



UNIVERSIDADE DA CORUÑA

Algoritmos: Algoritmos voraces

Alberto Valderruten

LFCIA - Departamento de Computación

Facultad de Informática

Universidad de A Coruña, España

www.lfcia.org/alg

www.fi.udc.es

Contenido

- Características
Devolver el cambio
- Problema de la mochila
- Ordenación topológica
- Arbol expandido mínimo
Kruskal, Prim
- Caminos mínimos
Dijkstra

Devolver el cambio

- *Sistema monetario M*: monedas de valor 1, 2, 5, 10, 20, 50, 100, 200
- **Problema**: pagar exactamente n unidades de valor con un mínimo de monedas.
→ Función Devolver cambio:

```
función Devolver cambio (n) : conjunto de monedas
  const M = {1, 2, 5, 10, 20, 50, 100, 200}; {denominaciones de las monedas}
  S := conjunto vacío; {la solución se construye en S}
  ss := 0; {suma de las monedas de S}
  {bucle voraz:}
  mientras ss <> n hacer
    x := mayor elemento de M : ss + x <= n;
    si no existe tal elemento entonces devolver "no hay soluciones";
    S := S U {una moneda de valor x};
    ss := ss + x;
  devolver S
fin función
```

- Intuitivamente, sabemos que se obtiene *siempre* una solución óptima

Devolver el cambio: caracterización

- *¿Por qué funciona?*
⇒ M adecuado y número suficiente de monedas.
 - **No funciona con cualquier M :**
Ejemplo: $M = \{1, 4, 6\}$, $n = 8 \rightarrow \{6, 1, 1\}$ en vez de $\{4, 4\}$ (óptimo)
Este problema se resolverá con Programación Dinámica
 - La función Devolver cambio es **voraz** (algoritmos ávidos, *greedy*)
¿Por qué voraz?
 - Selecciona el mejor *candidato* que puede en cada iteración, sin valorar consecuencias.
 - Una vez seleccionado un candidato, decide definitivamente:
 - aceptarlo, o
 - rechazarlosin evaluación en profundidad de alternativas, sin retroceso...
- Algoritmos sencillos: tanto en diseño como en implementación.
Cuando la técnica funciona, se obtienen algoritmos eficientes.



Características de los algoritmos voraces

- Resuelven problemas de *optimización*:
En cada fase, toman una decisión (selección), satisfaciendo un óptimo local según la información disponible, esperando así, en conjunto, satisfacer un óptimo global.
- Manejan un conjunto de *candidatos* C :
En cada fase, retiran el candidato seleccionado de C , y si es aceptado se incluye en S :
Conjunto donde se construye la solución \equiv candidatos aceptados.
- Utilizan 4 funciones (explícitamente o no):
 1. $\text{¿}S \text{ es Solución?}$
 2. $\text{¿}S \text{ es Factible?} \equiv \text{¿Puede completarse para obtener una solución?}$
 3. *Selección*: determina el mejor candidato
 4. *Objetivo*: valora S (relacionada con *Selección*)

→ Encontrar S : *Solución* y optimiza *Objetivo* (max/min)



Esquema de los algoritmos voraces

■ Función Voraz (esquema):

```
función Voraz (C:conjunto): conjunto
  S := conjunto vacío;           {la solución se construye en S}
  {bucle voraz:}
  mientras C <> conjunto vacío y no solución(S) hacer
    x := seleccionar(C);
    C := C \ {x};
    si factible (S ∪ {x}) entonces S := S ∪ {x}
  si solución (S) entonces devolver S
  sino devolver "no hay soluciones"
fin función
```

■ Ejemplo: Devolver cambio

→ tenemos que:

- adaptar el esquema al problema
- introducir mejoras (ejemplo: añadir div)

- ## ■ Problema: Asegurarse/demostrar que la técnica funciona.
- No siempre funciona: “tomar calle principal”



El problema de la mochila I (1)

- n objetos: $i = 1..n$ $\begin{cases} \text{peso } w_i > 0 \\ \text{valor } v_i > 0 \end{cases}$

Problema: cargar una mochila de capacidad W , maximizando el valor de su carga.

