FEBRERO 2015 – SEMANA 1 (MODELO A)

1. Considérese el vector [5, 2, 7, 3, 1, 8, 2, 6, 9]. Los vectores argumento de la primera invocación recursiva del algoritmo de quicksort, cuando se toma el elemento 5 de la primera posición como pivote, son:

(a) [2,3, 1,2] y [8,7,6,9]
(b) [2,7,3,1] y [8,2,6,9]
(c) [1,2,2,3] y [8,7,6,9]
(d) Ninguna de las opciones anteriores.

Al inicio de la ordenación se toma un elemento aproximadamente en la mitad del vector, por lo que, cuando se empieza a ordenar, hay que llegar hasta dicho vector ordenado respecto al punto de división o a la mitad del vector.

Se garantiza que los elementos a la izquierda de la mitad son los menores, y los situados a la derecha son los mayores. Se avanza el recorrido hasta que se cumpla que i > p y j <= p, y se detiene cuando i >= j.

Paso 1

P								
5	2	7	3	1	8	2	6	9
	i							j

No se cumple que i > p (2 < 5), así como tampoco que j <= p (9 > 5), por lo que se desplaza en una posición a la derecha e izquierda los índices “i” y “j”, respectivamente.

Paso 2

P								
5	2	7	3	1	8	2	6	9
		i					j	

En este caso se cumple que i > p (7 > 5), pero sigue sin cumplirse j <= p (6 > 5), por lo que tan solo se desplaza el índice “i” a la izquierda.

Paso 3

P								
5	2	7	3	1	8	2	6	9
		i				j		

Ahora si se cumple tanto que i > p (7 > 5), así como que j <= p (2 < 5), por lo que se intercambian los elementos en los índices “i” y “j” entre sí, y se desplazan los índices en una posición a la derecha e izquierda respectivamente.

Paso 4

P								
5	2	2	3	1	8	7	6	9
			i		j			

No se cumple i > p (3 < 5), así como tampoco que j <= p (8 > 5), por lo que se desplazan los índices “i” y “j” en una posición a la derecha e izquierda, respectivamente.

Paso 5

P				i				
5	2	2	3	1	8	7	6	9
				j				

Cuando sucede que tanto “i” como “j” comparten el mismo índice del vector, o la posición de “i” es superior a la de “j” (i >= j), el recorrido se detiene.

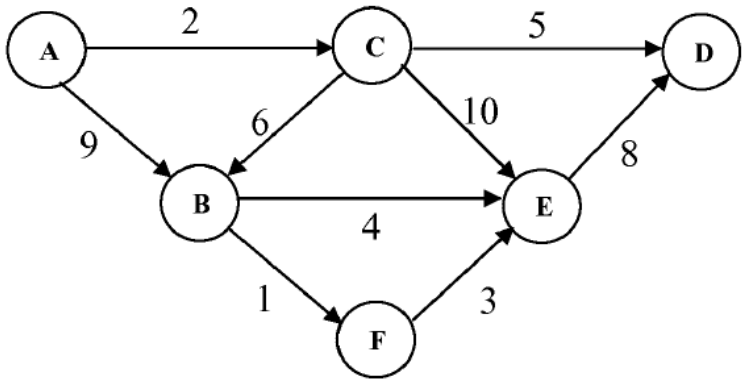
Justo en este punto, se intercambia el elemento en la posición “j” por el pivote “P”.

				P				
1	2	2	3	5	8	7	6	9

El vector resultante queda como: {1, 2, 2, 3} 5 {8, 7, 6, 9}

Respuesta C)

2. Dado el grafo de la siguiente figura:



indicad cuál sería el orden en que se seleccionarían (pasan a estar explorados) los nodos al aplicar el algoritmo de Dijkstra desde el nodo A:

(a) A, C, D, B, F, E
(b) A, F, C, E, B, D
(c) A, C, B, F, E, D
(d) Ninguna de las anteriores

Nodos que han salido	Nodos que no han salido	Vector Distancia especial [ ]					Predecesores				
		B	C	D	E	F	B	C	D	E	F
A	B, C, D, E, F	9	**2**	∞	∞	∞	A	A	A	A	A
A, C	B, D, E, F	8	2	**7**	12	∞	C	A	C	C	A
A, C, D	B, E, F	**8**	2	7	12	∞	C	A	C	C	A
A, C, D, B	E, F	8	2	7	12	**9**	C	A	C	C	B
A, C, D, B, F	E	8	2	7	12	9	C	A	C	C	B

El orden de selección de los nodos es: {A, C, D, B, F, E}

Respuesta A)

3. Se dispone de un conjunto A de n números enteros (tanto positivos como negativos) sin repeticiones almacenados en una lista. Dados dos valores enteros m y C, siendo m<n se desea resolver el problema de encontrar un subconjunto de A compuesto por exactamente m elementos y tal que la suma de los valores de esos m elementos sea C.

