caracterización de árboles. La equivalencia entre (a) y (b) nos dice que efectivamente los árboles son los análogos no dirigidos de las arborescencias. La equivalencia de (a) con (c) y con (d) va a darnos el significado "económico" de los árboles.

Sea $G = (X, E)$ un grafo simple. Un *árbol generador* de G es un grafo parcial de G que es un árbol.

Teorema. Sea $G = (X, E)$ un grafo simple y conexo. Entonces G tiene un árbol generador.

Del teorema anterior obtenemos, aplicando la equivalencia de (a) con (c):

Corolario. Sea $G = (X, E)$ un grafo simple y conexo. Entonces $|E| \geq |X| - 1$.

Demostración. Existirá un árbol generador de G: $\exists T = (X, E')$ tal que es un árbol y $E' \subseteq E$. Al ser T un árbol, $|E'| = |X| - 1$. Luego, puesto que $E' \subseteq E$, se tiene que $|E| \geq |E'| = |X| - 1$.

Este corolario nos proporciona una interpretación "económica" de la equivalencia entre (a) y (c): el árbol es el grafo simple conexo con menor número de aristas. Un árbol ha de tener el suficiente número de aristas como para ser conexo, pero no demasiadas, para no tener circuitos. La equivalencia entre (a) y (d) insiste en esta idea.

Teorema. Sea $G = (X, E)$ un grafo simple y conexo. Entonces todo árbol generador de G resuelve el siguiente problema:

Problema (no ponderado) de interconexión óptima: Encontrar un grafo simple parcial de G que sea conexo y tenga el mínimo número posible de aristas.

Para refinar el problema de interconexión óptima, necesitamos introducir el concepto de grafo ponderado.

Un *grafo ponderado* G está formado por un grafo (X, U) y una función $p : U \to \mathbb{R}$ que asigna a cada arco su *peso*; se denota $G = (X, U, p)$. Un *grafo simple ponderado* G está formado por un grafo simple (X, E) y una función $p : E \to \mathbb{R}$ que asigna a cada arista su *peso*; se denota $G = (X, E, p)$.

Los pesos se suelen interpretar como costes o beneficios; nosotros los interpretaremos como costes. Obsérvese que no tienen por qué ser no negativos.

En un grafo simple ponderado, el peso de un grafo parcial simple es la suma de los pesos de sus aristas.

Teorema. Sea $G = (X, E, p)$ un grafo simple ponderado y conexo. Entonces G tiene un árbol generador de mínimo peso.

Teorema. Sea $G = (X, E, p)$ un grafo simple ponderado y conexo. Supóngase que $p \geq 0$. Entonces todo árbol generador de mínimo peso de G resuelve el siguiente problema:

Problema de interconexión óptima: Encontrar un grafo simple parcial de G que sea conexo y tenga mínimo peso.

Dado un grafo simple ponderado y conexo $G = (X, E, p)$, el *algoritmo de Kruskal* permite hallar un árbol generador de mínimo peso. En la primera iteración, se escoge una arista de mínimo peso. En la segunda iteración se añade de entre las aristas adyacentes con la anterior una de mínimo peso. En general, en la iteración k-ésima se parte de que se han escogido ya $k-1$ aristas que forman un subgrafo parcial simple conexo y sin circuitos, y se añade una nueva arista de manera que las k aristas resultantes formen un subgrafo parcial simple conexo y sin circuitos, siendo la arista añadida de mínimo peso de entre las aristas que cumplan esto. Tras la iteración $(|X|-1)$-ésima se obtiene un árbol generador de mínimo peso.

3. Ejercicios sobre matemática discreta

1. ¿Cuántas aristas tiene un grafo simple si sus vértices tienen grados $4, 3, 3, 2, 2$?. Justifíquese la respuesta y después dibújese el grafo.

2. ¿Puede existir un grafo simple cuyos vértices tienen grados $4, 3, 3, 2, 1$?. Justifíquese la respuesta.

3. ¿Es posible que en un grupo de 7 personas cada una conozca exactamente a 3 del grupo?.

4. Probar que en cualquier grupo de n personas al menos existen dos que tienen el mismo número de amigos (en el grupo).

5. Probar que en un grafo simple el número de vértices con grado impar es par.

6. En un mapa aparecen 20 poblaciones. La red de carreteras que las comunica cumple las siguientes condiciones: los cruces de los tramos se producen siempre en las poblaciones y desde cada población salen 3 tramos. Razónese cuántos tramos de carreteras existen.