- Versión I: los objetos se pueden fraccionar \equiv fracción $x_i, 0 \leq x_i \leq 1$
 \Rightarrow el objeto i contribuye
en $x_i w_i$ al peso de la carga, limitado por W ,
en $x_i v_i$ al valor de la carga, que se quiere maximizar.

$$\Rightarrow \boxed{\max \sum_{i=1}^n x_i v_i \text{ con la restricción } \sum_{i=1}^n x_i w_i \leq W}$$

- + Hipótesis: $\sum_{i=1}^n w_i > W$, sino la solución es trivial.
 \Rightarrow en óptimo, $\sum_{i=1}^n x_i w_i = W$

El problema de la mochila I (2)

```
función Mochila 1 ( w[1..n], v[1..n], W): objetos[1..n]
  para i := 1 hasta n hacer
    x[i] := 0;           {la solución se construye en x}
  peso := 0;
  {bucle voraz:}
  mientras peso < W hacer
    i := el mejor objeto restante; {1}
    si peso+w[i] <= W entonces
      x[i] := 1;
      peso := peso+w[i]
    sino
      x[i] := (W-peso)/w[i];
      peso := W;
  devolver x
fin función
```


El problema de la mochila I (3)

- ¿Mejor objeto restante? → Ejemplar de problema: $n = 5, W = 100$

	1	2	3	4	5	
v_i	20	30	66	40	60	
w_i	10	20	30	40	50	$(\sum_{i=1}^n w_i > W)$

Selección:

1. ¿Objeto más valioso? $\leftrightarrow v_i$ max
2. ¿Objeto más ligero? $\leftrightarrow w_i$ min
3. ¿Objeto más rentable? $\leftrightarrow v_i/w_i$ max

	1	2	3	4	5	
v_i	20	30	66	40	60	
w_i	10	20	30	40	50	
v_i/w_i	2,0	1,5	2,2	1,0	1,2	Objetivo $(\sum_{i=1}^n x_i v_i)$
$x_i (v_i \text{ max})$	0	0	1	0,5	1	146
$x_i (w_i \text{ min})$	1	1	1	1	0	156
$x_i (v_i/w_i \text{ max})$	1	1	1	0	0,8	164

El problema de la mochila I (4)

- **Teorema:** Si los objetos se seleccionan por orden decreciente de v_i/w_i , el algoritmo Mochila 1 encuentra la solución óptima.
Demostración por absurdo.
- **Análisis:**
inicialización: $O(n)$;
bucle voraz: $O(1) * n$ (peor caso) $\rightarrow O(n)$
+ ordenación: $O(n \log n)$
- Mejora: con un montículo
inicialización: $+O(n)$ (Crear montículo);
bucle voraz: $O(\log n) * n$ (peor caso) $\rightarrow O(n \log n)$
 \rightarrow pero mejores $T(n)$

Ordenación topológica (1)

- **Definición:** Ordenación de los nodos de un grafo dirigido acíclico:
 \exists camino $v_i, \dots, v_j \Rightarrow v_j$ aparece después de v_i .
- **Aplicación:** sistema de prerequisites (llaves) en una titulación.
 $(u, v) \equiv u$ debe aprobarse antes de acceder a v
→ grafo acíclico, sino la ordenación no tiene sentido.
- **Observación:** la ordenación no es única.
- **Definición:** Grado de entrada de v : número de aristas (u, v) .
- **Algoritmo:** buscar nodo de grado 0, enviarlo a la salida y eliminarlo junto a las aristas que partan de él.
Hipótesis: el grafo ya está en memoria, *listas de adyacencia*.

Ordenación topológica (2)

```
procedimiento Ordenación topológica 1 (G:grafo)
{Precondición: Grado Entrada [1..|N|] previamente calculado}
  contador := 1;
  error := falso;
  mientras contador <= |N| y no error hacer
    v := Buscar nodo de grado 0 (...); {*}
    si v=0 entonces
      error:=cierto;
      error "el grafo tiene un ciclo"
    sino
      Número Topológico [v] := contador ;
      para cada w adyacente a v hacer
        Grado Entrada [w] := Grado Entrada [w]-1;
      incrementar contador
fin procedimiento
```

{*} recorre Grado Entrada buscando $0 \wedge$ sin Número Topológico asignado.