¿Cuál de los siguientes esquemas es más eficiente de los que puedan resolver el problema correctamente?

(a) Esquema voraz.
(b) Esquema de divide y vencerás.
(c) Esquema de vuelta atrás.
(d) Esquema de ramificación y poda.

Respuesta C)

4. Dadas las matrices: A_1 (3x5), A_2 (5x2) y A_3 (2x3) y A_4 (3x2) y siendo E(i,j) el número de operaciones mínimo para resolver la operación A_i x A_{i+1} x ... x A_j mediante programación dinámica, se pide indicar cuál de las siguientes opciones es cierta.

(a) E(2, 3) = 15
(b) E(1, 3) = 30
(c) E(2, 4) = 32
(d) E(2, 2) = 10

A(p x q) ; B(q x r) ; A x B → p x q x r

a) E(2, 3) = A_2 x A_3 = (5 x 2) x (2 x 3) = 5 x 2 x 3 = 30 → Falso
b) E(1, 3)
   - A_1 x (A_2 x A_3) = (5 x 2 x 3) + (3 x 5 x 3) = 30 + 45 = 75 → Falso
   - (A_1 x A_2) x A_3 = (3 x 5 x 2) + (3 x 2 x 3) = 30 + 18 = 48 → Falso
c) E(2, 4)
   - (A_2 x A_3) x A_4 = (5 x 2 x 3) + (5 x 3 x 2) = 30 + 30 = 60 → Falso
   - A_2 x (A_3 x A_4) = (2 x 3 x 2) + (5 x 2 x 2) = 12 + 20 = 32 → Cierto

d) No es posible resolver, ya que i = 2 y j = 2 (i = j), por tanto, E(2, 2) = 0 → Falso

Respuesta C)

5. Dado el problema de la devolución del cambio para una cantidad C>0. Indica cuál de estas afirmaciones es cierta.

(a) El esquema de ramificación y poda es el único que puede resolver de forma óptima el problema cuando se dispone de un conjunto finito de tipos de moneda T = {m^0, m^1, m^2, ... m^n}, siendo m>1 y n>0.

(b) Se puede encontrar una estrategia voraz que permita resolver el problema de forma óptima para cualquier conjunto de monedas.

(c) El esquema de programación dinámica puede resolver de forma óptima el problema cuando no se cumpla que T = {m^0, m^1, m^2, ... m^n}, siendo m>1 y n>0.

(d) El esquema de vuelta atrás es el más apropiado para resolver este problema cuando se dispone de un conjunto finito de tipos de moneda T = {m^0, m^1, m^2, ... m^n}, siendo m>1 y n>0.

Respuesta C)

6. ¿Cuál de las siguientes afirmaciones es <u>falsa</u> con respecto al coste de las funciones de manipulación de grafos?

(a) La función Etiqueta que devuelve la etiqueta o peso asociado a la arista que une dos vértices tiene un coste constante cuando el grafo se implementa con una matriz de adyacencia.

(b) La función BorrarArista es más costosa cuando el grafo se implementa mediante una lista de adyacencia.

(c) La operación Adyacente?, que comprueba si dos nodos son adyacentes, es más costosa cuando el grafo se implementa con una matriz de adyacencia.

(d) La función Adyacentes, que devuelve una lista con los vértices adyacentes a uno dado, es menos costosa cuando el grafo se implementa con una matriz de adyacencia.

