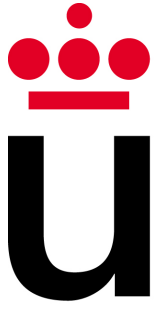
Algoritmos Voraces

Diseño y Análisis de Algoritmos



Universidad
Rey Juan Carlos

Contenidos

Introducción

Algoritmos voraces

- Los algoritmos voraces se suelen aplicar a problemas de optimización
  - Maximizar o minimizar una función objetivo
- Suelen ser rápidos y fáciles de implementar
- Exploran soluciones "locales"
- No siempre garantizan la solución óptima

Algoritmos voraces

- Son algoritmos que siguen una heurística mediante la cual toman **decisiones óptimas locales** en cada paso, de manera muy eficiente, con la esperanza de poder encontrar un óptimo global tras una serie de pasos
- Se aplican, sobre todo, a problemas duros, desde un punto de vista computacional
  - Ejemplo: problema del viajante (NP-completo)
  - Heurística: "escoge la ciudad más próxima no visitada aún"
- Para ciertos problemas se puede demostrar que algunas estrategias voraces sí que logran hallar un óptimo global de manera eficiente

Ventajas y desventajas

- Ventajas
  - Son fáciles de implementar
  - Producen soluciones eficientes
  - A veces encuentran la solución óptima
- Desventajas
  - No todos los problemas de optimización son resolubles con algoritmos voraces
  - La búsqueda de un óptimo local no implica encontrar un óptimo global
  - Dificultad de encontrar la función de selección que garantice la elección óptima

Elementos

- **Conjunto de candidatos** C: la solución se construirá con un subconjunto de estos candidatos
- **Función de selección**: selecciona el candidato "local" más idóneo
- **Función de factibilidad**: comprueba si un candidato es factible
- **Función objetivo**: determina el valor de la solución (función a optimizar)
- **Función solución**: determina si el subconjunto de candidatos ha alcanzado una solución

Esquema general

- La técnica voraz funciona por pasos:
  - Partimos de una solución vacía y de un conjunto de candidatos a formar parte de la solución
  - En cada paso se intenta añadir el mejor de los candidatos restantes a la solución parcial
  - Una vez tomada la decisión, no se puede deshacer
  - Si la solución ampliada es válida $\Rightarrow$ candidato incorporado
  - Si la solución ampliada no es válida $\Rightarrow$ candidato desechado
- El algoritmo acabará cuando el conjunto de elementos seleccionados constituya una solución o cuando no queden elementos sin considerar

Esquema general en pseudocódigo

función voraz(C) //C es el conjunto de candidatos//

$S \leftarrow \emptyset$
mientras $C \neq \emptyset$ y no solución(S) hacer
    $x \leftarrow$ seleccionar(C)
    $C \leftarrow C \backslash \{x\}$
    si factible($S \cup \{x\}$) entonces
        $S \leftarrow S \cup \{x\}$

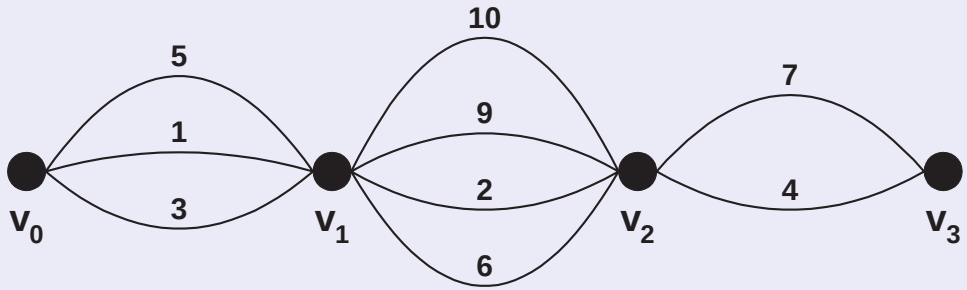
si solución(S) entonces
    devolver S //S es una solución//
si no
    devolver $\emptyset$ //No hay soluciones//

Ejemplos básicos

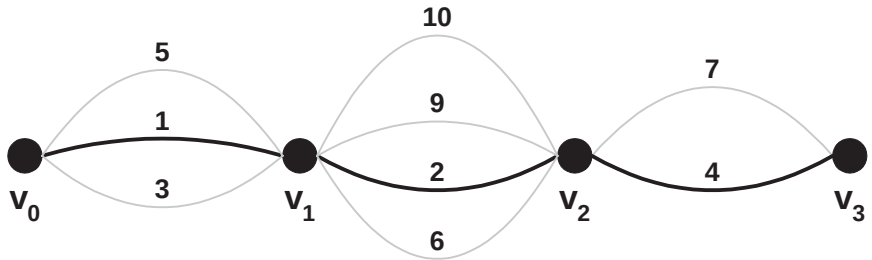
Camino más corto - 1

Camino más corto - 1

Encontrar el camino más corto de v_0 a v_n, donde solo hay caminos entre vértices adyacentes (v_{i-1} y v_i, para $i = 1, \ldots, n$).



- ¿Función de selección?
  - Método voraz: en cada paso se coge el arco de menor longitud

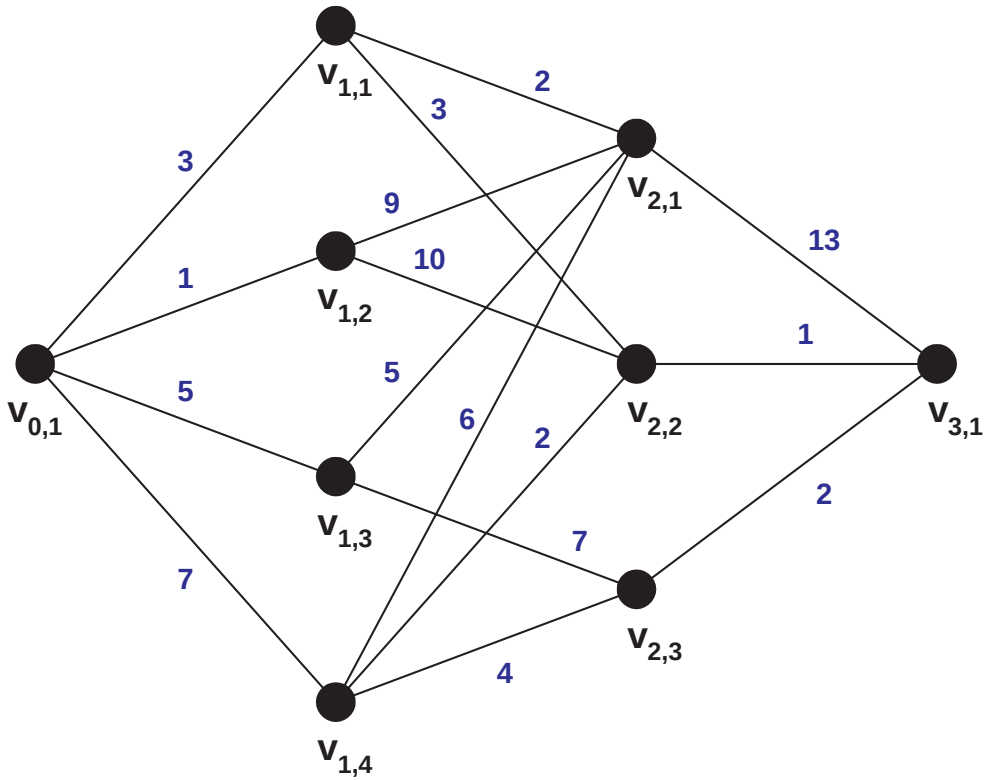


- Solución optima: $1 + 2 + 4 = 7$
- Se demuestra que la estrategia es óptima (por contradicción)