$|N| = n \Rightarrow O(n)$ para cada llamada (acceso secuencial)

n llamadas \Rightarrow $O(n^2)$ para el algoritmo.

Ordenación topológica (3)

- **Mejora:** estructura para nodos cuyo grado de entrada sea 0.
 - $\{*\}$ puede devolver cualquiera de ellos;
 - al decrementar un grado, decidir si se incluye el nodo en la estructura.
 → pila o cola.

- **Ejemplo:** evolución de Grado Entrada

nodo							
1	0						
2	1	0					
3	2	1	1	1	0		
4	3	2	1	0			
5	1	1	0				
6	3	3	3	3	2	1	0
7	2	2	2	1	0		
Ins	1	2	5	4	3,7	-	6
Elim	1	2	5	4	3	7	6

Ordenación topológica (4)

```
procedimiento Ordenación topológica 2 (G:grafo)
  Crear Cola (C);
  contador := 1 ;
  para cada nodo v hacer
    si Grado Entrada [v] = 0 entonces Insertar Cola (v, C);
  mientras no Cola Vacía (C) hacer
    v := Eliminar Cola (C);
    Número Topológico [v] := contador;
    incrementar contador;
    para cada w adyacente a v hacer
      Grado Entrada [w] := Grado Entrada [w]-1;
      si Grado Entrada [w] = 0 entonces Insertar Cola (w, C);
  si contador <= |N| entonces error "el grafo tiene un ciclo"
fin procedimiento
```

- $\rightarrow O(|A| + |N|)$ con listas de adyacencia.
Peor caso: grafo denso y visita todas las aristas.

Arbol expandido mínimo (1)

- **a. e. m.**, árbol de expansión, árbol de recubrimiento
- Sea $G = (N, A)$ conexo, no dirigido, pesado (costes ≥ 0 en las aristas):
Problema: T subconjunto de A tal que $G' = (N, T)$ conexo,
 \sum costes mínimo y $|T|$ mínimo.
- $|N| = n \Rightarrow |T| \geq n - 1$;
pero, si $|T| > n - 1 \Rightarrow \exists$ ciclo
→ podemos quitar una arista del ciclo
 $\Rightarrow |T| = n - 1 \wedge G'$ conexo \Rightarrow **árbol** (e. m.)
- **Aplicación:** instalación de cableado → ¿solución más económica?
- Técnica voraz:
 - *Candidatos:* aristas → S : conjunto de aristas
 - *Solución?:* $S = T$?
 - *Factible?:* (N, S) sin ciclos (ej: S vacío)

Arbol expandido mínimo (2)

- **Definición:** una arista *parte* de un conjunto de nodos
 \Leftrightarrow *uno* de sus extremos está en el conjunto.
(no parte \Leftrightarrow sus 2 extremos están dentro/fuera del conjunto)
- **Lema:** sean $G = (N, A)$ un grafo conexo, no dirigido, pesado;
 B un subconjunto (estricto) de N ;
 T un subconjunto (estricto) de A , *Factible*,
sin aristas que partan de B ;
 (u, v) : la arista *más corta* que parte de B
 $\Rightarrow TU\{(u, v)\}$ es *Factible*
- \rightarrow Algoritmos de Kruskal y Prim

Algoritmo de Kruskal (1)

- Inicialmente: T vacío
- Invariante:
(N, T) define un conjunto de componentes conexas (subgrafos): árboles
- Final: sólo una componente conexa: el a. e. m.
- *Selección*: lema \rightarrow **arista más corta...**
- *Factible?*: **...que una componentes conexas distintas**
- Estructuras de datos:
 - “grafo”: aristas ordenadas
 - componentes conexas: Conjuntos Disjuntos ($\text{buscar}(x)$, $\text{fusionar}(A, B)$)

