



UNIVERSIDAD
DE GRANADA



Algorítmica

Grado en Ingeniería Informática

Tema 3 – Algoritmos Voraces (Greedy)

Este documento está protegido por la Ley de Propiedad Intelectual (Real Decreto Ley 1/1996 de 12 de abril). Queda expresamente prohibido su uso o distribución sin autorización del autor.

Manuel Pegalajar Cuéllar

manupc@ugr.es

Departamento de Ciencias de la
Computación e Inteligencia Artificial

<http://decsai.ugr.es>

Objetivos del tema

- Plantearse la búsqueda de varias soluciones distintas para un mismo problema y evaluar la bondad de cada una de ellas.
- Tomar conciencia de la importancia del análisis de la eficiencia de un algoritmo como paso previo a su implementación en un lenguaje de programación.
- Comprender la técnica voraz (greedy, avance rápido) de resolución de problemas y los distintos casos que se pueden presentar en la resolución de problemas por esta técnica: obtención de la solución óptima, de una solución no óptima, o no obtención de la solución.
- Conocer los criterios de aplicación de cada una de las distintas técnicas de diseño de algoritmos.



Estudia este tema en...

- G. Brassard, P. Bratley, “Fundamentos de Algoritmia”, Prentice Hall, 1997, pp. 211-246
- J.L. Verdegay: Lecciones de Algorítmica. Editorial Técnica AVICAM (2017).

Anotación sobre estas diapositivas:

El contenido de estas diapositivas es esquemático y representa un apoyo para las clases presenciales teóricas. No se considera un sustituto para apuntes de la asignatura.

Se recomienda al alumno completar estas diapositivas con notas/apuntes propios, tomados en clase y/o desde la bibliografía principal de la asignatura.



UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces



1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Los algoritmos Greedy se caracterizan por:

- Construir la solución paso a paso, por etapas.
- En cada momento selecciona un movimiento de acuerdo con un criterio de selección (voracidad).
- No vuelve a considerar los movimientos ya seleccionados ni los modifica en posteriores etapas (miopía).
- Se necesita una función objetivo o criterio de optimalidad.

Ventajas

- Eficientes.
- Fáciles de diseñar.
- Fáciles de implementar.

Inconvenientes


- Pueden no alcanzar la solución óptima.
- Pueden no encontrar la solución, aunque exista.
- Es un enfoque poco elegante, si no se puede demostrar la optimalidad.

Ejemplo: El problema del cambio de monedas

- Supongamos que compramos un refresco en una máquina y, tras pagar, esta tiene que devolver un cambio n con el mínimo número de monedas ¿Qué algoritmo debería seguir la máquina?
- Supongamos en un primer momento que la máquina tiene puede dispensar un número infinito de cualquier tipo de moneda (1ctmo., 2 ctmos, etc.)



Ejemplo: El problema del cambio de monedas

 **C** es el conjunto de monedas de la máquina, **S** el conjunto de monedas a devolver y **s** la suma de estas monedas. Se debe devolver el cambio **n**.

$S = \emptyset; s = 0$

Mientras $s \neq n$ **hacer:**

$x =$ el mayor elemento de **C** tal que $s + x \leq n$

Si no existe ese elemento, **entonces**
Devolver “no encuentro la solución”

$S = S \cup \{\text{una moneda de valor } x\}$

$C = C \setminus \{x\}$

$s = s + x$

Fin-Mientras

Devolver **S**

Greedy y optimalidad: El problema del cambio de monedas

¿De verdad que el algoritmo anterior devuelve el mínimo número de monedas? **¡Hay que demostrarlo!**



- Si el algoritmo puede no dar la solución óptima, basta con proporcionar dicha solución frente a la óptima para demostrarlo (lo que se llama un **contraejemplo**).
- Si es óptimo, normalmente se puede demostrar por la técnica de **inducción** y/o por **reducción al absurdo**.

El algoritmo del cambio de monedas no es óptimo para todos los sistemas monetarios internacionales. Ejemplo del antiguo sistema español prescindiendo de duros: 100ptas, 50ptas, 25ptas, 10ptas y 1pta. ¿Darías un contraejemplo de la no optimalidad (Pista: $n=40$ ptas)?

Greedy y optimalidad

Un algoritmo Greedy puede incluso que no dé una solución al problema planteado, aunque esta exista. Ejemplo:

- La máquina tiene las siguiente monedas:

- De 50 ctmos.: 3

- De 20 ctmos.: 5

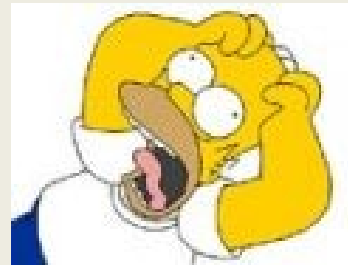
- De 1 ctmo.: 12

- De las demás, no quedan.

- Hay que **devolver 63 ctmos.**

- El algoritmo dice que no hay solución, ¡pero la hay!:

- 3x20 ctmos., 3x1 ctmo.





UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
- » 2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Pasos y requisitos para aplicar Greedy

Un algoritmo voraz podrá aplicarse siempre que se pueda...

- Diseñar una **lista de candidatos** para formar una solución.
- Identificar una **lista de candidatos ya utilizados**.
- Diseñar una **función solución** para saber cuándo un conjunto de candidatos es solución al problema.
- Diseñar un **criterio de factibilidad** (cuándo un conjunto de candidatos puede ampliarse para formar la solución final).
- Diseñar una **función de selección** del candidato más prometedor para formar parte de la solución
- Existe una **función objetivo** de minimización/maximización.



Plantilla de algoritmos Greedy

Una vez diseñadas las componentes Greedy, se aplicarán en el siguiente esquema, donde S es la solución al problema:

Función $S = \text{Voraz}(\text{vector candidatos } C)$

$S = \emptyset$

Mientras $(C \neq \emptyset)$ y $(S \text{ no es solución})$ **hacer:**

$x = \text{Selección de candidato de } C$

$C = C \setminus \{x\}$

Si es factible $(S \cup \{x\})$ **entonces**

$S = S \cup \{x\}$

Fin-Mientras

Si S es solución **entonces**

Devolver S

en otro caso

Devolver “No hay solución”

Ejemplo: Diseño de componentes del algoritmo de cambio de monedas

- **Lista de candidatos:** Las monedas posibles a devolver.
- **Lista de candidatos utilizados:** Las monedas que se han ido seleccionando para ser devueltas.
- **Función solución:** La suma de las monedas seleccionadas es la cantidad exacta que se debe devolver.
- **Función de selección:** La moneda de mayor valor que, sumada a las seleccionadas, no supere la cantidad a devolver.
- **Criterio de factibilidad:** La suma de las monedas actualmente seleccionadas no supera la cantidad que hay que devolver.
- **F. objetivo:** Minimizar el número de monedas que se devuelven para la cantidad n .

Ejemplo: Diseño del algoritmo de cambio de monedas

Plantilla:

Función S= Voraz(vector candidatos C)

$S = \emptyset$

Mientras $(C \neq \emptyset)$ y $(S$ no es solución)

hacer:

x = Selección de candidato de C

$C = C \setminus \{x\}$

Si es factible($S \cup \{x\}$) **entonces**
 $S = S \cup \{x\}$

Fin-Mientras

Si S es solución **entonces**

Devolver S

en otro caso

Devolver "No hay solución"

Algoritmo del cambio de monedas:

$S = \emptyset; s = 0$

Mientras $s \neq n$ **hacer:**

x = el mayor elemento de C tal que $s + x \leq n$

Si no existe ese elemento, **entonces**

Devolver "no encuentro la solución"

Fin-Si

$S = S \cup \{\text{una moneda de valor } x\}$

$C = C \setminus \{x\}$

$s = s + x$

Fin-Mientras

Devolver S



UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
- » 3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Enunciado del problema

Sea $G=(V,A)$ un grafo **no dirigido, conexo**, ponderado **con pesos no negativos**, con V el conjunto de vértices del grafo y A el conjunto de aristas del grafo. El problema consiste en

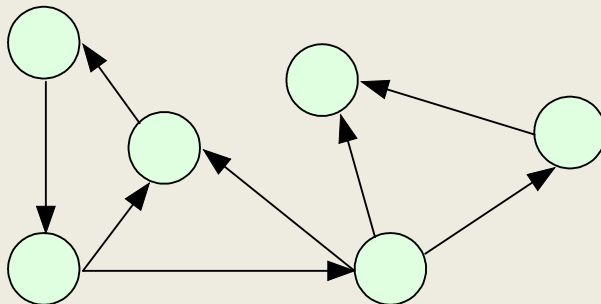
*“Obtener un **grafo parcial, conexo y acíclico**, tal que **la suma de sus aristas sea mínima**.”*

Repasaremos algunos conceptos de grafos antes...