Camino más corto - 2

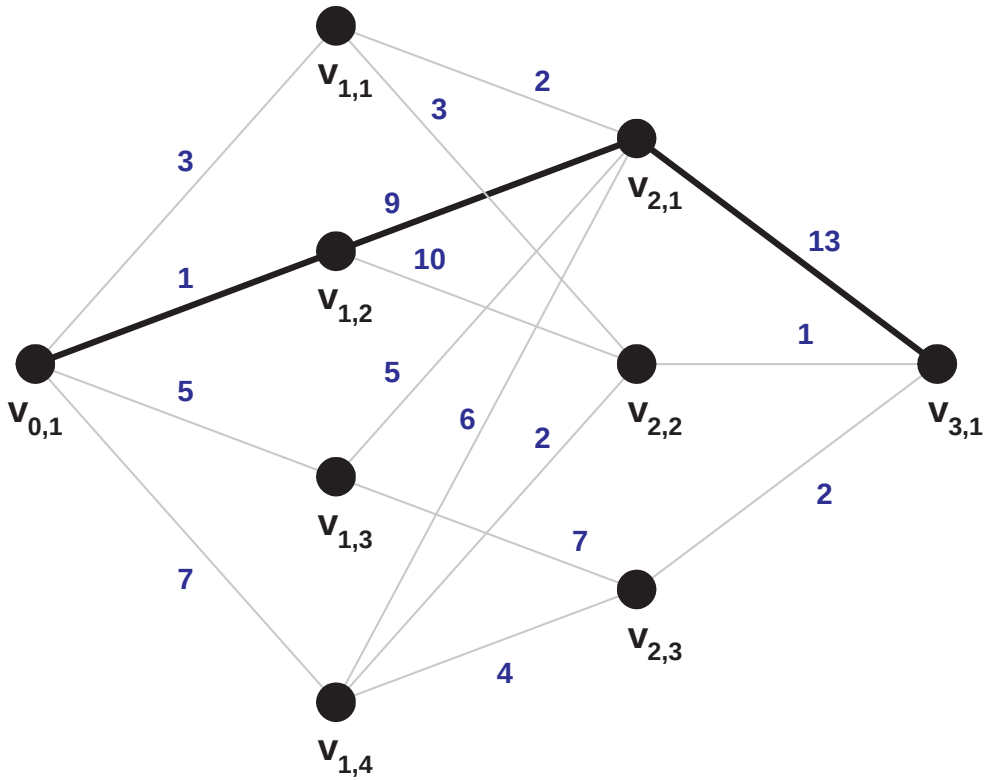
Camino más corto - 2

Encontrar el camino más corto de $v_{0,1}$ a $v_{n,1}$, donde solo hay caminos entre vértices de etapas adyacentes ($v_{i-1,j}$ y $v_{i,k}$, para $i = 1, \ldots, n$, y $\forall j, k$, donde puede haber cualquier número de vértices en las etapas 1, 2,. . . ,$n-1$).



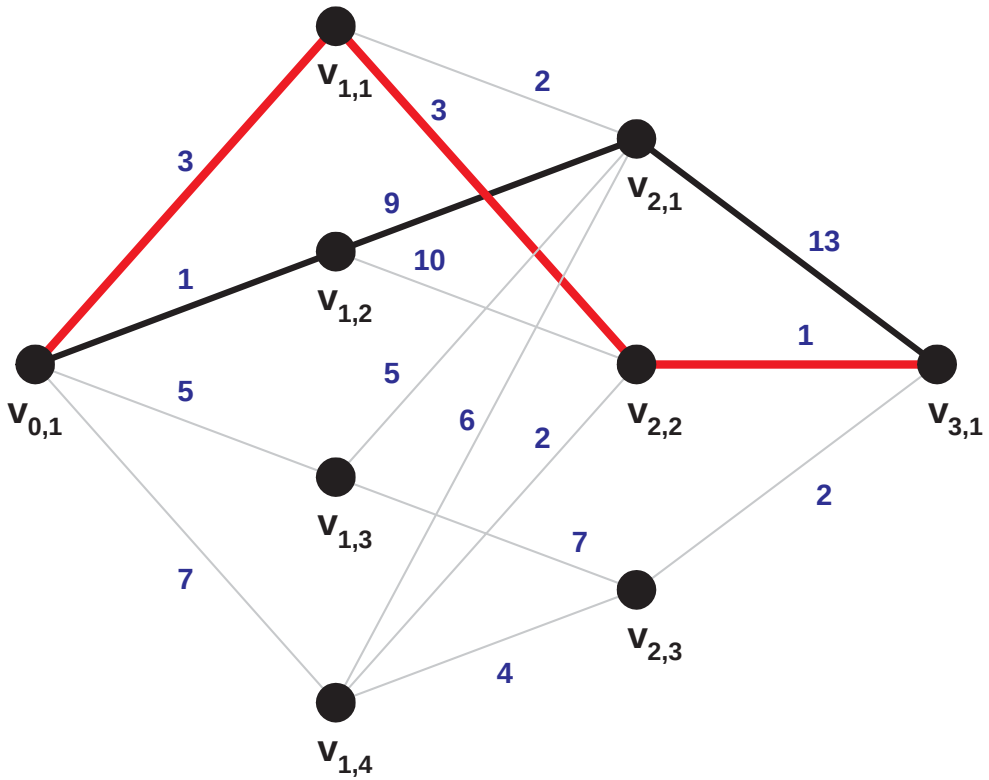
Camino más corto - 2

- ¿Función de selección?
  - Método voraz: en cada paso se coge el arco de menor longitud



- Longitud de la solución voraz: $1 + 9 + 13 = 23$ $(v_{0,1}, v_{1,2}, v_{2,1}, v_{3,1})$

Camino más corto - 2



- Se demuestra que la estrategia no es óptima (contraejemplo)
- Longitud de la solución optima: $3 + 3 + 1 = 7$ $(v_{0,1}, v_{1,1}, v_{2,2}, v_{3,1})$

Cambio de monedas

Cambio de monedas

Problema del cambio de monedas

Se dispone de n monedas de euro con valores de 1,2,5,10,20 y 50 céntimos de euro, 1 y 2 euros.

- Dada una cantidad X de euros, devolver dicha cantidad con el menor número posible de monedas

- Ejemplo: devolver 2.24€
  - Solución: 4 $(2 + 0{,}20 + 0{,}02 + 0{,}02)$

Cambio de monedas

- Conjunto de candidatos:
  - Todos los tipos de monedas
- Función solución:
  - Conjunto de monedas que suman X
- Función de factibilidad:
  - Si $\sum_{i=1}^{8} v_i n_i > X$, el conjunto obtenido no podrá ser solución
    - $n_i =$ número de monedas de tipo i
    - $v_i =$ valor de una moneda de tipo i
- Función objetivo:
  - Minimizar la cardinalidad de las soluciones posibles
- Función de selección:
  - Moneda de valor más alto posible, que no supere el valor que queda por devolver

Cambio de monedas

- ¿Es óptima la función de selección propuesta?

Problema del cambio de monedas

Se dispone de n monedas de euro con valores de 1,2,5,10,**12**,20 y 50 céntimos de euro, 1 y 2 euros.

- Dada una cantidad X de euros, devolver dicha cantidad con el menor número posible de monedas

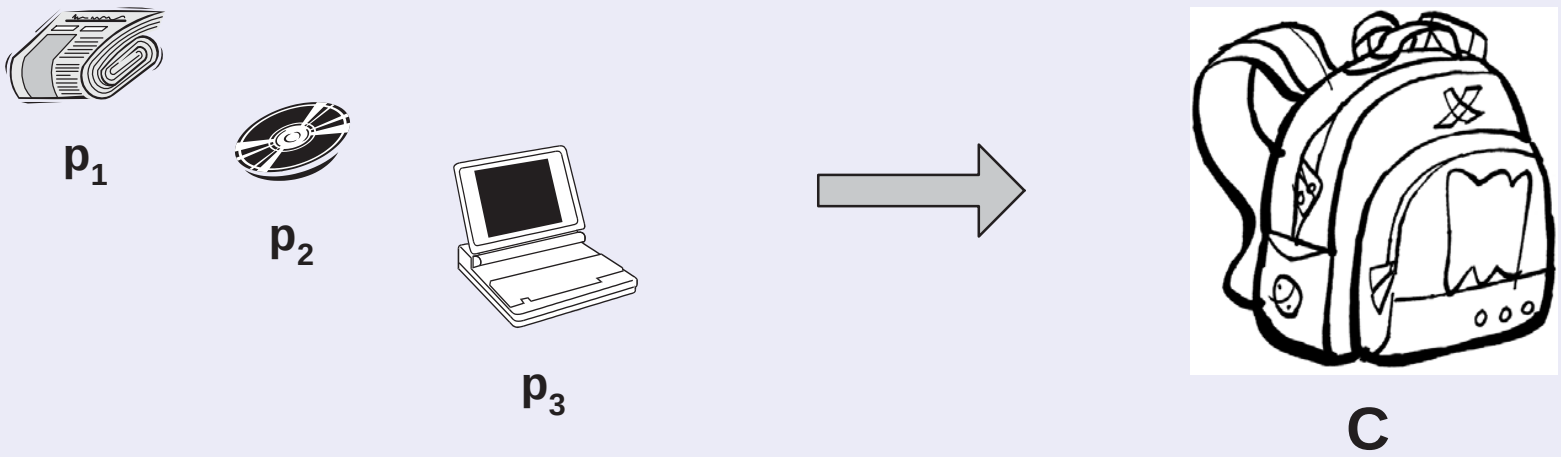
- Ejemplo: devolver 2.24€
  - Solución voraz: 4 $(2 + 0{,}20 + 0{,}02 + 0{,}02)$
  - Solución óptima: 3 $(2 + 0{,}12 + 0{,}12)$

Problema de la mochila

Problema de la mochila - 1

Problema de la mochila - Versión 1

Se tiene un conjunto de n objetos, cada uno con un peso p_i, y una mochila con capacidad C.