Algoritmo de Kruskal (2)

■ Ejemplo:

arista	(1,2)	(2,3)	(4,5)	(6,7)	(1,4)	(2,5)	(4,7)	(3,5)	...
peso	1	2	3	3	4	4	4	5	

paso	selección	componentes conexas							
ini	-	<table border="1"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td> </tr> </table>	1	2	3	4	5	6	7
1	2	3	4	5	6	7			
1	(1,2)	<table border="1"> <tr> <td>1,2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td> </tr> </table>	1,2	3	4	5	6	7	
1,2	3	4	5	6	7				
2	(2,3)	<table border="1"> <tr> <td>1,2,3</td><td>4</td><td>5</td><td>6</td><td>7</td> </tr> </table>	1,2,3	4	5	6	7		
1,2,3	4	5	6	7					
3	(4,5)	<table border="1"> <tr> <td>1,2,3</td><td>4,5</td><td>6</td><td>7</td> </tr> </table>	1,2,3	4,5	6	7			
1,2,3	4,5	6	7						
4	(6,7)	<table border="1"> <tr> <td>1,2,3</td><td>4,5</td><td>6,7</td> </tr> </table>	1,2,3	4,5	6,7				
1,2,3	4,5	6,7							
5	(1,4)	<table border="1"> <tr> <td>1,2,3,4,5</td><td>6,7</td> </tr> </table>	1,2,3,4,5	6,7					
1,2,3,4,5	6,7								
6	(2,5) rechazada	<table border="1"> <tr> <td>1,2,3,4,5</td><td>6,7</td> </tr> </table>	1,2,3,4,5	6,7					
1,2,3,4,5	6,7								
7	(4,7)	<table border="1"> <tr> <td>1,2,3,4,5,6,7</td> </tr> </table>	1,2,3,4,5,6,7						
1,2,3,4,5,6,7									

Algoritmo de Kruskal (3)

```
función Kruskal ( G =(N,A) ) : árbol
  Ordenar A según longitudes crecientes;
  n := |N|;
  T := conjunto vacío;
  inicializar n conjuntos, cada uno con un nodo de N;
  {bucle voraz:}
  repetir
    a := (u,v) : arista más corta de A aún sin considerar;
    Conjunto U := Buscar (u);
    Conjunto V := Buscar (v);
    si Conjunto U <> Conjunto V entonces
      Fusionar (Conjunto U, Conjunto V);
      T := TU{a}
  hasta |T| = n-1;
  devolver T
fin función
```

Algoritmo de Kruskal (4)

- **Teorema:** Kruskal calcula el árbol expandido mínimo.
Demostración por inducción sobre $|T|$ y utilizando el lema anterior.
- **Análisis:** $|N| = n \wedge |A| = m$
ordenar A : $\Theta(m \log m) \equiv \Theta(m \log n) : n - 1 \leq m \leq n(n - 1)/2$
+ inicializar n conjuntos disjuntos: $\Theta(n)$
+ $2m$ buscar (peor caso)
y $n - 1$ fusionar (siempre): $\Theta(2m\alpha(2m, n)) = \Theta(m \log n)$
+ resto: $\Theta(m)$ (peor caso)
 $\Rightarrow \boxed{T(n) = \Theta(m \log n)}$
- **Mejora:** utilizar un montículo de aristas en vez de ordenarlas
No cambia la complejidad del peor caso pero se obtienen mejores tiempos (ejercicio).