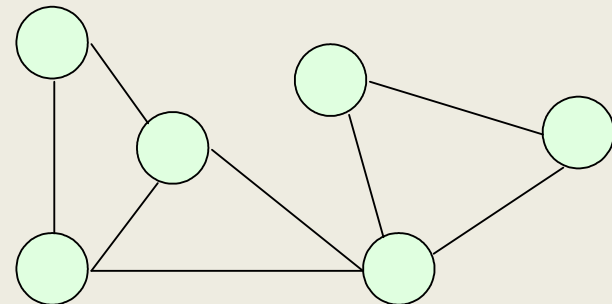


Repaso de conceptos sobre grafos

Grafo no dirigido: Para cada par de vértices v_i, v_j tal que existe una arista entre (v_i, v_j) , entonces también existe la arista (v_j, v_i) .



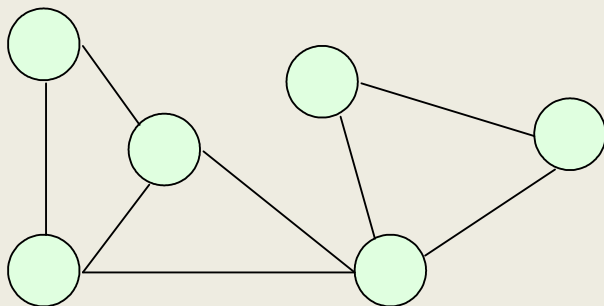
Grafo dirigido



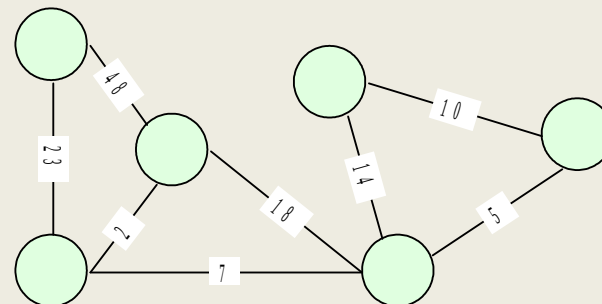
Grafo NO dirigido

Repaso de conceptos sobre grafos

Grafo ponderado: Las aristas tienen un valor numérico asociado. Se dice que es ponderado no negativo cuando este valor es 0 o mayor que 0.



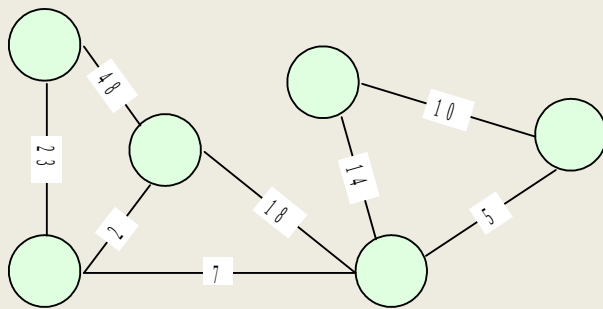
Grafo no ponderado



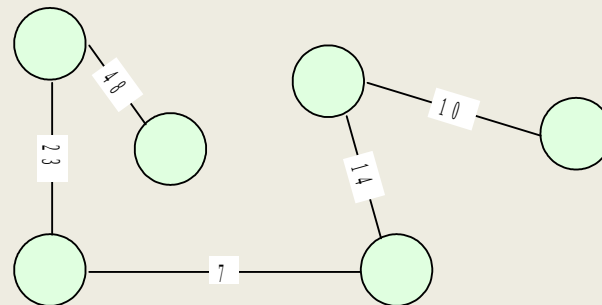
Grafo ponderado

Repaso de conceptos sobre grafos

Grafo parcial: Grafo obtenido a partir de un grafo original, eliminando aristas del mismo.



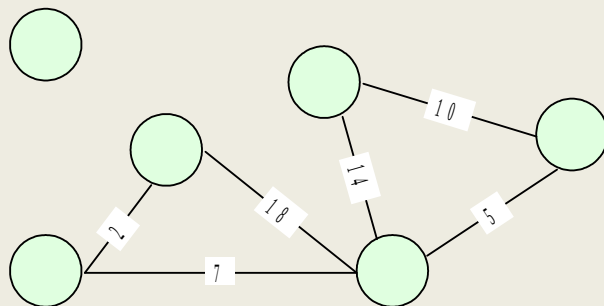
Grafo original



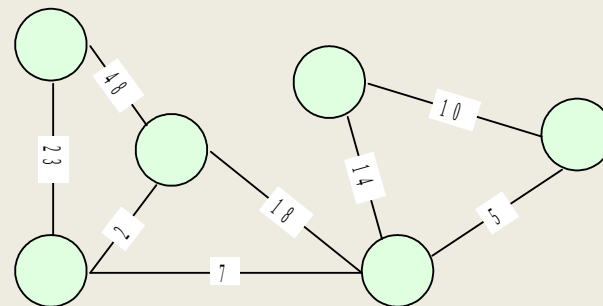
Grafo parcial

Repaso de conceptos sobre grafos

Grafo conexo: Para cada par de vértices del grafo, hay al menos un camino que los une. En grafos no conexos, cada subgrafo que sí que sea conexo se denomina **componente conexa** (*en el grafo de la izq. hay 2 componentes conexas*).



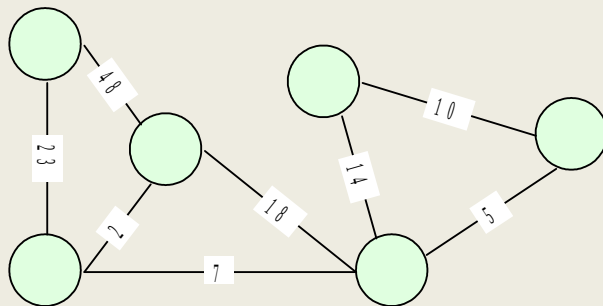
Grafo NO conexo



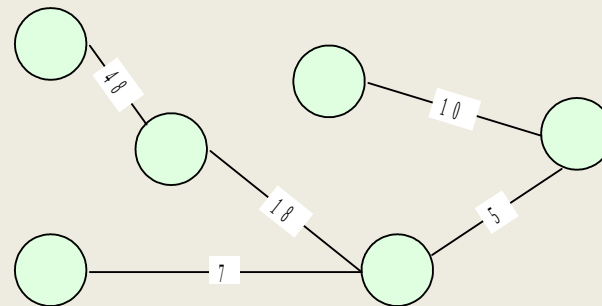
Grafo conexo

Repaso de conceptos sobre grafos

Grafo acíclico: No existen ciclos entre ningún subconjunto de vértices del grafo.



Grafo con ciclos



Grafo acíclico

Enunciado del problema

Sea $G=(V,A)$ un grafo **no dirigido, conexo**, ponderado **con pesos no negativos**, con V el conjunto de vértices del grafo y A el conjunto de aristas del grafo. El problema consiste en

*“Obtener un **grafo parcial, conexo y acíclico**, tal que **la suma de sus aristas sea mínima**.”*

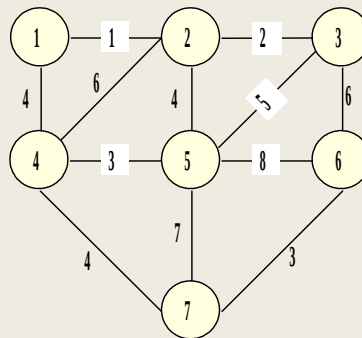
Estudiaremos dos alternativas:

- Algoritmo de **Kruskal**
- Algoritmo de **Prim**

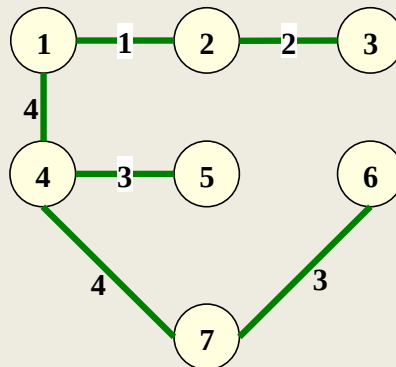


Entender el problema

Tenemos un grafo no dirigido, conexo y ponderado. Ejemplo:



Queremos un grafo parcial (mismos nodos, subconjunto de aristas), que no tenga ciclos y que una todos los nodos. **Requisito:** Que la suma del peso de las aristas sea mínimo. Ejemplo:



Algoritmo de Kruskal: Idea general

Inicialmente, la solución será un conjunto de aristas vacío.

En cada paso, iremos construyendo la solución final seleccionando la arista de menor coste de entre las aristas que forman la lista de candidatos.

Si dicha arista no forma ciclos con las ya escogidas, será añadida a la solución. En caso contrario, será descartada.

Como en cada paso vamos seleccionando la arista de menor coste, conviene tenerlas ordenadas previamente, para mejorar la eficiencia.

Algoritmo de Kruskal: Diseño de componentes Greedy

- **Lista de candidatos:** Las aristas del grafo.
- **Lista de candidatos utilizados:** Las aristas que se han ido seleccionando desde el grafo original, hayan sido añadidas a la solución o no.
- **Función solución:** Las aristas seleccionadas unen todos los nodos del grafo, sin formar ciclos.
- **Función de selección:** Se selecciona la arista de menor coste.
- **Criterio de factibilidad:** El conjunto de aristas seleccionadas no forma ciclos.
- **F. objetivo:** Minimizar la suma de los pesos de las aristas que forman la solución.