- Maximizar el **número de objetos** que se pueden introducir en la mochila sin sobrepasar la capacidad C

Problema de la mochila - 1

Problema de la mochila - Versión 1 - Formulación matemática

Dado un conjunto de n objetos, cada uno con un peso p_i, $i = 1, \ldots, n$, y una mochila con capacidad C. Maximizar el **número de objetos** que se pueden introducir en la mochila sin sobrepasar la capacidad C:

$$\begin{array}{ll} \underset{x}{\textit{maximizar}} & \sum_{i=1}^{n} x_i \\ \text{sujeto a} & x_i \in \{0,1\} \qquad i = 1, \ldots, n \\ & \sum_{i=1}^{n} x_i p_i \leq C \end{array}$$

- Las variables x_i determinan si se introduce el objeto i en la mochila
- Es un problema de programación lineal entera

Problema de la mochila - 1

- Ejemplo: $C = 15$, $p = (9, 6, 5)$
- ¿Función de selección?
  - Peso decreciente
    - Solución: 2 $(9 + 6)$
  - Peso creciente
    - Solución: 2 $(5 + 6)$

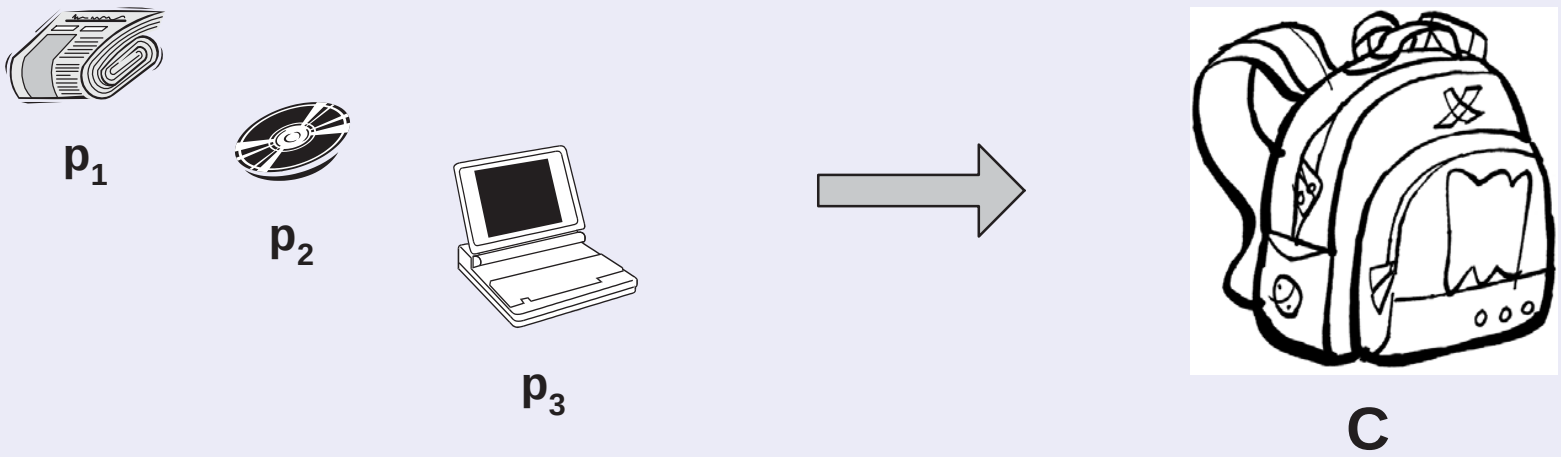
- Ejemplo: $C = 15$, $p = (9, 5, 6, 4)$
- ¿Función de selección?
  - Peso decreciente
    - Solución: 2 $(9 + 6)$
  - Peso creciente
    - Solución: 3 $(4 + 5 + 6)$

- La estrategia voraz escogiendo candidatos en orden creciente de peso es mejor (para este problema es óptima)

Problema de la mochila - 2

Problema de la mochila - Versión 2

Se tiene un conjunto de n objetos, cada uno con un peso p_i, y una mochila con capacidad C.



- Maximizar el **peso de los objetos** que se introducen en la mochila sin sobrepasar la capacidad C

Problema de la mochila - 2

Problema de la mochila - Versión 2 - Formulación matemática

Dado un conjunto de n objetos, cada uno con un peso p_i, $i = 1, \ldots, n$, y una mochila con capacidad C. Maximizar el **peso de los objetos** que se introducen en la mochila sin sobrepasar la capacidad C:

$$\underset{x}{\text{maximizar}} \quad \sum_{i=1}^{n} x_i p_i$$

$$\text{sujeto a} \quad x_i \in \{0, 1\} \qquad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_i p_i \leq C$$

- Las variables x_i determinan si se introduce el objeto i en la mochila
- Es un problema de programación lineal entera

Problema de la mochila - 2

- Ejemplo: $C = 15$, $p = (9, 6, 5)$
- ¿Función de selección?
  - Peso decreciente
    - Solución: 15 $(9 + 6)$
  - Peso creciente
    - Solución: 11 $(5 + 6)$

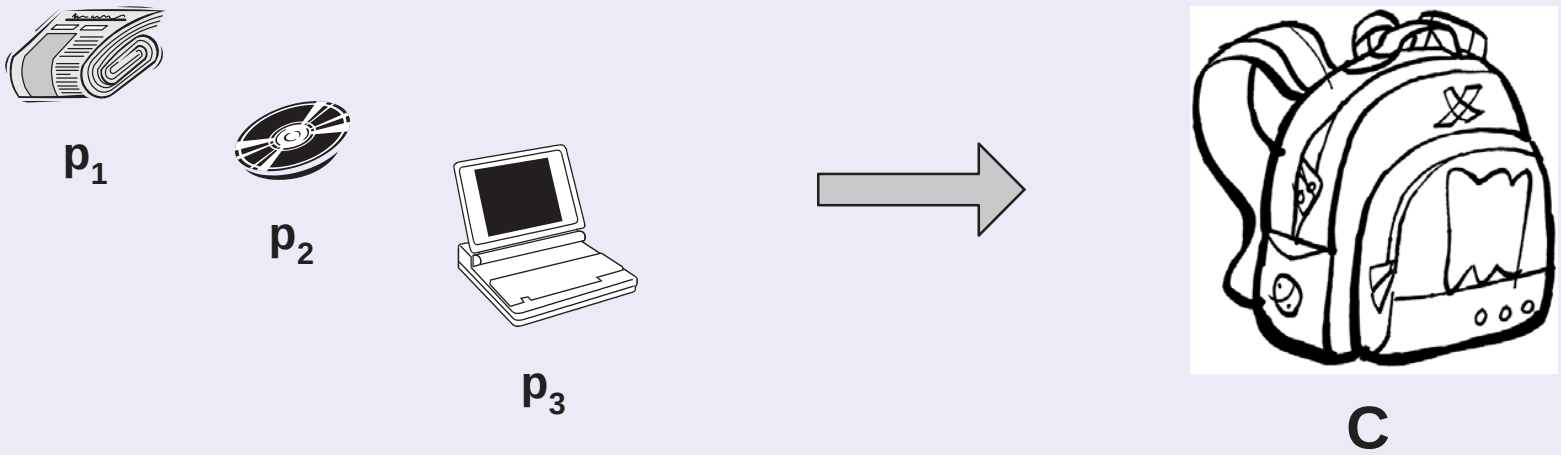
- Ejemplo: $C = 15$, $p = (10, 7, 6)$
- ¿Función de selección?
  - Peso decreciente
    - Solución: 10 (10)
  - Peso creciente
    - Solución: 13 $(6 + 7)$

- Ninguna de las estrategias de selección es óptima, ni mejor que la otra

Problema de la mochila - 3

Problema de la mochila - Versión 3

Se tiene un conjunto de n objetos, cada uno con un peso p_i y un valor v_i, y una mochila con capacidad C. Los objetos pueden partirse en fracciones más pequeñas.