Funciones Manipulación de Grafos	Matriz de Adyacencia	Lista de Adyacencia	Descripción
CrearGrafo	O(1)	O(1)	Devuelve un grafo vacío
AñadirArista	O(1)	O(1)	Añade una arista entre los vértices 'u' y 'v', y le asigna un peso 'p'
AñadirVertice	O(n)	O(1)	Añade el vértice 'v' al grafo 'g'
BorrarArista	O(1)	O(n)	Elimina la arista entre los vértices
BorrarVertice	O(n)	O(n + a)	Borra el vértice 'v' al grafo 'g' y todas las aristas que partan o lleguen a él
Adyacente? o esAdyacente	O(1)	O(n)	Comprueba si los vértices son adyacentes
Adyacentes	O(n)	O(1)	Devuelve una lista con los vértices adyacentes a 'v'
Etiqueta	O(1)	O(n)	Devuelve la etiqueta o peso asociado a la arista que une los vértices

***<u>Nota:</u>** Hubo una errata con las opciones dadas a esta pregunta, por lo que se aceptaron como respuestas correctas:*

Respuesta C) o D)

<u>PROBLEMA (4 Puntos)</u>

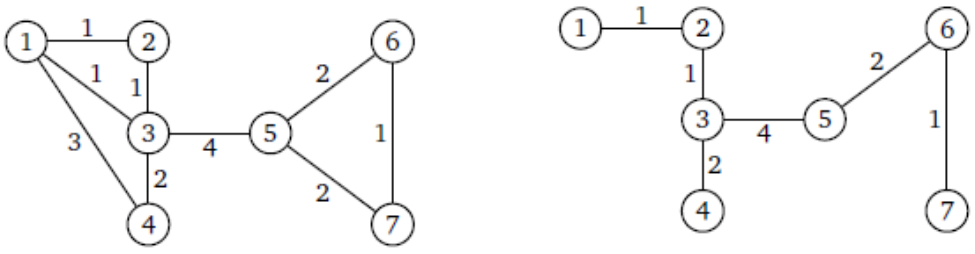
Tras unas lluvias torrenciales, las calles de una ciudad han quedado seriamente dañadas. La institución competente no puede arreglar todas las calles debido al elevado coste que ello supondría, por lo que han decidido volver a pavimentar solo aquellas que les permitan ir de una intersección a otra de la ciudad. Quieren gastarse lo menos posible en la pavimentación, teniendo en cuenta que el coste es directamente proporcional a la longitud de las calles que hay que pavimentar. Desarrolla un algoritmo que permita solucionar de forma óptima este problema con el menor coste. La resolución de este problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).

Si consideramos las intersecciones como vértices de un grafo, donde las aristas son las calles de la ciudad, etiquetadas con el coste correspondiente de pavimentación (valores reales positivos), lo que se obtiene es un grafo valorado no dirigido (se supone que las calles son de dos direcciones) y conexo.

El objetivo es pavimentar las suficientes calles para poder acceder a todas las intersecciones desde cualquier otra, pero gastando lo mínimo en pavimentación. Eso significa encontrar un subgrafo que contenga a todos los vértices, que siga siendo conexo y que el coste total de sus aristas sea mínimo; es decir, se quiere encontrar un árbol (libre) de recubrimiento de coste mínimo (si el subgrafo contiene un ciclo, se puede eliminar una arista de dicho ciclo sin perder la propiedad de ser conexo y logrando un coste total menor).

En la siguiente figura se muestra un grafo no dirigido valorado y conexo, y uno de sus árboles de recubrimiento de coste mínimo.



(a) Grafo no dirigido valorado y conexo

(b) Árbol de recubrimiento de coste mínimo

La idea para construir un árbol de recubrimiento se refleja en el siguiente esquema:

$AR := \emptyset$; *candidatas* := A
mientras queden vértices sin conectar **hacer**
    a := seleccionar(*candidatas*) ; *candidatas* := *candidatas* − $\{a\}$
    si sin-ciclos?($AR \cup \{a\}$) **entonces** $AR := AR \cup \{a\}$ **fsi**
fmientras

Dado un grafo valorado conexo, cuyas aristas tienen asociados valores numéricos no negativos, se dice que un subconjunto de aristas T es prometedor si se puede extender (añadiendo aristas) para obtener un árbol de recubrimiento de coste mínimo. El conjunto vacío siempre es prometedor (pues todo grafo conexo contiene al menos un subgrafo acíclico conexo que alcanza a todos los vértices), así que para que el árbol de recubrimiento finalmente obtenido sea de coste mínimo, es necesario definir una función de selección de aristas adecuada que mantenga AR prometedor en cada etapa.