Algoritmo de Kruskal: Diseño del algoritmo Greedy

ALGORITMO $T = \text{Kruskal}(\text{grafo } G=(V,A))$

Ordenar A por orden creciente de pesos

N = Número de vértices en V

$T = \emptyset$ // T es la Solución a construir

Repetir:

a = Arista de A más corta no considerada

$A = A \setminus \{a\}$

Si $(T \cup \{a\})$ no forma ciclos en V **entonces**

$T = T \cup \{a\}$

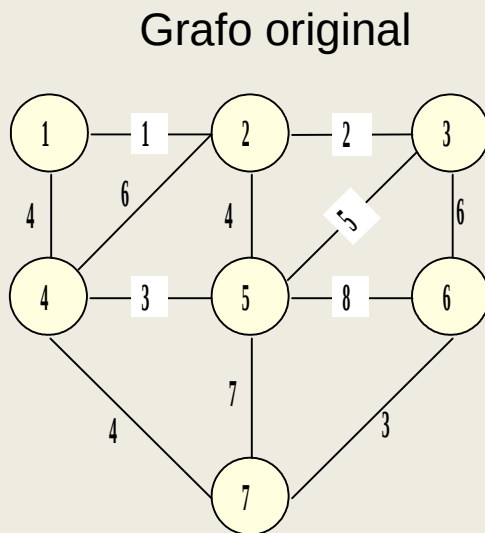
Hasta que número de aristas en T sea igual a $N-1$

Devolver T

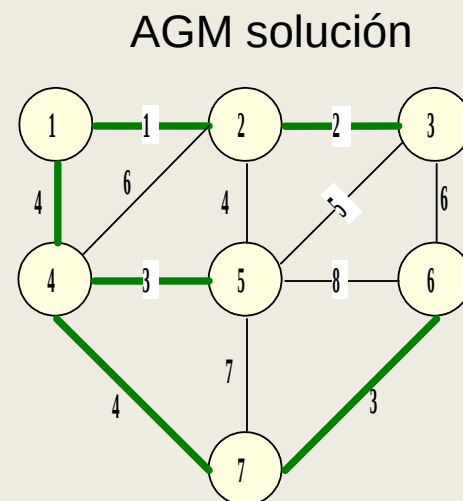
Eficiencia del algoritmo de Kruskal: **$O(|A| \cdot \log(|V|))$**

Algoritmo de Kruskal: Ejemplo de funcionamiento

Selección de aristas



(1,2)
 (2,3)
 (4,5)
 (6,7)
 (1,4)
 (2,5) X
 (4,7)



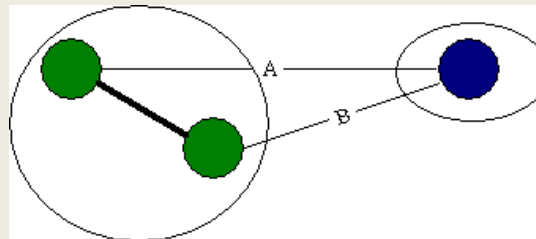
Coste de la solución: 17

Algoritmo de Kruskal: Optimalidad

Demostraremos la optimalidad por **reducción al absurdo**:

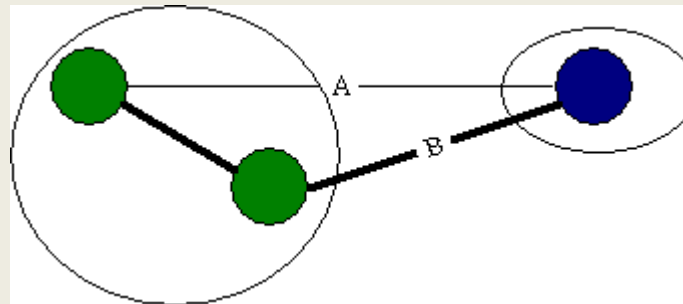
En un caso trivial donde existe un grafo completo compuesto por 2 nodos, la única arista que une ambos nodos es elegida por el algoritmo, dando la solución óptima.

En un caso base de dos subgrafos donde uno contiene 2 nodos unidos por una arista previamente escogida, se deberá escoger otra arista (**A** o **B**) que una los dos subgrafos:

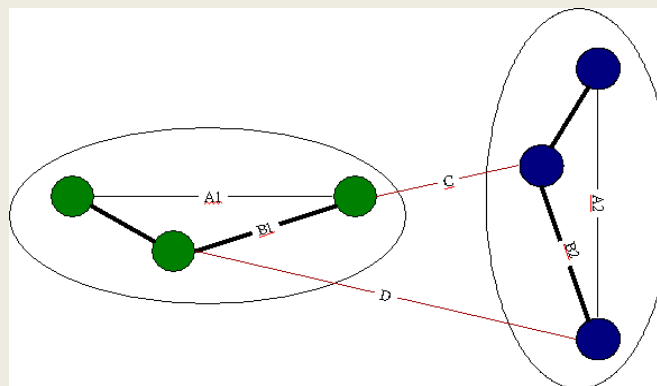


Algoritmo de Kruskal: Optimalidad

El algoritmo escogerá la arista de menor peso (**B**), y se comprueba que la solución es óptima.

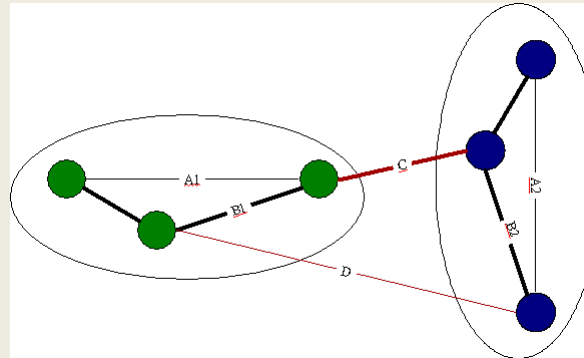


Supongamos que el algoritmo sigue ejecutándose, y que en un momento tenemos dos subgrafos que se deben unir para formar el AGM, y que hay 2 potenciales aristas para unirlos, **C** y **D**:



Algoritmo de Kruskal: Optimalidad

El algoritmo escogerá la arista de menor peso (**C**):



¿Es esta solución óptima? En el caso de que no lo fuera, debería existir otra arista (A1, A2 o D) que hiciese que la suma de los pesos de las aristas seleccionadas fuese menor que la suma actual.

Cualquiera que fuese esta arista, tendría un peso menor que **C**, y el algoritmo la habría seleccionado con antelación.

Como esto no ha sido así, entonces esa arista no existe, y **C** es la arista de menor peso que hace que la suma sea mínima.

Algoritmo de Prim: Idea general







La solución, al igual que en Kruskal, es un conjunto de aristas.

Sin embargo, la idea general de Prim no es seleccionar aristas, sino nodos.

Inicialmente, la solución será un conjunto de aristas vacío y tendremos un nodo inicial como candidato utilizado.

En cada paso, iremos construyendo la solución final seleccionando un nodo para unirlo con los que ya tenemos seleccionados, bajo el criterio de que la arista que une ese nodo con alguno de los ya seleccionados tenga coste mínimo.

Algoritmo de Prim: Diseño de componentes Greedy

-  **Lista de candidatos:** Los nodos del grafo original.
-  **Lista de candidatos utilizados:** Los nodos usados para obtener las aristas que forman la solución.
-  **Función solución:** El número de nodos seleccionados es igual al número de nodos del grafo.
-  **Función de selección:** Se selecciona el nodo no utilizado tal que la arista que une el nodo con uno ya utilizado tiene coste mínimo.
-  **Criterio de factibilidad:** No debe existir ciclos en los nodos seleccionados. Como siempre cogemos un nodo nuevo con la función de selección, el criterio de factibilidad siempre se cumple.
-  **F. objetivo:** Minimizar la suma de los pesos de las aristas que forman la solución.