- Maximizar la **suma de los valores asociados a los objetos** que se introducen en la mochila, sin sobrepasar la capacidad C

Problema de la mochila - 3

Problema de la mochila - Versión 3 - Formulación matemática

Dado un conjunto de n objetos, cada uno con un peso p_i y un valor v_i, $i = 1, \ldots, n$, y una mochila con capacidad C. Maximizar la suma de los valores asociados a los objetos que se introducen en la mochila, sin sobrepasar la capacidad C, sabiendo que los objetos pueden partirse en fracciones más pequeñas:

$$\underset{x}{\text{maximizar}} \quad \sum_{i=1}^{n} x_i v_i$$

$$\text{sujeto a} \quad 0 \leq x_i \leq 1 \qquad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_i p_i \leq C$$

- Las variables x_i determinan la fracción del objeto i que se introduce en la mochila
- Es un problema de programación lineal

Problema de la mochila - 3

- Conjunto de candidatos:
  - Todos los objetos
- Función de factibilidad:
  - $\sum_{i=1}^{n} x_i p_i \leq C$
- Función objetivo:
  - Maximizar $\sum_{i=1}^{n} x_i v_i$
- Funciones de selección posibles:
  - Seleccionar el objeto con mayor valor
  - Seleccionar el objeto con menor peso restante
  - Seleccionar el objeto cuyo valor por unidad de peso sea el mayor posible
- Función solución:
  - Cualquier conjunto de elementos es válido si no se ha sobrepasado C

Problema de la mochila - 3

- Ejemplo: $C = 100$, $n = 5$, con:

i	1	2	3	4	5
p_i	10	20	30	40	50
v_i	20	30	66	40	60
v_i/p_i	2,0	1,5	2,2	1,0	1,2

seleccionar x_i	x_i					valor total
maximizar v_i	0	0	1	0,5	1	146
minimizar p_i	1	1	1	1	0	156
maximizar v_i/p_i	1	1	1	0	0,8	164

- Función de selección óptima es: maximizar v_i/p_i

Problema de la mochila - 4

Problema de la mochila - Versión 4 - Formulación matemática

Dado un conjunto de n objetos, cada uno con un peso p_i y un valor v_i, $i = 1, \ldots, n$, y una mochila con capacidad C. Maximizar la suma de los valores asociados a los objetos que se introducen en la mochila, sin sobrepasar la capacidad C, sabiendo que los objetos **NO** pueden partirse en fracciones más pequeñas:

$$\underset{x}{\text{maximizar}} \quad \sum_{i=1}^{n} x_i v_i$$

$$\text{sujeto a} \quad x_i \in \{0, 1\} \qquad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_i p_i \leq C$$

- Las variables x_i determinan si se introduce el objeto i en la mochila
- Es un problema NP-completo (Problema de la mochila 0-1)

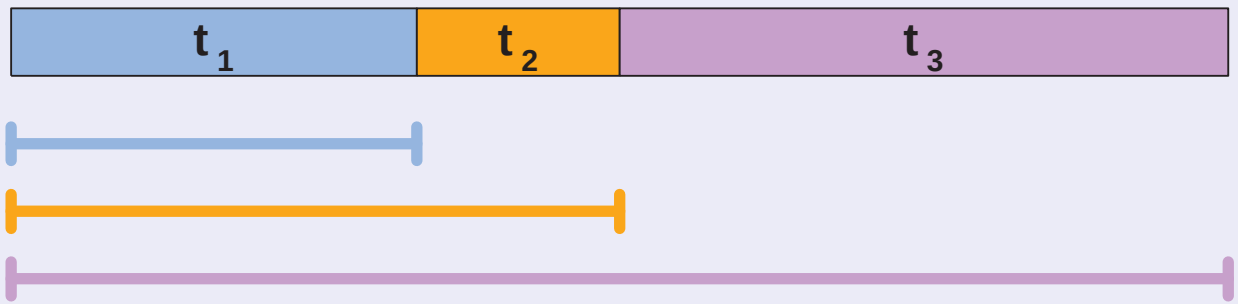
Problemas de planificación de tareas

Minimización del tiempo en el sistema

Minimización del tiempo en el sistema

Considérese un servidor que tiene que dar servicio a n clientes, donde t_i, con $i = 1, \ldots, n$, es el tiempo requerido por el cliente i. Suponiendo que todos los clientes llegan al mismo tiempo al servidor pero solo uno puede usarlo, minimizar el tiempo T en el sistema para los n clientes:

$$T = \sum_{i=1}^{n} (\text{tiempo en el sistema para el cliente } i)$$



Ejemplo

- Supongamos que tenemos 3 clientes con $t_1 = 5$, $t_2 = 10$, y $t_3 = 3$, hay varias posibilidades según el orden en el que sean tratados en el servidor

Orden	T	
123:	$5 + (5 + 10) + (5 + 10 + 3) = 38$	
132:	$5 + (5 + 3) + (5 + 3 + 10) = 31$	
213:	$10 + (10 + 5) + (10 + 5 + 3) = 43$	$\leftarrow$ peor planificación
231:	$10 + (10 + 3) + (10 + 3 + 5) = 41$	
312:	$3 + (3 + 5) + (3 + 5 + 10) = 29$	$\leftarrow$ mejor planificación
321:	$3 + (3 + 10) + (3 + 10 + 5) = 34$	

Estrategia voraz

- Dar servicio en orden creciente de tiempo t_i
- Es una estrategia óptima:
  - Sea $P = \langle p_1, p_2, \ldots, p_n \rangle$ una permutación de los clientes (de enteros de 1 a n)
  - Sea $s_i = t_{p_i}$, entonces

$$\begin{aligned} T(P) &= s_1 + (s_1 + s_2) + (s_1 + s_2 + s_3) + \ldots \\ &= n s_1 + (n-1) s_2 + (n-2) s_3 + \ldots \\ &= \sum_{k=1}^{n} (n - k + 1) s_k \end{aligned}$$

  - Haciendo s_1 lo menor posible, luego s_2, etc. conseguimos minimizar T

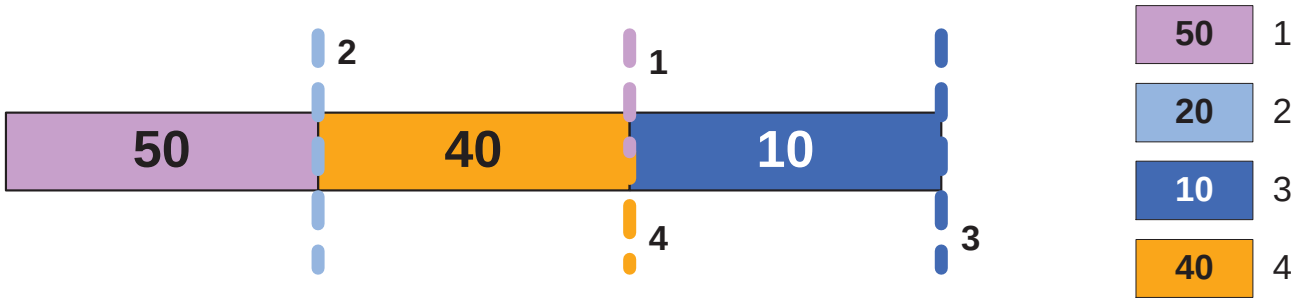
Secuencia de tareas con plazo

Secuencia de tareas con plazo

- Se van a ejecutar un subconjunto de n tareas que requieren una unidad de tiempo de ejecución
- Las tareas solo se ejecutan una vez
- En cualquier instante $T = 1, 2, \ldots$ se puede ejecutar una única tarea i, que aportará un beneficio g_i
- Dar una planificación de tareas que **maximice el beneficio**, sabiendo que la tarea i debe terminar de ejecutarse no más tarde que d_i (restricción de plazos)

Ejemplo

i	1	2	3	4
g_i	50	20	10	40
d_i	2	1	3	2



Estrategia voraz

- Conjuntos factibles: todos los que tienen 3 elementos o menos, excepto el $\{1, 2, 4\}$ (ya que la tercera tarea siempre ha de ser la 3)

conjuntos factibles	orden de proceso	beneficio
$\{1, 2, 3\}$	$2 - 1 - 3$	80
$\{1, 2\}$	$2 - 1$	70
$\{1, 3, 4\}$	$\boxed{1 - 4 - 3}$ y $4 - 1 - 3$	$\boxed{100}$
$\{1, 3\}$	$1 - 3$ y $3 - 1$	60
$\{1, 4\}$	$1 - 4$ y $4 - 1$	90
$\{1\}$	1	50
$\{2, 3, 4\}$	$2 - 4 - 3$	80
$\vdots$	$\vdots$	$\vdots$

- **Estrategia óptima**: escoger la tarea con mayor beneficio g_i que no se haya elegido previamente, siempre que el resto de tareas en el subconjunto permanezcan factibles
  - Una vez escogida la tarea 1 como la óptima para la 1^a etapa, se obtiene un nuevo problema idéntico pero más pequeño (una tarea menos)

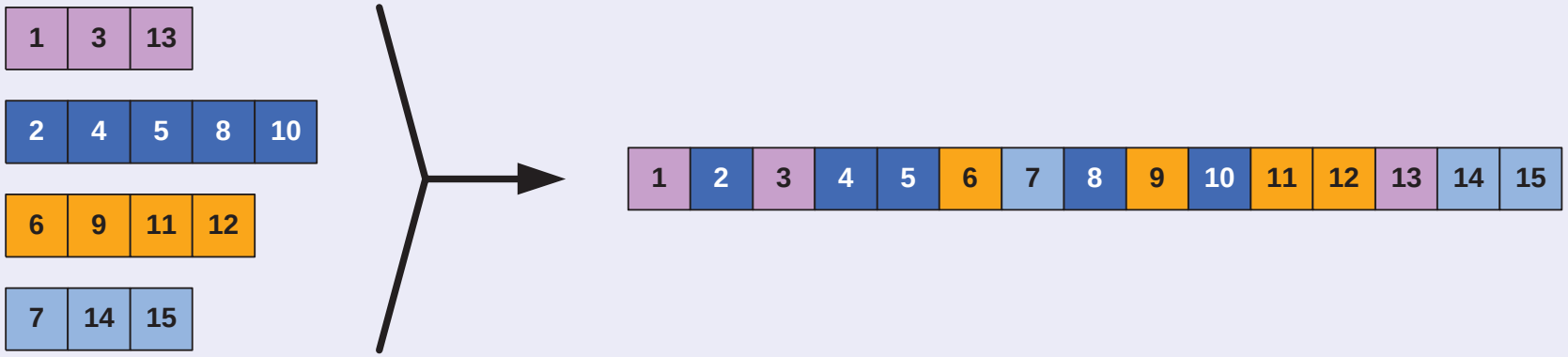
Patrones óptimos de mezcla

Mezcla de listas

Mezcla de listas

Sean m listas ordenadas, cada una con n_i elementos ($i = 1, \ldots, m$)

- ¿Cuál es la sucesión óptima del proceso de mezcla, mezclando listas dos a dos, para mezclar todas las listas de manera ordenada con el menor número de comparaciones?