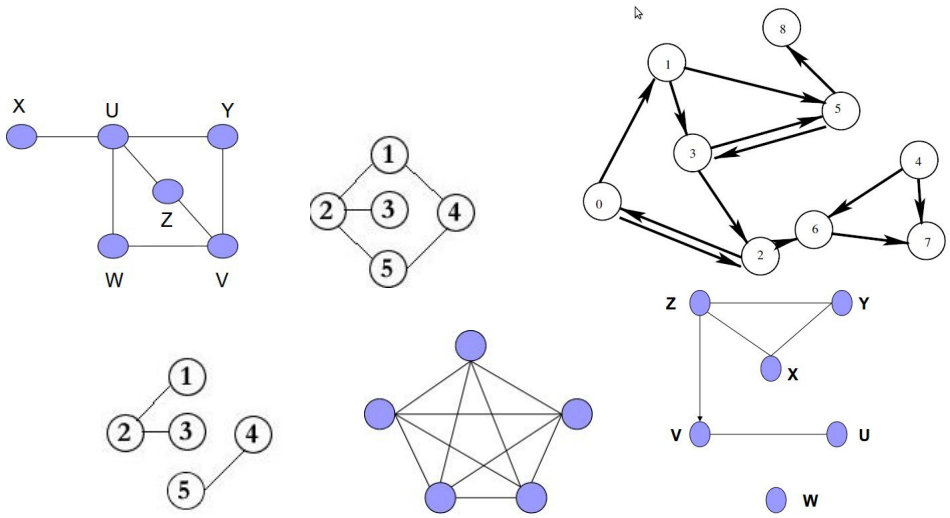
7. En un mapa de carreteras figuran 52 tramos de carretera, justifíquese cuál es el mayor número de poblaciones que pueden estar comunicadas (*desde una población cualquiera debe ser posible llegar a cualquier otra a través de la red*) asumiendo que solo hay cruces en las poblaciones.

8. Supongamos que se tienen cuatro aulas y las siguientes asignaturas con sus respectivos horarios para un mismo día:

   - Contabilidad I 8 a 12 hs.
   - Contabilidad II 10 a 14 hs.
   - Álgebra 14 a 18 hs.
   - Econometría 11 a 15 hs.
   - Macroeconomía 12 a 16 hs.
   - Microeconomía 9 a 13 hs.
   - Métodos 14 a 18 hs.
   - Estadística 14 a 18 hs.

Decídase si existe una forma de asignar aulas de modo que se puedan dictar todas las materias respetando los horarios. Modelícese como un problema de grafos.

9. Dados los siguientes grafos y grafos simples:

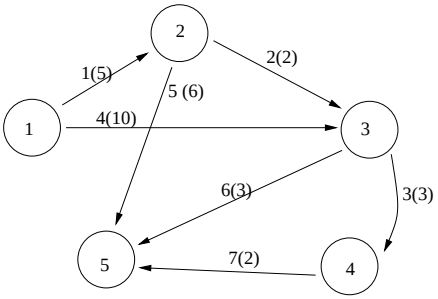


   a) Determínese qué grafos son conexos y cuáles son no conexos.

   b) Calcúlese la simplificación de los grafos.

   c) Calcúlese un árbol generador de los grafos simples conexos anteriores. En el caso de grafos calcúlese el árbol generador de su simplificación.

10. Considérese el siguiente grafo ponderado:



   a) Calcúlese la matriz de incidencia del grafo anterior.

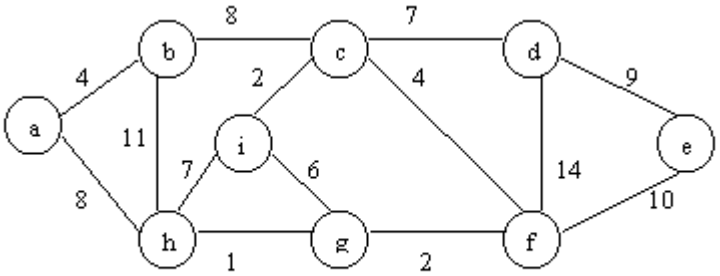
b) Calcúlese la matriz de adyacencia del grafo anterior.

c) A partir de la matriz de adyacencia determínese el intragrado y extragrado de todos los vértices del grafo.

d) Calcúlese la simplificación del grafo anterior.

e) Calcúlese un árbol generador de peso mínimo del grafo anterior usando el algoritmo de Kruskal.

11. Calcúlese un árbol generador de peso mínimo del siguiente grafo simple usando el algoritmo de Kruskal:



12. Un grafo simple se dice r-regular (o regular de grado r) si todos los vértices tienen grado r. Un grafo simple se dice completo si cada vértice es adyacente a todos los demas.

a) Demuéstrese que un grafo simple de n vértices r-regular tiene $n * r/2$ aristas.

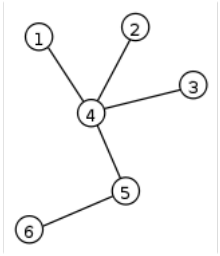
b) Demuéstrese que un grafo simple de n vértices completo tiene $n(n-1)/2$ aristas.

13. Pruébese que un grafo simple de n vértices que tiene más de $(n-1)(n-2)/2$ aristas es conexo.

14. Un grafo simple se dice bipartido si el conjunto de vértices se puede separar en dos conjuntos disjuntos V_1 y V_2 y las aristas siempre unen vértices de un conjunto con vértices de otro.