Algoritmo de Prim: Diseño del algoritmo Greedy

ALGORITMO T=Prim(grafo $G=(V,A)$)

$B = \{ \text{elemento cualquiera de } V \}$ // Candidatos usados

$T = \emptyset$ // Solución a crear

Mientras $(|B| \neq |V|)$ **hacer:**

 Seleccionar un nodo v tal que existe una arista $a = (b, v)$,
de peso mínimo, que una un nodo b en B y otro nodo v en
 $V \setminus B$

$T = T \cup \{a\}$

$B = B \cup \{v\}$

Fin-Mientras

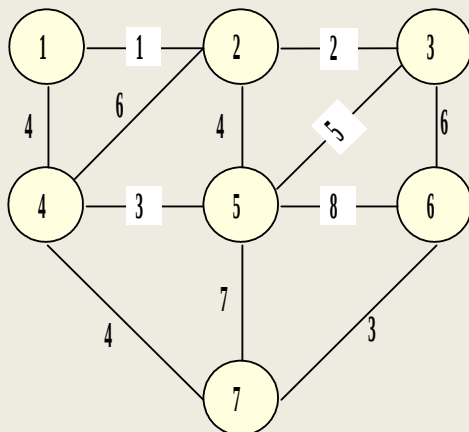
Devolver T

Eficiencia del algoritmo de Prim: $O(|V|^2)$

Algoritmo de Prim: Ejemplo de funcionamiento

Selección de nodos
(Nodo inicial= 1)

Grafo original



2-> (1,2)

3-> (2,3)

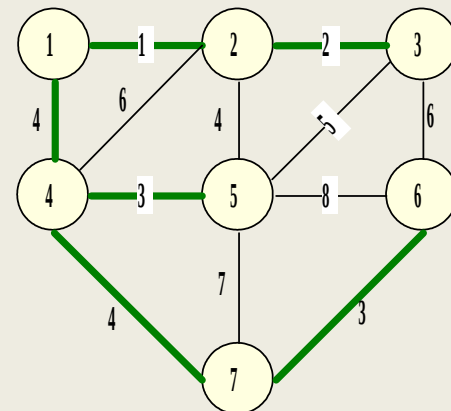
4-> (1,4)

5-> (4,5)

7-> (4,7)

6-> (7,6)

AGM solución



Coste de la solución: 17

Algoritmo de Prim: Optimalidad

El algoritmo de Prim también devuelve la solución óptima. Su demostración es análoga a Kruskal.





UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
- » 4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Enunciado del problema

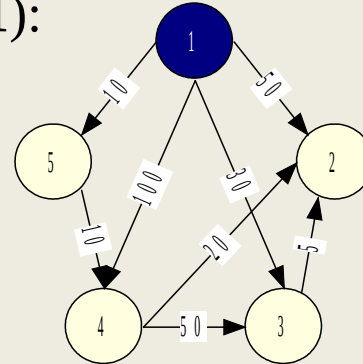
Sea $G=(V,A)$ un grafo **dirigido**, ponderado **con pesos no negativos**, con V el conjunto de vértices del grafo y A el conjunto de aristas del grafo. El problema consiste en

“Obtener un conjunto de secuencias de nodos/aristas que definan un camino mínimo entre un nodo origen y todos los demás nodos del grafo.”

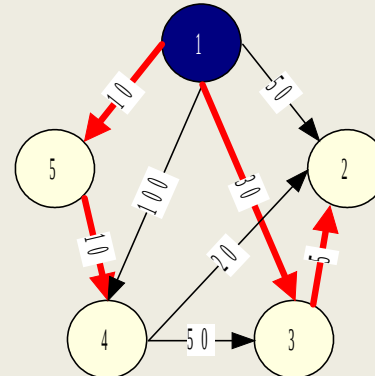


Entender el problema

Tenemos un grafo, en el que seleccionamos un nodo como **origen** S . (por ejemplo, el nodo $S=1$):

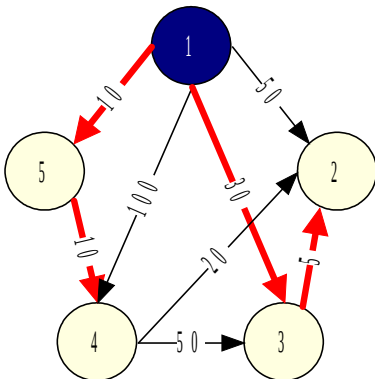


La idea que se persigue es encontrar los nodos por los que hay que pasar para ir desde el nodo origen hasta cualquier otro nodo del grafo, de modo que el recorrido tenga coste mínimo.



Algoritmo de Dijkstra: Idea general

- Supondremos que los nodos están numerados entre **0** y **n-1**.
- La solución serán dos vectores **P** y **D**.
 - **P[i]** contiene el elemento anterior por el que hay que pasar en el camino mínimo entre el nodo inicial **S** dado y el nodo **i**.
 - **D[i]** contiene la distancia existente para el camino mínimo entre el nodo inicial **S** dado y el nodo **i**.
- Existe una matrix **L**, donde cada componente **L[i][j]** indica el coste de viajar desde el nodo **i** al nodo **j** directamente (**matriz de adyacencia**).



En el ejemplo anterior:

- **Vector D** = {0, 35, 30, 20, 10}
- **Vector P** = {-1, 3, 1, 5, 1}

Algoritmo de Dijkstra: Idea general

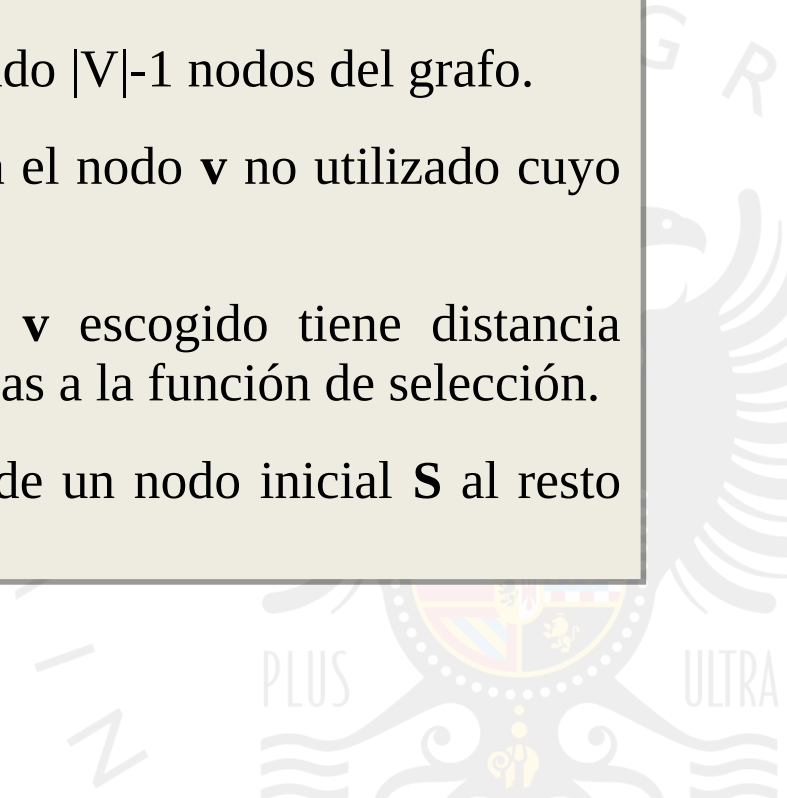
- Inicialmente, supondremos que la distancia mínima entre S y cualquier otro nodo es la del peso de la arista que une directamente S con ese nodo en la matriz de adyacencia:

$$D[i] = L[S, i]$$

- Seguidamente, para cada nodo i , iremos preguntándonos si es mejor ir directamente desde S hasta i , o pasando por otro nodo v . Ejemplo: “*Para ir de S a 4, el camino inicial que tengo es $L[1, 4]$. ¿Es mejor ese camino, o pasar antes por 5?*”
- Nos iremos haciendo esa pregunta para cada nodo i y cada nodo intermedio v por el que se pueda pasar, hasta que hayamos agotado todas las posibilidades.
- Iremos actualizando el camino mínimo en cada paso.

Algoritmo de Dijkstra: Diseño de componentes Greedy

- **Lista de candidatos:** Los nodos del grafo original.
- **Lista de candidatos utilizados:** Los nodos usados para obtener las aristas que forman la solución.
- **Función solución:** Se han seleccionado $|V|-1$ nodos del grafo.
- **Función de selección:** Se selecciona el nodo v no utilizado cuyo valor $D[v]$ sea mínimo.
- **Criterio de factibilidad:** El nodo v escogido tiene distancia mínima a S . Siempre se cumple gracias a la función de selección.
- **F. objetivo:** Minimizar la distancia de un nodo inicial S al resto de nodos del grafo.