Algoritmo de Prim (1)

- Kruskal es un bosque que crece, en cambio Prim es un único árbol que va creciendo hasta alcanzar todos los nodos.
- Inicialización: $B = \{\text{nodo arbitrario}\} = \{1\}$, T vacío
- *Selección*: arista más corta que parte de B :
 $(u, v), u \in B \wedge v \in N - B$
 \Rightarrow se añade (u, v) a T y v a B
- Invariante: T define en todo momento un a.e.m. del subgrafo (B, A)
- Final: $B = N$ (*Solución?*)

Algoritmo de Prim (2)

```
función Prim 1 ( G =(N,A) ) : árbol
  T := conjunto vacío;
  B := {un nodo de N};
  mientras B <> N hacer
    a := (u,v): arista más corta que parte de B
              (u pertenece a B y v no);
    T := T U {a};
    B := B U {v}
  devolver T
fin función
```



Algoritmo de Prim (4)

- **Ejemplo** (el mismo que para Kruskal):

paso	selección	B
ini	-	1
1	(1,2)	1,2
2	(2,3)	1,2,3
3	(1,4)	1,2,3,4
4	(4,5)	1,2,3,4,5
5	(4,7)	1,2,3,4,5,7
6	(7,6)	1,2,3,4,5,6,7 = N

- **Observación:** No se producen rechazos.
- **Teorema:** Prim calcula el árbol expandido mínimo.
Demostración por inducción sobre $|T|$ y utilizando el lema anterior.

Algoritmo de Prim (5)

■ Implementación:

L : matriz de adyacencia $\equiv L[i, j] = \begin{cases} \text{distancia si } \exists(i, j) \\ \infty \text{ sino} \end{cases}$
→ matriz simétrica (\equiv desperdicio de memoria)

Para cada nodo $i \in N - B$:

Más Próximo $[i]$: nodo $\in B$ más próximo

Distancia Mínima $[i]$: distancia de $(i, \text{Más Próximo } [i])$

Para cada nodo $i \in B$:

Distancia Mínima $[i] = -1$

Algoritmo de Prim (6)

```
función Prim 2 ( L[1..n,1..n] ) : árbol
  Distancia Mínima [1] := -1;
  T := conjunto vacío;
  para i := 2 hasta n hacer
    Más Próximo [i] := 1;
    Distancia Mínima [i] := L[i,1]
  {bucle voraz:}
  repetir n-1 veces:
    min := infinito;
    para j := 2 hasta n hacer
      si 0 <= Distancia Mínima [j] < min entonces
        min := Distancia Mínima [j];
        k := j
    T := T U { ( Más Próximo [k], k ) };
    Distancia Mínima [k] := -1;           {añadir k a B}
    para j := 2 hasta n hacer
      si L[j,k] < Distancia Mínima [j] entonces
        Distancia Mínima [j] := L[j,k];
        Más Próximo [j] := k

  devolver T
fin función
```

Algoritmo de Prim (7)

- **Análisis:**

inicialización = $\Theta(n)$

bucle voraz: $n - 1$ iteraciones, cada para anidado = $\Theta(n)$

$$\Rightarrow T(n) = \Theta(n^2)$$

- ¿Posible mejora con un montículo?

→ $\Theta(m \log n)$, igual que Kruskal (ejercicio).

- **Comparación:**

	Prim $\Theta(n^2)$	Kruskal $\Theta(m \log n)$
Grafo denso: $m \rightarrow n(n - 1)/2$	$\Theta(n^2)$	$\Theta(n^2 \log n)$
Grafo disperso: $m \rightarrow n$	$\Theta(n^2)$	$\Theta(n \log n)$

Tabla: Complejidad temporal de los algoritmos de Prim y Kruskal.

Arbol expandido mínimo (3)

■ Observaciones:

- Existen algoritmos más eficientes, más sofisticados, más recientes (Kruskal es de 1956, Prim de 1957 [1930]).
- Son ejemplos importantes de aplicación de la técnica voraz. Contrastar con la función voraz (ejercicio).

Caminos mínimos (1)

■ Problema:

Dado un grafo $G = (N, A)$ dirigido,
con longitudes en las aristas ≥ 0 ,
con un nodo distinguido como *origen* de los caminos (el nodo 1),
→ encontrar los caminos mínimos entre el nodo origen
y los demás nodos de N
⇒ **algoritmo de Dijkstra**

■ Técnica voraz:

- 2 conjuntos de nodos: $\begin{cases} S \equiv \text{seleccionados:} & \text{camino mínimo establecido} \\ C \equiv \text{candidatos:} & \text{los demás} \end{cases}$
- invariante: $N = S \cup C$
- inicialmente, $S = 1$ → final: $S = N$: *función solución*
- *función selección*: nodo de C con menor distancia conocida desde 1
→ existe una información *provisional* sobre distancias mínimas

Caminos mínimos (2)

■ Definición:

Un camino desde el origen a un nodo v es *especial* ssi todos sus *nodos intermedios* están en S .