Presentamos aquí dos estrategias diferentes que se corresponden con algoritmos bien conocidos:

- Escoger la arista de menor valor entre las restantes, manteniendo conexo el subgrafo que se está construyendo (algoritmo de Prim).

- Escoger la arista de menor valor entre las restantes, aunque el subgrafo resultante no sea conexo (algoritmo de Kruskal).

En realidad, el algoritmo de Prim parte de un vértice cualquiera y va extendiendo el árbol de recubrimiento, incorporando en cada etapa un nuevo vértice.

En cambio, el algoritmo de Kruskal parte de un bosque de árboles formados por un único vértice cada uno, y en cada etapa del algoritmo se conectan un par de árboles mediante una arista, hasta que en el bosque quede un único árbol.

La corrección de ambos algoritmos se basa en el siguiente resultado:

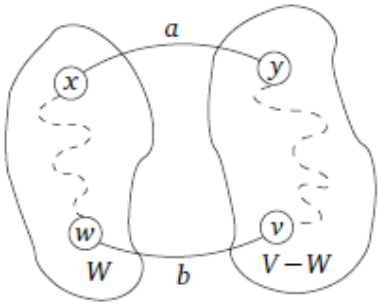
Lema

Sea G = ⟨ V, A ⟩ un grafo valorado conexo con aristas de valores numéricos no negativos. Sean W ⊂ V, T ⊂ A prometedor tal que si ⟨ v,w ⟩ ∈ T entonces v,w ∈ W o v,w ∉ W, y sea a = ⟨ x, y ⟩ ∈ A una de las aristas con menor valor con x ∈ W e y ∉ W. Entonces T ∪ {a} es prometedor.

Demostración

Sea R = ⟨ V, B ⟩ un árbol de recubrimiento de coste mínimo de G con T ⊂ B (R existe porque T es prometedor). Si a ∈ B entonces ya tenemos que T ∪ {a} es prometedor.

Por otra parte, si a ∉ B, se tiene que en ⟨ V, B∪{a} ⟩ habrá un ciclo y formando parte de ese ciclo habrá una arista b = ⟨ v,w ⟩ ∈ B con w ∈ W ∧ v ∉ W, como ilustra la figura:



Selección de aristas para árbol de recubrimiento

Por las hipótesis del lema se tiene que b ∉ T y valor(a,G) ≤ valor(b,G), por lo que T ∪{a} ⊆ C = (B−{b})∪{a} y, además, ⟨ V, C ⟩ es también un árbol de recubrimiento de G y de coste mínimo, puesto que R lo era y hemos sustituido una arista por otra de valor no mayor.

Hemos explicado que el algoritmo de Prim comienza con un vértice cualquiera y va "conectando" vértices en cada etapa.

Si W ⊂ V es el conjunto de vértices ya conectados y ARM es el conjunto de aristas seleccionadas por el algoritmo de Prim, se cumple que todas las aristas en ARM tienen su origen y su destino en W. Supongamos que al comienzo de la etapa i-ésima ARM es

prometedor y que se selecciona la arista a = ⟨ x, y ⟩ ∈ A, eso significa que x ∈ W ∧ y ∉ W y que el coste de a es el menor de entre todas las aristas en A que tienen su origen en W y su destino en V −W.

Así pues, se cumplen todas las hipótesis del lema, por lo que ARM ∪ {a} sigue siendo prometedor (W pasa a ser W ∪ {y}). El algoritmo termina cuando todos los vértices están conectados, es decir, cuando W = V; entonces ARM no solo es prometedor, sino que ⟨V, ARM⟩ es un árbol de recubrimiento de coste mínimo del grafo de partida.

Veamos ahora la corrección del algoritmo de Kruskal.

Sea ARM el conjunto de aristas escogidas para la solución por el algoritmo antes de comenzar la etapa i-ésima y que forman un "bosque de recubrimiento" (un conjunto de árboles libres que recubren todo el conjunto de vértices del grafo); en dicha etapa se selecciona una arista a = ⟨ x, y ⟩ ∈ A, que solo es finalmente incorporada a la solución si no forma ciclos con las demás aristas en ARM. Eso significa que x e y no pueden pertenecer al mismo árbol.