Algoritmo de Dijkstra: Diseño del algoritmo Greedy

ALGORITMO [D,P]= **Dijkstra**($G=(V,A)$, L, S)

$C = V \setminus \{S\}$ // **Candidatos no utilizados**

Para $i=1$ **hasta** n , **hacer**: $D[i] = L[S][i]$; $P[i] = S$; // **Inicializar D y P**

Repetir $n-2$ **veces**:

v = elemento de C tal que $D[v]$ es mínimo

$C = C \setminus \{v\}$

Para cada w en C , **hacer**: // **Comprobar camino para los demás nodos**

Si $D[w] > D[v] + L[v][w]$, **entonces**: // **Es mejor pasar por v**
 $D[w] = D[v] + L[v][w]$; $P[w] \leftarrow v$;

Fin-Si

Fin-Para

Fin-Repetir

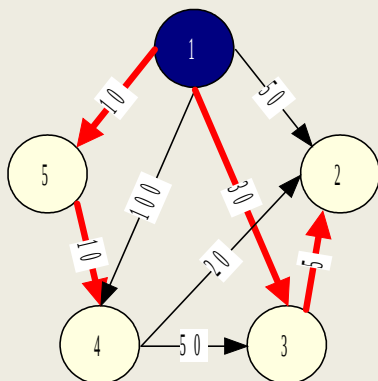
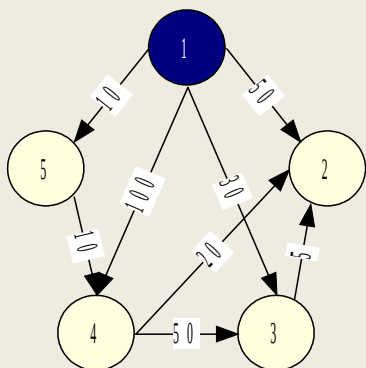
Devolver D, P

Eficiencia del algoritmo de Dijkstra:

- $O(|V|^2)$ para grafos densos
- $O(|A| \cdot \log(|V|))$ para grafos densos

Algoritmo de Dijkstra: Ejemplo de funcionamiento

Ejemplo: $S=1$



Paso	v	C	D
0	-	2, 3, 4, 5	0, 50, 30, 100, 10
i=1	5	2, 3, 4	0, 50, 30, 20, 10
i=2	4	2, 3	0, 40, 30, 20, 10
i=3	3	2	0, 35, 30, 20, 10

Coste de la solución: $0+35+30+20+10= 95$

Algoritmo de Dijkstra: Optimalidad

Hay que demostrar que:

1. Si un nodo **i** **no está en C** , entonces **$D[i]$** es óptimo.
2. Si un nodo **i** **está en C** , entonces **$D[i]$** es óptimo considerando sólo los nodos intermedios entre S e i que no están en C

Demostración. Caso base

Inicialmente, $C = V \setminus \{S\}$, por lo que se cumple que el camino óptimo de S a cualquier otro nodo, sin pasar por nodos intermedios, es óptimo dado que se deriva directamente de la matriz de adyacencia.

Paso de demostración para el caso (1)

Cuando se selecciona un nodo v , tiene $D[v]$ **mínimo**. No existirá otro nodo posterior $v2$ a seleccionar que haga que $D[v]$ se **reduzca**.

Si existiese tal nodo $v2$, se hubiera seleccionado $v2$ antes que v por el criterio de selección, por lo que deducimos que no puede existir un nodo $v2$ con tales características.

Por tanto, todos los nodos en $V \setminus C$ tienen distancia D óptima porque no puede existir el nodo $v2$, seleccionado posteriormente, que haga que un nodo v en $V \setminus C$ reduzca su distancia mínima $D[v]$.

UNIVERSITY



PLUS

ULTRA

Paso de demostración para el caso (2)

En el momento de seleccionar un nodo v , pasa al conjunto $V \setminus C$ (candidatos utilizados). Todos los demás nodos w en C actualizan su distancia mínima $D[w]$ si es necesario, a

$$D[w] = \min\{D[w], D[v] + L[v][w]\}.$$

$D[w]$ es optimal según la hipótesis del caso 2. Si no lo fuese, entonces existiría otro nodo x en $V \setminus C$, visitado antes que v , que cumpliese:

$$D[x] + L[x][w] \leq \min\{D[w], D[v] + L[v][w]\}$$

haciendo que la distancia optimal $D[w]$ sea inferior a la que el algoritmo calcula.

Este nodo x no puede existir, pues al ser añadido anteriormente, el algoritmo ya hizo la comprobación del mínimo y $D[w]$ se debería haber actualizado entonces, de modo que:

$$D[w] = D[x] + L[x][w] \leq D[v] + L[v][w]$$

Paso de demostración para el caso (2)

Como no puede existir ese nodo anteriormente escogido x , entonces $D[w]$ debe ser óptimo bajo la hipótesis del caso 2.

Al finalizar el algoritmo todos los nodos (menos uno, llamémosle w) están en $V \setminus C$.

Por tanto, todos los nodos en $V \setminus C$ tienen distancia $D[v]$ óptima, y también $D[w]$, pues no existe ningún otro nodo en C que pueda hacer reducir su distancia.

E
>
-
Z



PLUS

ULTRA



UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
- » 5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas

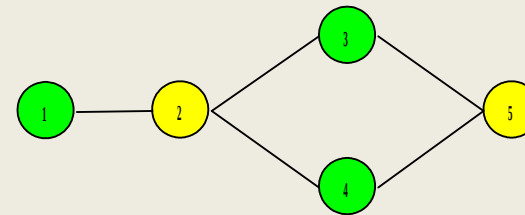
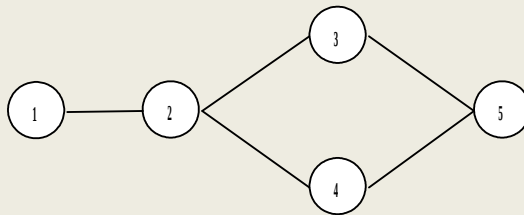


DECSAI

Enunciado del problema

Sea $G = \langle V, A \rangle$ un grafo no dirigido. Se desea asignar un color a cada nodo del grafo de modo que:

- No haya dos nodos adyacentes con el mismo color.
- Se utilicen el mínimo número de colores posible.



Complejidad del problema del coloreo de un grafo

- **Clase de problemas P:** Engloba a todos los problemas que son resolubles en un tiempo inferior a orden polinómico $O(n^k)$
- **Clase de problemas NP:** Engloba a todos los problemas cuya solución es fácil de verificar, pero donde el cálculo de la misma conlleva una complejidad muy grande debido a la gran cantidad de potenciales soluciones (óptimas y no óptimas) a explorar.
- El coloreo de un grafo es **NP**, por lo que no se conoce una solución eficiente al mismo.

Heurísticas

- Al no conocerse soluciones óptimas eficientes a los problemas **NP**, si la eficiencia es un requisito en la implementación nos conformamos con soluciones “**suficientemente buenas**” o “**las mejores que podemos encontrar**”.

¿Qué es una heurística?

- Las Heurísticas son criterios, métodos, o principios para decidir cual, entre una serie de cauces alternativos de acción, promete ser más efectivo a la hora de lograr alguna meta.

Esto representa un compromiso entre dos exigencias:

- Las heurísticas deben ser computacionalmente simples.
- Deben discriminar correctamente entre buenas y malas opciones.

E
>
-
Z



Algoritmo del coloreo de un grafo: Idea general

- Inicialmente, supondremos que ningún nodo del grafo estará coloreado.
- Seleccionaremos un nodo del grafo al azar, y lo pintaremos de un color.
- Escogeremos todos los nodos no adyacentes del nodo seleccionado que aún no estén coloreados, y los pintaremos del mismo color.

Repetiremos el procedimiento anterior hasta completar todos los nodos del grafo.

E
>
-
Z



Algoritmo del coloreo de un grafo: Diseño Greedy

- **Lista de candidatos:** Los nodos del grafo.
- **Lista de candidatos utilizados:** Los nodos ya coloreados.
- **Función solución:** Todos los nodos del grafo están coloreados.
- **Función de selección:** Seleccionar un nodo al azar.
- **Criterio de factibilidad:** El nodo seleccionado no puede estar coloreado. Siempre se cumple, gracias a la gestión de la lista de candidatos.
- **F. objetivo:** Minimizar el número de colores usado para colorear el grafo.



Algoritmo del coloreo de un grafo: Diseño del algoritmo

ALGORITMO T= Coloreo($G=<V,A>$)

$C=V$ // Conjunto de candidatos

$T=\{0\}_n$ // Vector de n colores asignados al grafo, sin valor

$K= 1$; // Color actual

Mientras ($|C| > 0$) **hacer:**

 Seleccionar i = Nodo cualquiera de C

$C= C\setminus\{i\}$

$T[i]= K$ // Asignar color K al nodo i

Para cada nodo j en C no adyacente a i , **si es factible,**
hacer:

$T[j]= K,$

$C= C\setminus\{j\}$

Fin-Para

$K= K+1$

Fin-Mientras

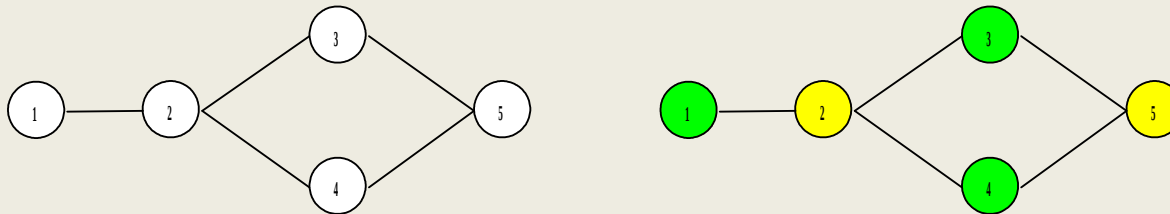
Devolver T



Algoritmo del coloreo de un grafo: Ejemplo

Paso 1: Seleccionar nodo 3 (al azar) y rellenar también 1 y 4

Paso 2: Seleccionar nodo 5 (al azar) y rellenar también 2



Coste de la solución: 2 colores utilizados



UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
- » 6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Enunciado del problema