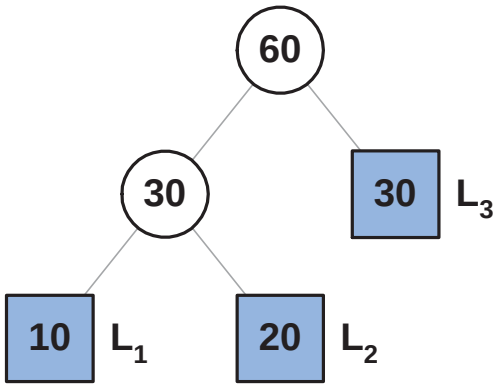


Ejemplo

- Sean tres listas L_1, L_2, y L_3, con longitudes 10, 20, y 30, respectivamente

$L_1 + L_2 = 30$	$L_1 + L_3 = 40$	$L_2 + L_3 = 50$
$30 + L_3 = 60$	$40 + L_2 = 60$	$50 + L_1 = 60$
$90 \Leftarrow$	100	110

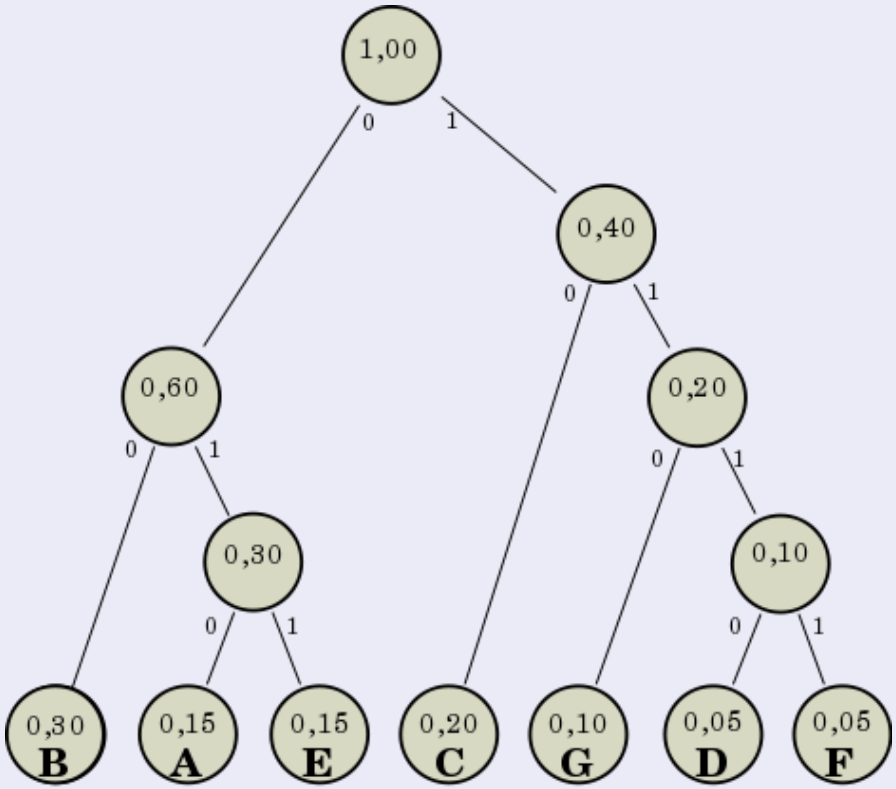
- Solución voraz: escoger en cada momento los dos lotes de menor tamaño
  - Construye un árbol binario de abajo hacia arriba (árbol de mezclas)



Códigos de Huffman

Códigos de Huffman

Dado un conjunto de n objetos y sus respectivas frecuencias de aparición f_i, $i = 1, \ldots, n$, obtener una codificación binaria para los objetos que minimice el promedio de bits necesarios para representarlos.



Códigos de Huffman

- La codificación Huffman es un método usado para compresión de datos
- Se codifica una serie de objetos mediante una secuencia de bits según su frecuencia de aparición
- Los objetos con mayor frecuencia se codifican con menos bits
- La codificación Huffman minimiza el promedio de bits necesarios para representar los objetos

Ejemplo

- Tenemos un texto de 100000 caracteres
- Solo aparecen 6 caracteres con las siguientes frecuencias absolutas de aparición

carácter	a	b	c	d	e	f
frecuencia	45000	13000	12000	16000	9000	5000

- Solución 1: Usar 3 bits por cada carácter $\Rightarrow$ 300000 bits
- Solución 2: Usar códigos de longitud variable en el que los más frecuentes tienen el código más corto $\Rightarrow$ 224000 bits

Objetivo: crear una codificación de longitud variable para minimizar el número de bits, en promedio, necesarios para representar un texto

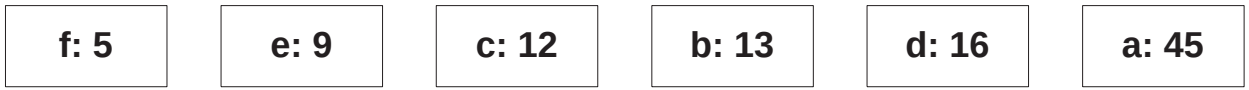
Estrategia voraz

- **Asignar códigos más largos a los caracteres con menor frecuencia de aparición**

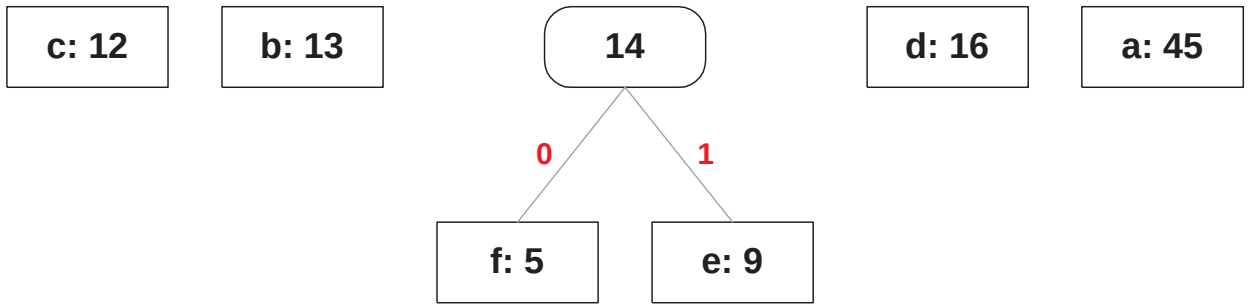
- Utiliza la tabla de frecuencias de aparición de cada carácter
- Construye un árbol binario de longitud variable de abajo hacia arriba
- Utiliza una cola Q de árboles con prioridades
- Inicialmente Q contiene un árbol por cada carácter
- En cada paso, se "mezclan" los dos árboles de Q que tienen menos frecuencia dando lugar a un nuevo árbol

Estrategia voraz

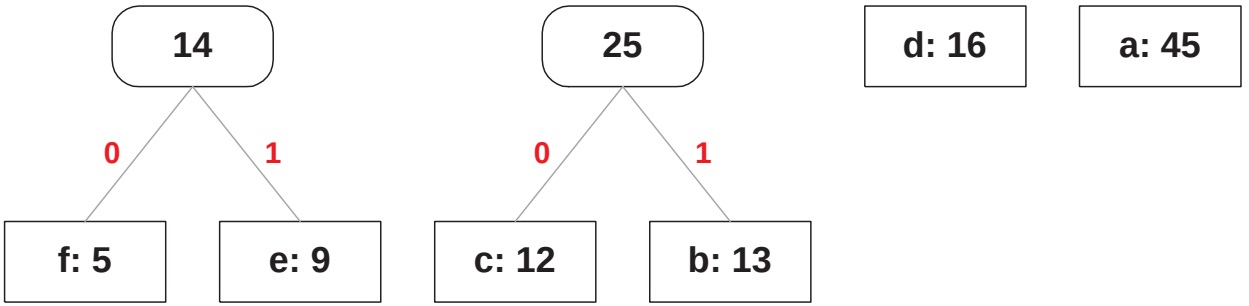
- **Fase 1**
  - Orden creciente de frecuencias (en miles)