⇒ D : vector con longitudes de caminos especiales mínimos;

Selección de $v \leftrightarrow$ **el camino especial mínimo** $1..v$
es también camino mínimo (se demuestra!).

Al final, D contiene las longitudes de los caminos mínimos.

■ Implementación:

- $N = 1, 2, \dots, n$ (1 es el origen)

- $$\begin{cases} L[i, j] & \geq 0 \text{ si } (i, j) \in A \\ & = \infty \text{ sino} \end{cases}$$

Matriz de adyacencia, no simétrica.

Algoritmo de Dijkstra

```
función Dijkstra ( L[1..n,1..n] ) : vector
  C := { 2, 3, ..., n};
  para i := 2 hasta n hacer
    D[i] := L[1,i];           {1}
  {bucle voraz:}
  repetir n-2 veces:
    v := nodo de C que minimiza D[v];
    C := C\{v};
    para cada w en C hacer
      D[w] := min ( D[w], D[v]+L[v,w] )   {2}
  devolver D
fin función
```

```
{1} D[i] := L[1,i]; P[i] := 1
```

```
{2} si D[w] > D[v]+L[v,w] entonces
    D[w] := D[v]+L[v,w];
    P[w] := v
```

Algoritmo de Dijkstra (2)

- **Ejemplo:**

paso	selección	C	D[2]	D[3]	D[4]	D[5]
ini	-	2, 3, 4, 5	50	30	100	10
1	5	2, 3, 4	50	30	20	10
2	4	2, 3	40	30	20	10
3	3	2	35	30	20	10

Tabla: Evolución del conjunto C y de los caminos mínimos.

- **Observación:** 3 iteraciones = $n - 2$
- ¿Calcular también los nodos intermedios?
 - vector $P[2..n]$:
 $P[v] \equiv$ nodo que *precede* a v en el camino mínimo
 - Seguir precedentes hasta el origen.

Algoritmo de Dijkstra (3)

- **Teorema:** Dijkstra encuentra los caminos mínimos desde el origen hacia los demás nodos del grafo.

Demostración por inducción.

- **Análisis:** $|N| = n, |A| = m, L[1..n, 1..n]$

Inicialización = $O(n)$

¿Selección de v ?

→ “implementación rápida”: recorrido sobre C

≡ examinar $n - 1, n - 2, \dots, 2$ valores en D

$$\sum = \Theta(n^2)$$

Para anidado: $n - 2, n - 3, \dots, 1$ iteraciones

$$\sum = \Theta(n^2)$$

$$T(n) = \Theta(n^2)$$

Mejora: si el grafo es disperso ($m \ll n^2$), utilizar listas de adyacencia

→ ahorro en para anidado: recorrer lista y no fila o columna de L

Algoritmo de Dijkstra (4)

- **Análisis (Cont.):**

¿Cómo evitar $\Omega(n^2)$ en selección?

→ C : montículo min, ordenado según $D[i]$

⇒ inicialización en $\Theta(n)$

$C := C - v$ en $O(\log n)$

Para anidado: modificar $D[w] = O(\log n) \equiv$ flotar

¿Nº de veces? Máximo 1 vez por arista (peor caso)

En total:

- extraer la raíz $n - 2$ veces (siempre)

- modificar un máximo de m veces un valor de D (peor caso)

⇒ $T(n) = \Theta((m + n)\log n)$

- **Observación:** La “implementación rápida” es preferible si el grafo es denso.



UNIVERSIDADE DA CORUÑA

Algoritmos: Algoritmos voraces

Alberto Valderruten

LFCIA - Departamento de Computación

Facultad de Informática

Universidad de A Coruña, España

www.lfcia.org/alg

www.fi.udc.es