- Se dispone de un conjunto de **n** objetos, indexados de **1...n**.
- Para cada objeto **i**, se conoce su beneficio **b_i** y su peso **w_i** (números reales positivos).
- Se dispone de un contenedor (mochila) con una capacidad para almacenar un peso máximo de **M**.
- Se desea encontrar un conjunto de valores **X = {x₁, x₂, ..., x_n}**, donde **x_i** es la cantidad a llevar para el objeto **i**, tal que se maximice la siguiente expresión:

$$\max_{x_i} \left\{ \sum_{i=1}^n b_i x_i \right\}$$

Sujeto a la restricción: $\sum_{i=1}^n w_i x_i \leq M$

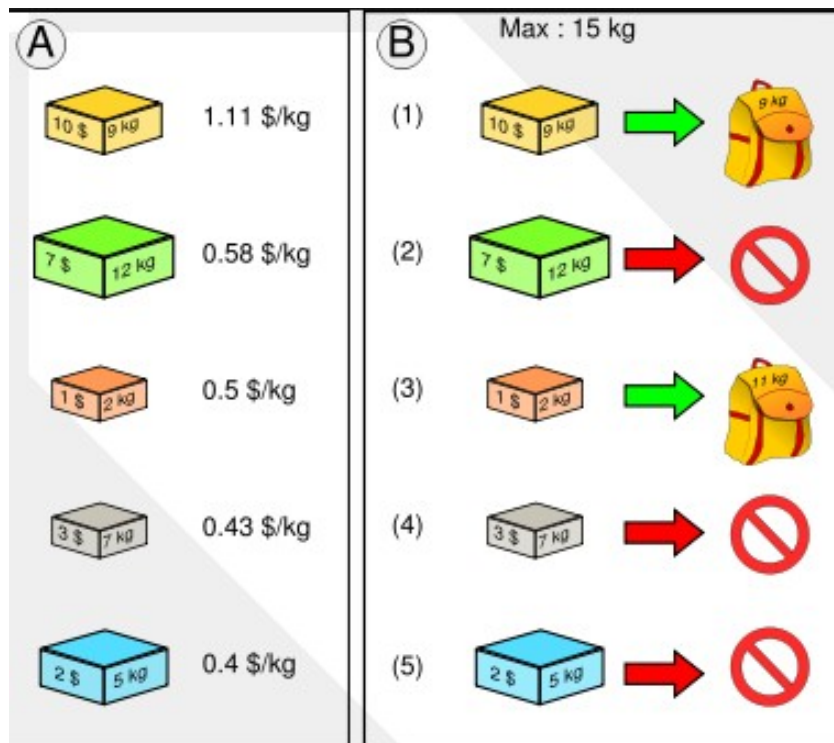
Problemas de la mochila

Tres tipos:

■ **Mochila continuo:** Cada objeto que podemos llevar puede fraccionarse ($1/2$, $1/4$, 0.7 del objeto, etc.)

■ **Mochila 0/1:** Los objetos son únicos y no se fraccionan. Nos llevamos todo el objeto o nada.

■ **Mochila discreto:** Los objetos no se fraccionan, pero podemos llevarnos más de uno del mismo tipo (2 de A, 1 de B, 3 de C, etc.)



Problema de la mochila **continuo**

■ Supondremos que las cantidades que podemos llevar de cada objeto es un número real, normalizado al intervalo $[0, 1]$ ($x_i = 0$ no llevamos nada del objeto i , $x_i = 1$ nos llevamos el objeto i al completo, $x_i = r$ entre 0 y 1 nos llevamos el $r\%$ de i).

Posibles heurísticas para el problema de la mochila continuo

- Escoger máxima cantidad del objeto de máximo beneficio.
- Escoger máxima cantidad del objeto de menor peso.
- Escoger máxima cantidad del objeto i tal que $i = \max_i \{b_i / w_i\}$.

Algoritmo de la mochila continuo: Idea general

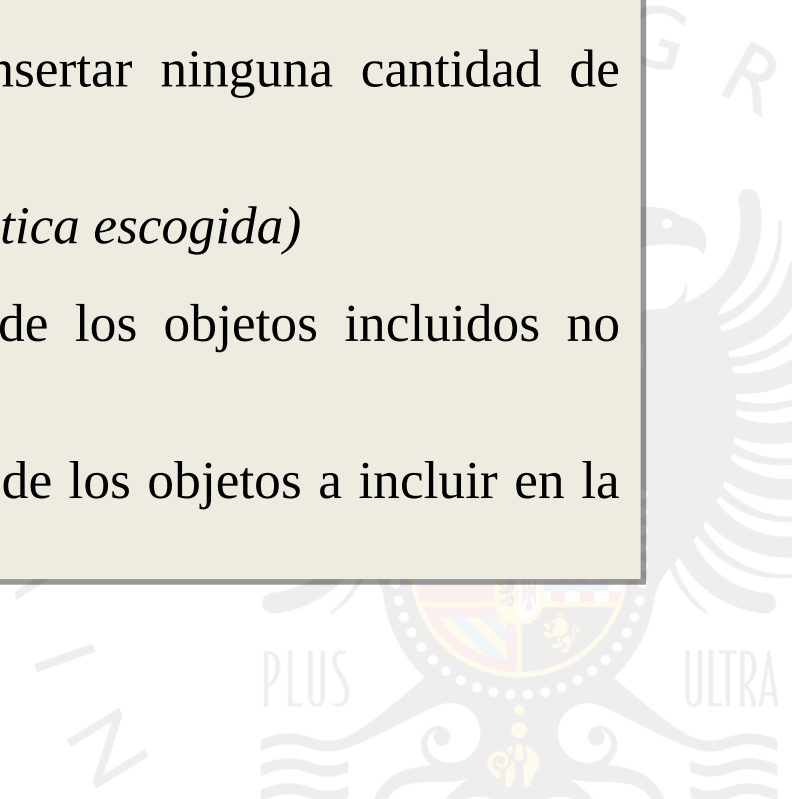
- Inicialmente, supondremos que la mochila está vacía y que no estamos llevando ningún objeto.
- En cada paso, iremos seleccionando un objeto a añadir entre los existentes, según el criterio que se desee escoger entre las 3 heurísticas explicadas.
- Insertaremos la máxima cantidad posible de ese elemento.

Repetiremos este procedimiento hasta que no queden objetos por añadir o hasta que la capacidad de la mochila esté completa.



Algoritmo de la mochila continuo: Diseño Greedy

- **Lista de candidatos:** Los objetos a llevar.
- **Lista de candidatos utilizados:** Los objetos incluidos ya en la mochila o descartados.
- **Función solución:** No se puede insertar ninguna cantidad de ningún objeto más en la mochila.
- **Función de selección:** *(Según heurística escogida)*
- **Criterio de factibilidad:** El peso de los objetos incluidos no supera la capacidad de la mochila.
- **F. objetivo:** Maximizar el beneficio de los objetos a incluir en la mochila, sin superar su capacidad.



Algoritmo de la mochila continuo: Diseño del algoritmo

ALGORITMO T= MochilaContinuo(M, B[0..n-1], W[0..n-1])

$C = \{0..n-1\}$ // Conjunto de objetos posibles

$T = \{0\}_n$ // Vector de n cantidades a llevar a 0. *Solución a crear*

Mientras $\text{suma}(T) < M$ y queden candidatos en C , **hacer:**

 Seleccionar i = mejor objeto restante en C

$C = C \setminus \{i\}$

Si $\text{suma}(T) + w_i \leq M$ **entonces**

$T_i = 1,$

en otro caso

$T_i = (M - \text{suma}(\text{peso de llevar las cantidades de } T)) / w_i$

Fin-Mientras

Devolver T

Eficiencia del algoritmo: **$O(n)$**

Algoritmo de la mochila continuo: Ejemplo.

Ejemplo: Probemos con las 3 heurísticas en el problema siguiente, con $n=5$ y $M=100$

	1	2	3	4	5
W	10	20	30	40	50
B	20	30	66	40	60
B/W	2.0	1.5	2.2	1.0	1.2

Soluciones:

Heurística	x_1	x_2	x_3	x_4	x_5	Beneficio
Máx b_i	0	0	1	0.5	1	146
Mín w_i	1	1	1	1	0	156
Máx v_i/w_i	1	1	1	0	0.8	164

La estrategia de Máx v_i/w_i devuelve el óptimo para el caso continuo

Algoritmo de la mochila continuo: Optimalidad

Supongamos que tenemos los objetos ordenados de mejor a peor beneficio/peso:

$$b_1/w_1 \geq b_2/w_2 \geq \dots \geq b_n/w_n$$

Y que las cantidades de objetos devueltas por el algoritmo son:

$$X = (x_1, x_2, \dots, x_n)$$

Si todos los $x_i=1$, entonces la solución es óptima trivial.