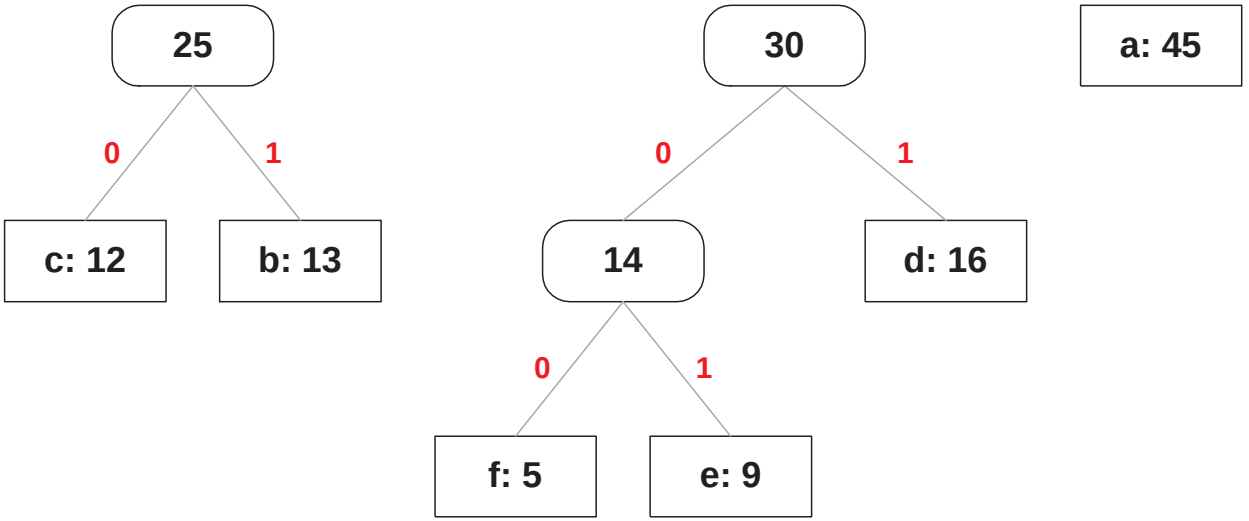


- **Fase 2 y posteriores**
  - Fusionar árboles hasta obtener un sólo árbol manteniendo la ordenación creciente

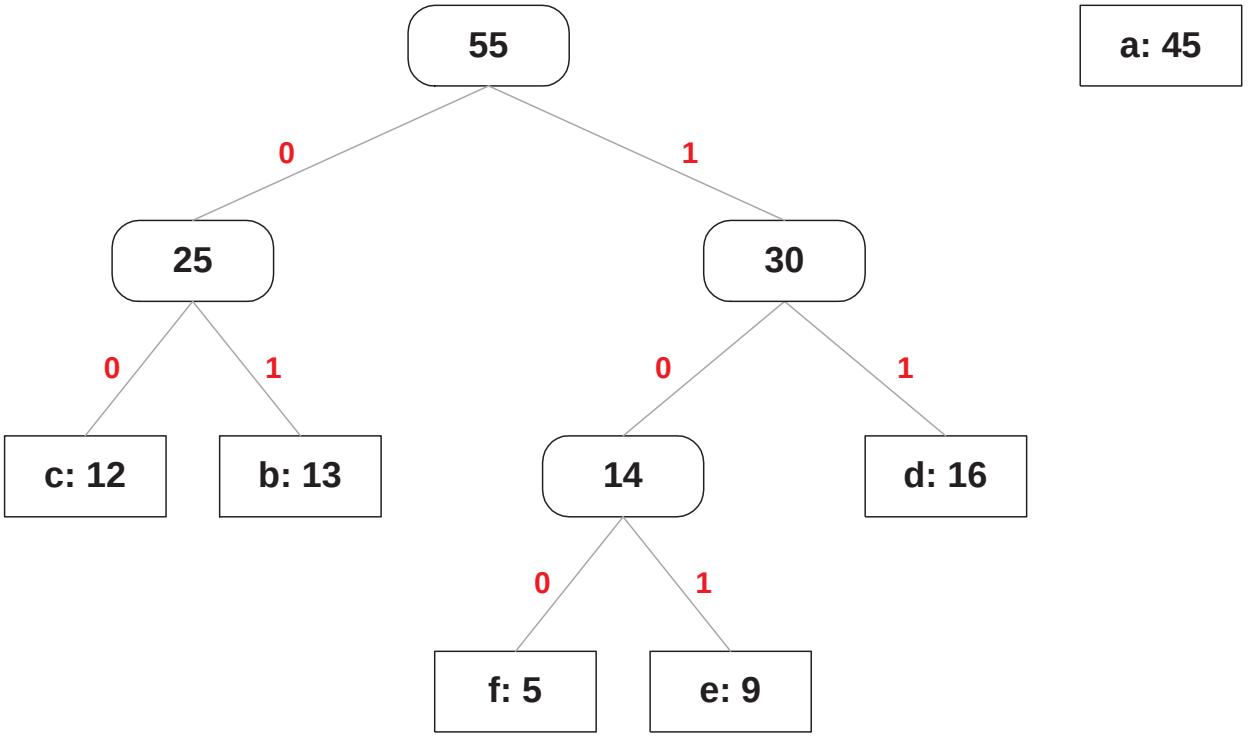


Estrategia voraz

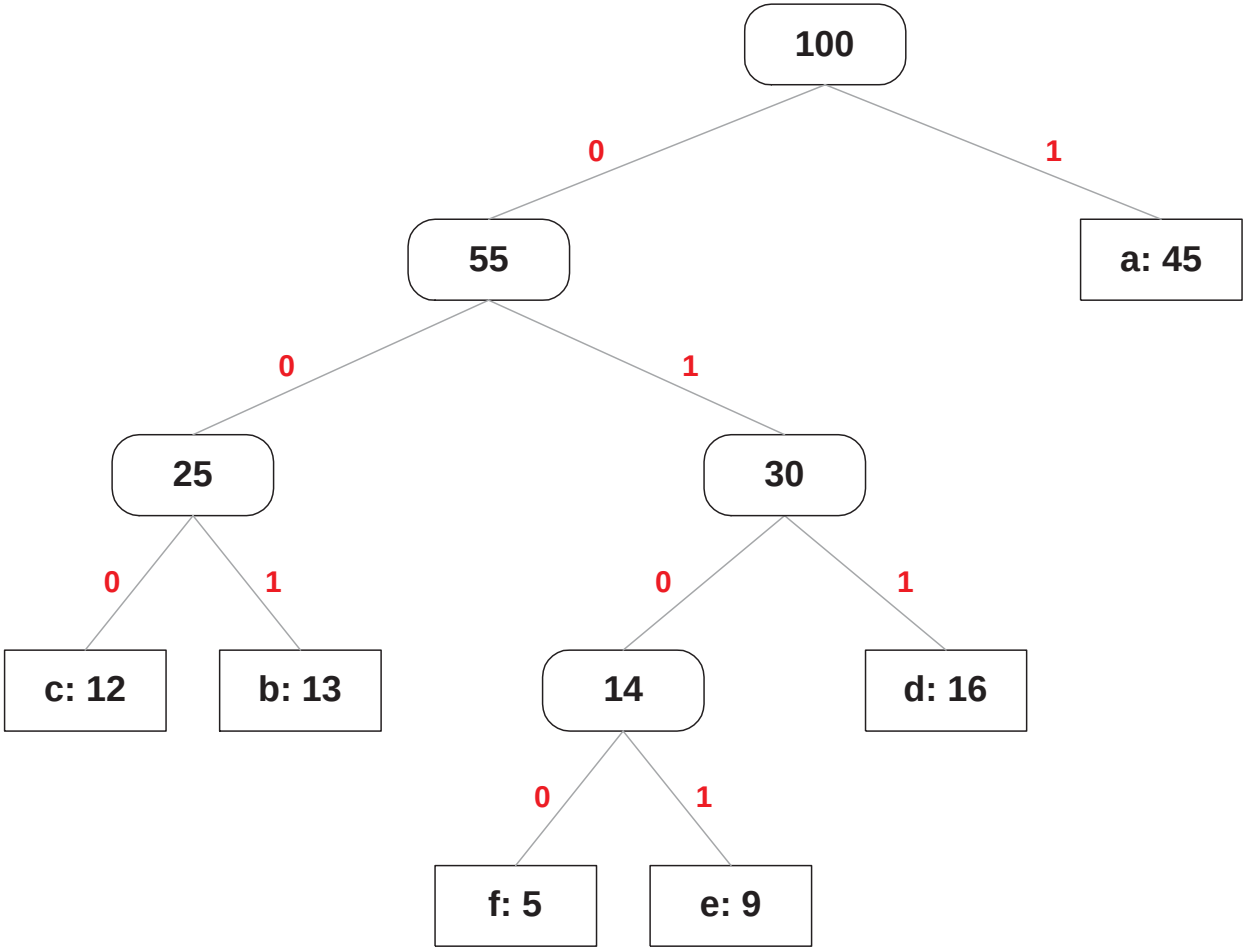




Estrategia voraz



Estrategia voraz



Codificación final

- Para hallar la codificación de un carácter se usa el camino desde la raíz del árbol hasta la hoja que representa el carácter
- La secuencia de bits la determina la rama (izquierda: 0, derecha: 1) por la que se avanza hasta la hoja
- Codificación final:

carácter	código
a	1
d	011
e	0101
f	0100
b	001
c	000

Árbol de recubrimiento de coste mínimo

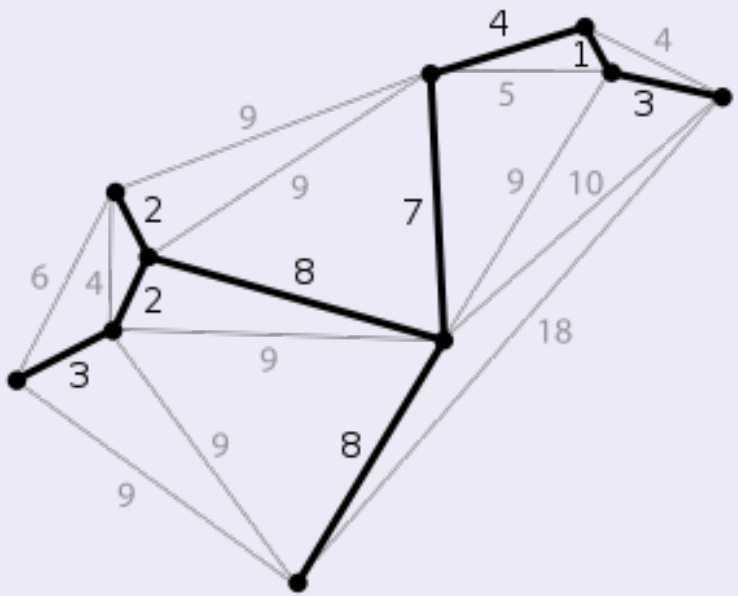
Terminología

- Árbol libre: es un grafo no dirigido conexo acíclico:
  - Todo árbol libre con n vértices tiene n-1 aristas
  - Si se añade una arista se introduce un ciclo
  - Si se borra una arista quedan vértices no conectados
  - Cualquier par de vértices está unido por un único camino simple
- Árbol de recubrimiento de un grafo no dirigido y ponderado no negativamente: es cualquier subgrafo que contenga todos los vértices y que sea un árbol libre
- Árbol de recubrimiento de coste mínimo: es un árbol de recubrimiento y no hay ningún otro árbol de recubrimiento cuya suma de los pesos de las aristas sea menor. Lo denotamos $arm(G)$, donde G es un grafo no dirigido, conexo, y ponderado no negativamente.

Árbol de recubrimiento de coste mínimo

Árbol de recubrimiento de coste mínimo

Dado un grafo $G = \langle V, E \rangle$ no dirigido, conexo, y ponderado no negativamente, hallar $arm(G)$.



- Valor 38 en la figura
- No tiene por qué ser único

Propiedad fundamental

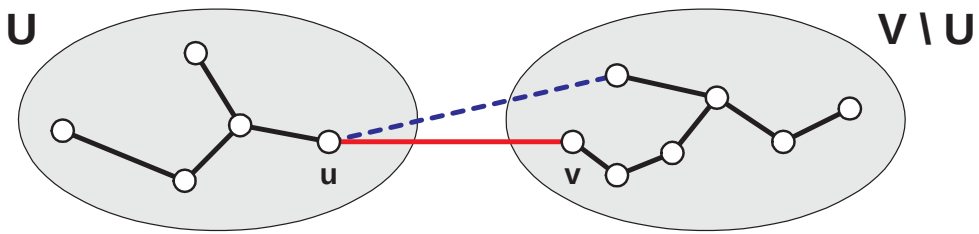
- Sea $G = \langle V, E \rangle$ un grafo no dirigido, conexo, y ponderado no negativamente,

$$G \in \{f : V \times V \longrightarrow E\}$$

- Sea U un conjunto de vértices $U \subset V$, $U \neq \emptyset$

Si $\langle u, v \rangle$ es la arista más pequeña de G tal que $u \in U$, y $v \in V \setminus U$, entonces existe algún árbol de recubrimiento de coste mínimo de G que la contiene

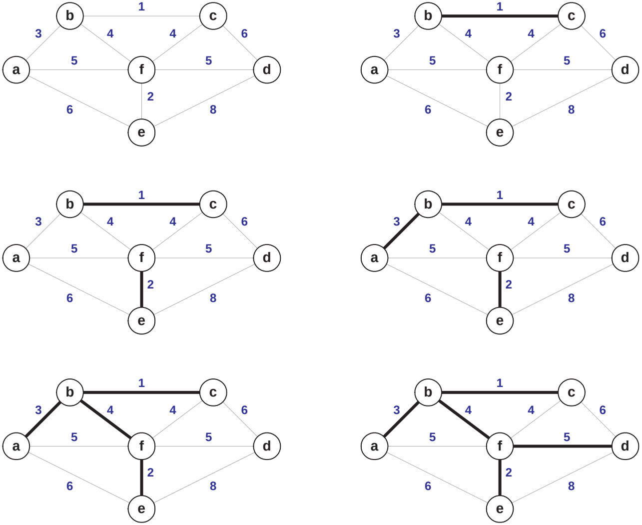
  - Demostración por reducción al absurdo (contradicción)



Algoritmo de Kruskal

- Seleccionar la arista más corta en cada etapa y construir el árbol
- Se basa en la propiedad de los árboles de recubrimiento de coste mínimo: partiendo del árbol vacío, se selecciona en cada paso la arista de menor peso que no provoque ciclo sin requerir ninguna otra condición sobre sus extremos
- Es una estrategia óptima

Algoritmo de Kruskal paso a paso



Algoritmo de Kruskal - Pseudocódigo

Kruskal(G)
    // Entrada: $G = \langle V, E \rangle$, es un grafo no dirigido, conexo,
        // y ponderado no negativamente
    // Salida: E_T = conjunto de aristas que forman un $arm(G)$

    ordenar E en orden no decreciente según los pesos asociados a las aristas:
        $w(e_{i_1}) \leq \ldots \leq w(e_{i_{|E|}})$
    $E_T \leftarrow \emptyset$
    $cont \leftarrow 0$ // nº de aristas de $arm(G)$
    $k \leftarrow 0$ // nº de aristas procesadas

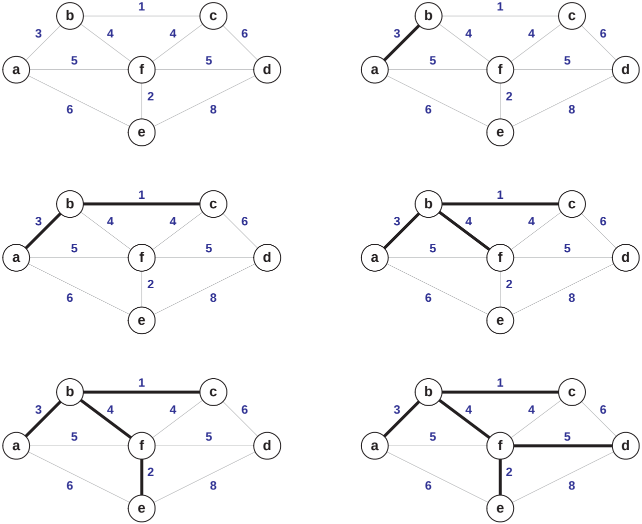
    while $cont < |V| - 1$
        $k \leftarrow k + 1$
        si $E_T \cup \{e_{i_k}\}$ no contiene ciclos
            $E_T \leftarrow E_T \cup \{e_{i_k}\}$
            $cont \leftarrow cont + 1$

    devolver E_T

Algoritmo de Prim

- Seleccionar un nodo y construir el árbol a partir de él
- Aplica reiteradamente la propiedad de los árboles de recubrimiento de coste mínimo incorporando a cada paso una arista
- Se usa un conjunto U de vértices tratados y se selecciona en cada paso la arista mínima que une un vértice de U con otro de su complementario
- Es una estrategia óptima