Si denominamos W el subconjunto de vértices conectados (mediante un árbol) al que pertenece x, entonces se tiene que en ARM no hay aristas con el origen en W y el destino en V − W, y que a tiene el menor coste de entre todas las aristas en A que tienen su origen en W y su destino en V − W.

De nuevo se cumplen todas las hipótesis del lema, por lo que ARM ∪ {a} sigue siendo prometedor.

El algoritmo termina cuando el bosque contiene un único árbol; entonces ARM corresponde a un árbol de recubrimiento y, como ARM es prometedor, ⟨ V, ARM ⟩ es un árbol de recubrimiento de coste mínimo del grafo de partida.

2. Algoritmo <u>completo</u> a partir del refinamiento del esquema general (3 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad. Si se trata del esquema de programación dinámica deben darse las ecuaciones de recurrencia.

Recordemos que el algoritmo de Prim comienza con un vértice cualquiera y en cada etapa conecta el vértice que menos le cueste. Para cada vértice que queda por incorporar al árbol que va construyendo el algoritmo, llevaremos calculado el coste mínimo para conectarlo al árbol (coste-mín) y el vértice del árbol que permite la conexión (conexión). De esta forma, bastará encontrar el mínimo de entre los valores de coste-mín; la correspondiente conexión nos permitirá determinar la arista a incorporar al árbol.

Los valores de coste-mín y conexión deben actualizarse para los vértices que todavía no se han incorporado

al árbol; para ello, se comprueba si el último vértice incorporado, sea v, permite reducir el coste de conexión a los demás; es decir, si para algún w se tiene que gv-valor(v,w,G) < coste-mín(w); en caso afirmativo, el valor de conexión(w) pasaría a ser v. Para facilitar

estas comprobaciones, elegimos $+\infty$ como valor especial en la matriz de valores para las aristas que no existen en el grafo.

Extendemos las operaciones aritméticas para operar con valores infinitos de la forma esperada.

Si el algoritmo comienza incorporando el vértice 1 al árbol, para el resto de vértices del grafo los valores de coste-mín[i] y conexión[i] se inicializarán con el valor de gv-valor(1, i,G) y 1 respectivamente. Para que los vértices ya incorporados al árbol no interfieran en la búsqueda del mínimo ni en la actualización de los costesmínimos, su valor de coste-mín será −1.

Siguiendo todas estas ideas, el algoritmo resultante es el siguiente:

```
tipos
    arista = reg
                origen : 1..n
                destino : 1..n
             freg
ftipos
{ G es conexo }
fun Prim1(G : grafo-val[n]) dev ARM : conjunto[arista]
var coste-mín[2..n] de real_∞, conexión[2..n] de 1..n, a : arista
   ARM := cjto-vacio()
   para i = 2 hasta n hacer
      coste-mín[i] := G[1,i] ; conexión[i] := 1
   fpara
   para i = 1 hasta n − 1 hacer
      { encontrar el mínimo en coste-mín }
      mínimo := +∞
      para j = 2 hasta n hacer
         si 0 ≤ coste-mín[j] ∧ coste-mín[j] < mínimo entonces
            mínimo := coste-mín[j] ; elegido := j
         fsi
      fpara
      a.origen := elegido ; a.destino := conexión[elegido]
      añadir(a, ARM)
      coste-mín[elegido] := −1
      para j = 2 hasta n hacer { actualizar los vectores }
         si G[elegido, j] < coste-mín[j] entonces
            coste-mín[j] := G[elegido, j] ; conexión[j] := elegido
         fsi
      fpara
   fpara
ffun
```

3. Estudio del coste <u>del algoritmo desarrollado</u> (0.5 puntos solo si el punto 1 es correcto).

El coste en tiempo de esta implementación está en $O(n^2)$, puesto que se realizan n − 1 etapas y en cada etapa se realizan en secuencia dos bucles de n − 1 iteraciones cada uno.

Como se ve, el coste es independiente del número de aristas en el grafo. En otra implementación distinta, también con el algoritmo de Prim, el coste sería de O(m log n), siendo m el número de aristas.

El coste en espacio adicional está en O(n), por los dos vectores auxiliares (*coste-mín* y *conexión*).