En caso contrario, sea j el primer índice tal que $x_j < 1$. Todos los índices $k > j$ serán por tanto $x_k = 0$, y que $\sum_{i=1}^n w_i x_i = M$, con beneficio $B(X)$ para la solución X .

Algoritmo de la mochila continuo: Optimalidad

Ahora supongamos otra solución factible $Y = (y_1, y_2, \dots, y_n)$, con beneficio $B(Y)$.

Si calculamos la diferencia de los beneficios entre X e Y :

$$\begin{aligned} B(X) - B(Y) &= \sum_{i=1}^n b_i x_i - \sum_{i=1}^n b_i y_i = \sum_{i=1}^n b_i (x_i - y_i) \\ &= \sum_{i=1}^n w_i \frac{b_i}{w_i} (x_i - y_i) \end{aligned}$$

Cuando $i < j$, $x_i = 1$ y la diferencia de todos los $(x_i - y_i) \geq 0$

Cuando $i > j$, $x_i = 0$ y la diferencia de todos los $(x_i - y_i) \leq 0$

Por tanto, **siempre se cumpliría** $(x_i - y_i) \frac{b_i}{w_i} \geq (x_i - y_i) \frac{b_j}{w_j}$ y podemos reescribir la diferencia como:

$$B(X) - B(Y) \geq \frac{b_j}{w_j} \sum_{i=1}^n w_i (x_i - y_i) \geq 0$$

Por tanto, el beneficio de la solución de X es máximo global.



UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
- » 7. El viajante de comercio
8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Enunciado del problema

Sea un grafo $G=(V,A)$, **no dirigido**, **completo**, con V vértices y A aristas, donde las **aristas están ponderadas con pesos no negativos**.

El problema del viajante de comercio consiste en:

“Encontrar el circuito hamiltoniano minimal del grafo G .”

Volvamos antes al repaso de grafos...



Repaso de conceptos sobre grafos

- **Grafo completo:** Aquel en el que, para cada par de vértices v_i y v_j , existe la arista (v_i, v_j) .
- **Circuito:** Camino no vacío donde el nodo inicial y el final es el mismo.
- **Circuito hamiltoniano:** Circuito que pasa por todos los nodos del grafo, sin repetir ningún nodo.
- **Circuito hamiltoniano minimal:** Circuito hamiltoniano con la propiedad de que no existe ningún otro circuito hamiltoniano sobre el mismo grafo cuya suma del peso de sus aristas sea inferior al circuito considerado.

¡OJO! No confundir ciclo y circuito

Enunciado del problema

Sea un grafo $G=(V,A)$, **no dirigido**, **completo**, con V vértices y A aristas, donde las **aristas están ponderadas con pesos no negativos**.

El problema del viajante de comercio consiste en:

“Encontrar el circuito hamiltoniano minimal del grafo G .”

Ejemplo de problema tipo del viajante de comercio

Un repartidor de periódicos sale todas las mañanas de su almacén, y debe repartir la mercancía por todos los kioskos de la ciudad. Diseñar una ruta que le permita repartir los periódicos en todos los kioskos y terminar en el almacén de donde partió, sin repetir ningún kiosko y haciendo que la ruta completa tenga un coste mínimo.

Posibles heurísticas para el problema de la mochila continuo

- Seleccionar la arista de menor coste que haga que ningún nodo tenga asociadas más de dos aristas
 - Seleccionar el siguiente nodo más cercano a visitar
 - ... etc.
- **Aún no existe ninguna solución óptima de orden polinomial al problema del viajante de comercio.**

El viajante de comercio: Idea general

- Supondremos que se parte de un nodo cualquiera del grafo.
- A continuación, se seleccionará el nodo adyacente no visitado más cercano, e insertaremos la arista que lo une en la solución.
- Al finalizar, uniremos la arista del último nodo seleccionado con el primero, para cerrar el circuito.

Algoritmo del viajante de comercio: Diseño Greedy

- **Lista de candidatos:** Los nodos del grafo original.
- **Lista de candidatos utilizados:** Los nodos seleccionados previamente.
- **Función solución:** El número de nodos en la solución es $|V|$
- **Función de selección:** Se selecciona un nodo v tal que la arista a que lo une con el último nodo visitado tiene coste mínimo.
- **Criterio de factibilidad:** No se pueden formar ciclos. Siempre se cumple, dado que siempre escogemos un nodo nuevo.
- **F. objetivo:** Encontrar un ciclo hamiltoniano minimal del grafo.



Algoritmo del viajante de comercio: Diseño del algoritmo

ALGORITMO TravelingSalesmanProblem($G=(V,A)$)

s = Seleccionar un nodo cualquiera de V

$C = V \setminus \{s\}$ // Candidatos no utilizados

$T = \{s\}$

Repetir hasta que $|C|=0$:

v = Seleccionar nodo en C donde $a=(s, v)$ tiene peso mínimo

$C = C \setminus \{v\}$

$T = T \cup \{(s,v)\}$

Actualizar $s = v$

Fin-Repetir

a = Arista restante que queda para cerrar el circuito

$T = T \cup \{a\}$

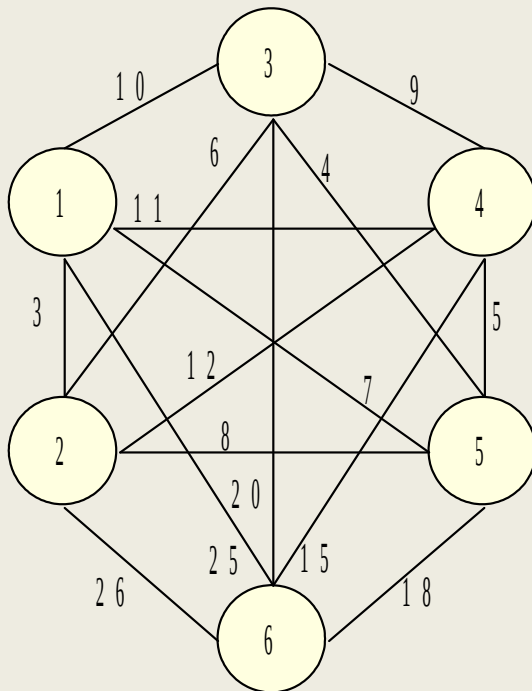
Devolver T

Eficiencia del algoritmo: **$O(n)$**

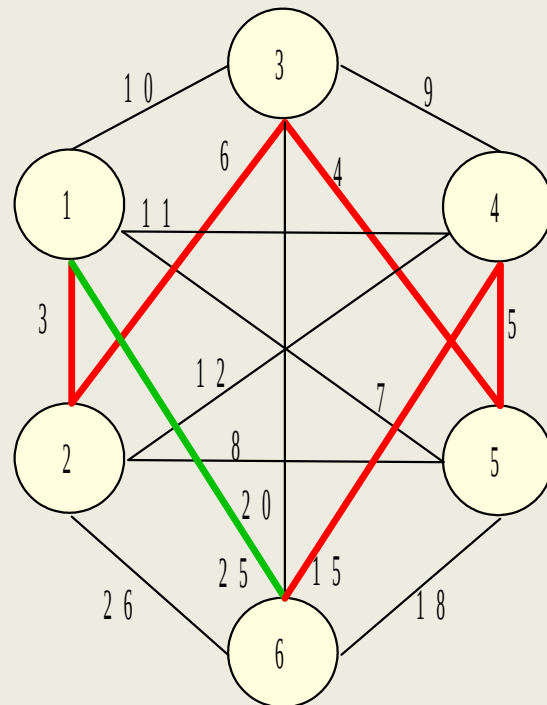
Algoritmo del viajante de comercio: Ejemplo.

Nodo inicial $s = 1$

Inserción de aristas: (1,2), (3,5), (4,5), (2,3),
(1,5) X, (4,6). Finalmente se añade (1,6)



Solución:
(Coste 58)





UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
- » 8. Planificación de tareas en el sistema
9. Asignación de tareas



DECSAI

Aclaración

Existen múltiples alternativas/enunciados que puedan considerarse un problema de planificación de tareas. En este tema plantearemos uno de estos problemas: la **minimización del tiempo que cada tarea espera hasta haberse finalizado**.

Enunciado del problema

Sean **n** tareas pendientes por ejecutar.

Sabiendo que cada tarea **i** tarda en ejecutarse un tiempo t_i , se desea encontrar el **orden de ejecución** de dichas tareas para minimizar el tiempo total **T** que **todas las tareas esperan** antes de finalizar su ejecución. Es decir, asumiendo que en la posición **j** se ejecuta en la **tarea p(j)**, minimizar:

$$T = \sum_{i=1}^n \left(t_i + \sum_{j=1}^{i-1} t_{p(j)} \right)$$

Ejemplo de planificación de tareas en el sistema

Supongamos que tenemos 3 tareas pendientes, cuyo tiempo de ejecución es $t_1=5$, $t_2=10$, $t_3=3$.