Algoritmo de Prim paso a paso



Algoritmo de Prim - Pseudocódigo

Prim(G)
    // Entrada: $G = \langle V, E \rangle$, es un grafo no dirigido, conexo
        // (puede usar pesos negativos)
    // Salida: E_T = conjunto de aristas que forman un $arm(G)$

    $V_T \leftarrow \{v_0\}$ // v_0 puede ser cualquier vértice
    $E_T \leftarrow \emptyset$

    desde $i \leftarrow 1$ **hasta** $|V| - 1$ **hacer**
        encontrar una arista $e^* = (v^*, u^*)$ de peso mínimo entre todas
            las aristas (v, u) tales que $v \in V_T$ y $u \in V \setminus V_T$
        $V_T \leftarrow V_T \cup \{u^*\}$
        $E_T \leftarrow E_T \cup \{e^*\}$

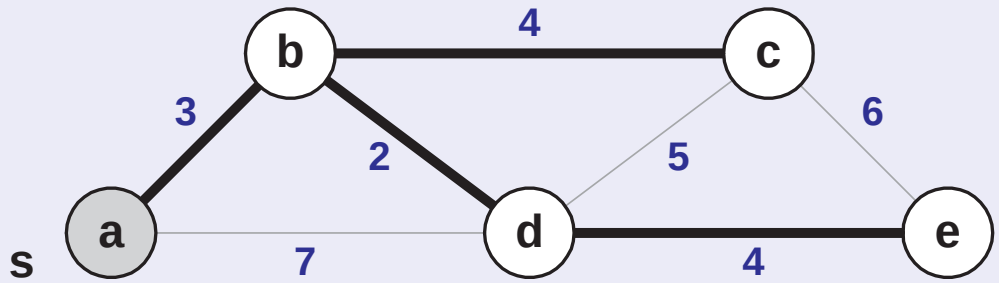
    devolver E_T

Caminos mínimos desde un nodo

Caminos mínimos desde un nodo

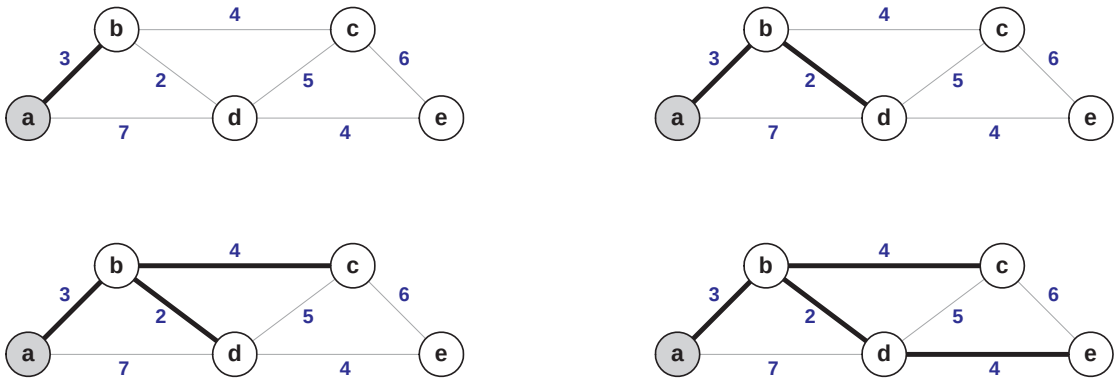
Caminos mínimos desde un nodo

Dado un grafo $G = \langle V, E \rangle$ conexo y ponderado no negativamente, y un vértice $s \in V$, hallar la longitud de los caminos mínimos desde s al resto de vértices.



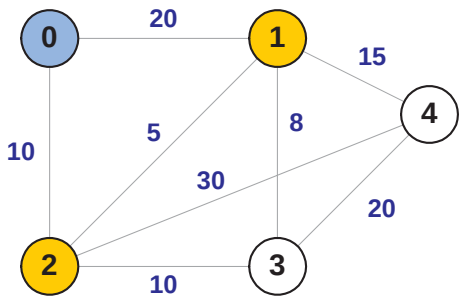
- Sirve para grafos dirigidos y no dirigidos (si es no dirigido se puede sustituir cada arista por dos con el mismo peso, y en "direcciones" contrarias)
- Veremos el algoritmo de Dijkstra

Algoritmo de Dijkstra paso a paso

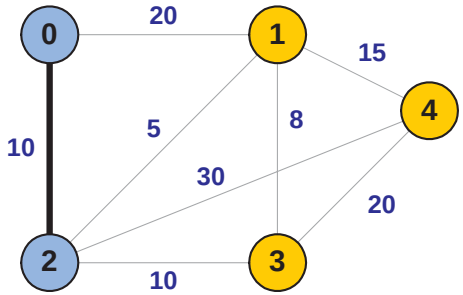


- Inserta vértices en un árbol T progresivamente, según calcula nuevas longitudes mínimas
  - El árbol T es conceptual (es una lista de nodos)
- Para un nuevo vértice tiene en cuenta no sólo la longitud de la nueva arista, sino también todo el camino hasta dicho vértice
  - Es la diferencia con respecto al algoritmo de Prim
- En cada paso inserta en T el vértice con menor longitud desde el origen (estrategia voraz óptima)
- Tras insertar un nuevo vértice v_{sig}, debe recalcular distancias de nodos conectados a v_{sig} y T

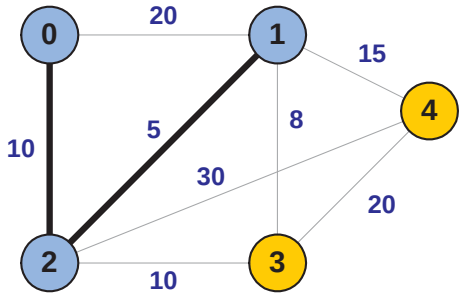
Algoritmo de Dijkstra en detalle



i	Camino	d_i	$\in T$
0	0	0	✓
1	0, 1	20	✗
2	0, 2	10	✗
3	0, 3	∞	✗
4	0, 4	∞	✗

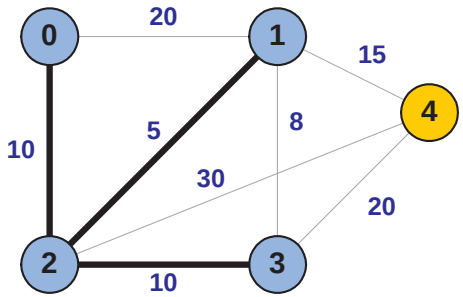


i	Camino	d_i	$\in T$
0	0	0	✓
1	0, 2, 1	15	✗
2	0, 2	10	✓
3	0, 2, 3	20	✗
4	0, 2, 4	40	✗

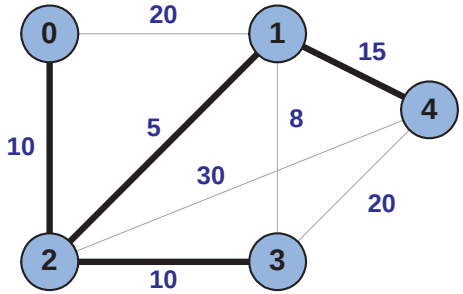


i	Camino	d_i	$\in T$
0	0	0	✓
1	0, 2, 1	15	✓
2	0, 2	10	✓
3	0, 2, 3	20	✗
4	0, 2, 1, 4	30	✗

Algoritmo de Dijkstra en detalle



i	Camino	d_i	$\in T$
0	0	0	✓
1	0, 2, 1	15	✓
2	0, 2	10	✓
3	0, 2, 3	20	✓
4	0, 2, 1, 4	30	✗



i	Camino	d_i	$\in T$
0	0	0	✓
1	0, 2, 1	15	✓
2	0, 2	10	✓
3	0, 2, 3	20	✓
4	0, 2, 1, 4	30	✓

Nota: el ejemplo usa un grafo no dirigido

Algoritmo de Dijkstra - pseudocódigo

Dijkstra(G, s, t)
    // Entrada: $G = \langle V, E \rangle$, es un grafo no dirigido, conexo,
        // y ponderado no negativamente con pesos w entre aristas
    // Salida: Distancias mínima d desde el vértice s hasta el t,
        // y todas las distancias mínimas a vértices que son menores que d

    $T \leftarrow \{s\}$
    desde $i \leftarrow 1$ **hasta** $|V|$ **hacer** $d_i \leftarrow \infty$
    para cada arista (s, v) **hacer** $d_v = w(s, v)$
    $ultimo \leftarrow s$
    mientras ($ultimo \neq t$) **hacer**
        seleccionar $v_{sig} \notin T$ // el vértice desconocido que minimiza d
        para cada arista (v_{sig}, x) **hacer**
            $d_x \leftarrow \text{mín}[d_x, d_{v_{sig}} + w(v_{sig}, x)]$
        $ultimo \leftarrow v_{sig}$
        $T \leftarrow T \cup \{v_{sig}\}$

    devolver d