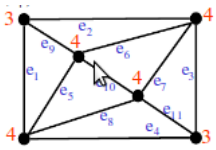
a) Propóngase un ejemplo de grafo simple bipartido.

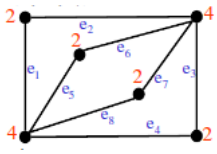
b) Decídase si el siguiente grafo simple es bipartido:

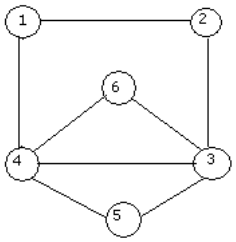


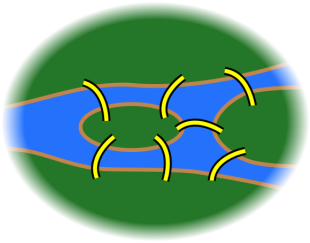
c) Calcúlese la matriz de adyacencia del grafo anterior. ¿Qué se puede decir de la matriz de adyacencia de un grafo simple bipartido?.

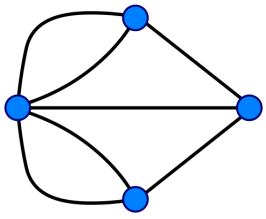
15. En la teoría de grafos, una cadena euleriana es un cadena que pasa por cada arista una y solo una vez. Un circuito euleriano es un circuito que recorre cada arista exactamente una vez. Determínese si en los siguientes grafos es posible construir cadenas o circuitos eulerianos.



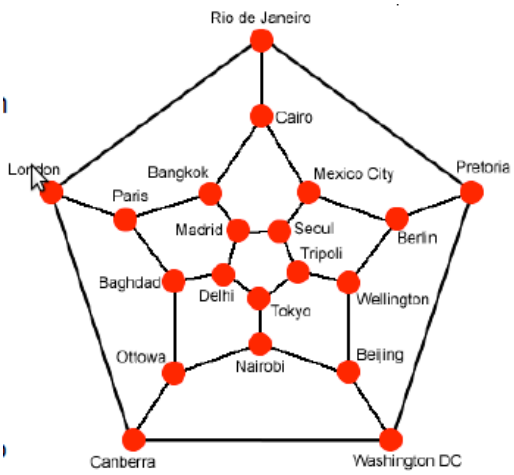








16. En la teoría de grafos, un cadena hamiltoniana es un cadena elemental que pasa por todos los vértices del grafo. Si además el último vértice visitado es adyacente al primero, la cadena se denomina circuito hamiltoniano. Constrúyase un circuito hamiltoniano del siguiente grafo simple.


Rio de Janeiro
Cairo
London
Bangkok
Mexico City
Pretoria
Paris
Madrid
Seoul
Berlin
Tripoli
Baghdad
Delhi
Wellington
Tokyo
Ottowa
Nairobi
Beijing
Canberra
Washington DC

Bibliografía

El lector no se sentirá decepcionado si consulta las siguientes referencias bibliográficas. Para la primera parte se recomiendan: [1], [4], [6], [8], [12]; para la segunda: [2], [3], [4], [5], [7], [9], [10] , [11], [12], [13]; y para el apéndice: [1], [4], [6], [7].

[1] Biggs, N.L. (2002): Discrete Mathematics. Segunda edición. Oxford U.P. (Existe traducción al castellano de la primera edición.)

[2] Bazaraa, M.S.; Jarvis, J.J.; Sherali, H.D. (1990): Linear Programming and Network Flows. Wiley. (Existe traducción al castellano.)

[3] Brinkhuis, J.; Tikhomirov, V. (2005): Optimization: Insights and Applications. Princeton U.P.

[4] Carré, B. (1979): Graphs and Networks. Oxford U.P.

[5] Goberna, M.A.; Jornet, V.; Puente, R.O. (2004): Optimización lineal. Teoría, métodos y modelos. McGraw-Hill.

[6] Jungnickel, D. (1994): Graphen, Netzwerke und Algorithmen. Wissenschaftsverlag Mannheim. (Existe traducción inglesa actualizada (2005): Graphs, Networks and Algorithms. Springer.)

[7] Lawler, E.L. (1976): Combinatorial Optimization. Networks and Matroids. Dover (reedición, 2001).

[8] Lewis, H.; Papadimitriou, C. (1997): Elements of the Theory of Computation. Pearson.

[9] Nemhauser, G.L.; Wolsey, L.A. (1999): Integer and Combinatorial Optimization. Wiley.

[10] Padberg, M. (1999): Linear optimization and extensions. Springer.

[11] Salazar González, J.J. (2001): Programación Matemática. Díaz de Santos.

[12] Schrijver, A. (1989): Theory of linear and integer programming. Wiley.

[13] Solow, D. (1984): Linear Programming. North-Holland.