- Orden $P=(1, 2, 3)= 5+(5+10)+(5+10+3)= 38$
- Orden $P=(1, 3, 2)= 5+(5+3)+(5+3+10)= 31$
- ...
- Orden $P=(3, 1, 2)= 3+(3+5)+(3+5+10)= 29$ **óptimo**

Planificación de tareas en el sistema: Idea general

- En este problema, la mejor planificación se obtiene cuando **las tareas se ejecutan en orden ascendente de tiempo de ejecución**.
- Es decir: Ejecutar primero la más corta; a continuación, la siguiente más corta, etc.

Algoritmo de planificación de tareas: Diseño Greedy

- **Lista de candidatos:** Las tareas a ejecutar.
- **Lista de candidatos utilizados:** Las tareas ya planificadas.
- **Función solución:** El número de tareas planificadas es **n**.
- **Función de selección:** Se selecciona una tarea **i** tal que su tiempo de ejecución **t_i** es mínimo.
- **Criterio de factibilidad:** La tarea **i** debe tener un tiempo de ejecución **t_i** mínimo. Siempre se cumple, gracias a la función de selección.
- **F. objetivo:** Minimizar $T = \sum_{i=1}^n (t_i + \sum_{j=1}^{i-1} t_{p(j)})$, donde **p(j)** es la tarea que se ejecuta en la **j**-ésima posición.

Algoritmo de planificación de tareas: Algoritmo Greedy

ALGORITMO P= PlanificacionTareas(vector t, n)

$C = \{1..n\}$ // Candidatos, tareas a ejecutar

$P = \{0\}_n$ // Vector de tareas planificadas, solución a devolver

Para i desde 1 hasta n , **hacer:**

j = Seleccionar tarea j en C donde $t[j]$ sea mínimo

$C = C \setminus \{j\}$

$P(i) = j$

Fin-Para

Devolver P

Eficiencia del algoritmo: **$O(n)$**



Planificación de tareas en el sistema: Optimalidad

Sea $P=(p_1, p_2, \dots, p_n)$ una permutación de las tareas, ordenadas por orden creciente de t_i , y sea $s_i=t_{p(i)}$. Por tanto, el tiempo total de atención para la permutación P , $T(P)$, se calcula como:

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

Supongamos ahora una nueva permutación P' resultante de intercambiar dos posiciones **a** y **b** de la permutación P , de modo que $p'_a = p_b$ y $p'_b = p_a$. En este caso:

$$T(P') = (n - a + 1)s_b + (n - b + 1)s_a + \sum_{\substack{k=1 \\ k \neq a, b}}^n (n - k + 1)s_k$$

Planificación de tareas en el sistema: Optimalidad

Si comparamos los tiempos $T(P)$ con $T(P')$:

$$\begin{aligned}
 & T(P') - T(P) \\
 &= (n - a + 1)s_b + (n - b + 1)s_a + \sum_{\substack{k=1 \\ k \neq a, b}}^n (n - k + 1)s_k \\
 &\quad - \sum_{k=1}^n (n - k + 1)s_k \\
 &= (n - a + 1)s_b + (n - b + 1)s_a \\
 &\quad - ((n - a + 1)s_a + (n - b + 1)s_b) \\
 &= (n - a + 1)(s_b - s_a) + (b - n + 1)(s_a - s_b) \geq 0
 \end{aligned}$$

Cualquier cambio que haga que una tarea de mayor tiempo preceda a una de menor tiempo hace que $T(P')$ sea mayor o igual que $T(P)$. Por tanto, la decisión óptima es la tomada por el algoritmo, seleccionando siempre la tarea de menor tiempo t_i .



UNIVERSIDAD
DE GRANADA

Algorítmica

Grado en Ingeniería Informática

Algoritmos Voraces

1. Introducción
2. Diseño de algoritmos voraces
3. El árbol generador minimal
4. Caminos mínimos
5. El coloreo de un grafo
6. El problema de la mochila
7. El viajante de comercio
8. Planificación de tareas en el sistema
- » 9. Asignación de tareas



DECSAI

Aclaración

Al igual que en el problema de planificación de tareas, existen múltiples alternativas/enunciados que puedan considerarse un problema de asignación.. En este tema plantearemos uno de estos problemas: la **minimización del coste de asignar n trabajadores a n tareas**.

UNIVERSITY



Enunciado del problema

Sea n un número entero positivo, $T=\{T1..Tn\}$ un conjunto de n **tareas** y $P=\{P1..Pn\}$ un conjunto de n **trabajadores**. Denotaremos $X=[x_{ij}]_{n,n}$ a una matriz que indica la asignación entre trabajadores y tareas. Diremos que un trabajador i tiene asignada la tarea j si $x_{ij}=1$, y que no la tiene asignada si $x_{ij}=0$.

Una asignación X sólo será válida si una tarea no tiene más de un trabajador asignado y un trabajador no participa en más de una tarea (las filas/columnas de X sólo tienen una componente a 1).

Denotaremos también a $C=[c_{ij}]_{n,n}$ a la matriz de costes, donde c_{ij} es el coste de asignar al trabajador i para realizar la tarea j .

En el problema, se desea minimizar el coste de asignar los n trabajadores a las n tareas; es decir:

$$\min_{x_{ij}} \left\{ \sum_{i=1}^n \sum_{j=1}^n b_{ij} x_{ij} \right\} \quad \text{sujeto a} \quad \sum_{j=1}^n b_{ij} x_{ij} = 1 \quad \forall i, \quad \sum_{i=1}^n b_{ij} x_{ij} = 1 \quad \forall j$$

Heurísticas

Hay 2 heurísticas simples para el problema:

1. Asignar cada tarea al mejor trabajador posible
2. Asignar cada trabajador a la mejor tarea posible

Ninguna de las dos es óptima.

E
>
-
Z



Heurística 1

■ Matriz de costes C

T1-> P3;

T2-> P1;

T3-> P2;

T4->P4;

	T1	T2	T3	T4
P1	11	12	18	40
P2	14	15	13	22
P3	11	17	19	23
P4	17	14	20	28

■ Coste= 11+12+13+28= 64

■ La mejor solución es:

T1->P1, T2->P3, T3->P2, T4->P4, con coste= 61

>
-
Z

PLUS

ULTRA

Heurística 2

Matriz de costes C

P1- > T1;

P2- > T3;

P3- > T2;

P4- > T4;

	T1	T2	T3	T4
P1	11	12	18	40
P2	14	15	13	22
P3	11	17	19	23
P4	17	14	20	28

Coste= 11+13+17+28= 69

La mejor solución es:

T1->P1, T2->P3, T3->P2, T4->P4, con coste= 61

Algoritmo del asignación de tareas: Diseño Greedy

- **Lista de candidatos:** Los trabajadores (si usamos heurística 1) o las tareas (si usamos heurística 2).
- **Lista de candidatos utilizados:** Los trabajadores (si usamos heurística 1) o las tareas (si usamos heurística 2) que ya han sido asignados.
- **Función solución:** Todos los trabajadores/tareas han sido asignados.
- **Función de selección:** El trabajador de menor coste para cada tarea (si usamos heurística 1) o la tarea de menor coste para cada trabajador (si usamos la heurística 2).
- **Criterio de factibilidad:** Un trabajador/tarea sólo puede estar asignado una única vez.
- **F. objetivo:** Minimizar el coste total de asignar cada trabajador a cada tarea, es decir:

$$\min_{x_{ij}} \left\{ \sum_{i=1}^n \sum_{j=1}^n b_{ij} x_{ij} \right\}$$

Algoritmo de asignación de tareas: Diseño del algoritmo

ALGORITMO P= AsignaciónTareasHeuristica1(Matriz C, n)

$U = \{1..n\}$ // Candidatos, trabajadores a asignar

$X = \{0\}_n$ // Vector de trabajadores asignados, solución a devolver

Para i = desde 1 hasta n , **hacer:**

j = Seleccionar trabajador j en U donde $C[j][i]$ sea mínimo

$U = U \setminus \{j\}$

$X(j) = i$

Fin-Para

Devolver X

ALGORITMO P= AsignaciónTareasHeuristica2(Matriz C, n)

$U = \{1..n\}$ // Candidatos, tareas a asignar

$X = \{0\}_n$ // Vector de tareas asignadas, solución a devolver

Para i = desde 1 hasta n , **hacer:**

j = Seleccionar tarea j en U donde $C[i][j]$ sea mínimo

$U = U \setminus \{j\}$

$X(i) = j$

Fin-Para

Devolver X

Trabajo final del tema

Buscar en la literatura al menos 3 problemas de optimización combinatoria. Explicarlos en clase y cómo se resuelven mediante Greedy.





UNIVERSIDAD
DE GRANADA



Algorítmica

Grado en Ingeniería Informática

Tema 3 – Algoritmos Voraces (Greedy)

Este documento está protegido por la Ley de Propiedad Intelectual (Real Decreto Ley 1/1996 de 12 de abril). Queda expresamente prohibido su uso o distribución sin autorización del autor.

Manuel Pegalajar Cuéllar

manupc@ugr.es

Departamento de Ciencias de la
Computación e Inteligencia Artificial

<http://decsai.ugr.es>