SOLUCIÓN EXAMEN ALGORITMIA. SEMESTRE PRIMAVERA. 11 DE JUNIO DE 2012.

1.- Supongamos que el siguiente algoritmo se ejecuta sobre un número natural n que es potencia de 5. Calcular el orden del algoritmo.

```
proc mickey(n : nat)
        si n = 1 entonces seguir
                si no mouse := n/5
                        para i = 1 hasta 4 hacer
                                mickey(mouse)
                        fpara
        fsi
fproc
```

Solución: mickey es una función recursiva. Tiene una comparación en la que se comprueba si el parámetro n es igual a 1. Si n es diferente a 1, se asigna a la variable mouse el valor n entre 5. Después se llama a la función Mickey(mouse) cuatro veces, por lo tanto la ecuación recursiva que tenemos que resolver es la siguiente:

$$t(n) = \begin{cases} 1 & si\ n = 1 \\ 4 * t\left(\frac{n}{5}\right) & si\ n > 1 \end{cases}$$

Si desarrollamos la ecuación recursiva:

$$t(n) = 4 * t\left(\frac{n}{5}\right) = 4 * 4 * t\left(\frac{n}{5^2}\right) = \cdots = 4^k * t\left(\frac{n}{5^k}\right)$$

Si hacemos el cambio de variable $n = 5^k$ entonces tenemos que:

$$t(n) = 4^k * 1$$

Si deshacemos el cambio de variable

$$t(n) = 4^{\log_5 n} = n^{\log_5 4}$$

Por lo tanto el orden del algoritmo es $O\left(n^{\log_5 4}\right)$

2.- Si tenemos tres vectores ordenados de números reales y distintos A1, A2 y A3. ¿Podemos encontrar un algoritmo que pueda encontrar un elemento en A1 U A2 U A3 en un tiempo menor que Ω(nlogn)?

Solución: Sabemos que el algoritmo más eficaz para buscar un número en un vector es la búsqueda binaria utilizando la estrategia Divide y Vencerás. La búsqueda binaria en un vector ordenado es del orden O(logn). En este caso tenemos 3 vectores que están ordenados. La mezcla de vectores ordenados para formar un vector ordenado tiene una complejidad de O(n) (la mezcla de vectores ordenados se utiliza en el algoritmo mergesort). Por lo tanto, un algoritmo que primero mezcle los tres vectores en O(n) (siendo n el número total de elementos) y después utilice la búsqueda binaria en O(logn) tendrá una complejidad del orden de O(n). Cómo O(n) es la complejidad en el peor caso posible entonces si que podemos encontrar un algoritmo que pueda encontrar

un elemento en A1UA2UA3 en tiempo menor que Ω(nlogn) (siendo Ω(nlogn) la cota inferior).

Otra posible solución es aplicar la búsqueda binaria 3 veces. Si tenemos tres vectores podemos aplicar la búsqueda binaria en cada vector. Por lo tanto tendremos una complejidad de O(k*log(n/k)) siendo k el número de vectores y n el número total de elementos. En este caso tendríamos que estudiar para que valores de n y k es mejor utilizar esta solución que la anterior.

3.- Un camionero conduce desde Bilbao a Málaga siguiendo una ruta dada y llevando un camión que le permite, con el tanque de gasolina lleno, recorrer n kilómetros sin parar. El camionero dispone de un mapa de carreteras que le indica las distancias entre las gasolineras que hay en su ruta. Como va con prisa, el camionero desea pararse a repostar el menor número de veces posible. Deseamos diseñar un algoritmo ávido para determinar en qué gasolineras tiene que parar y demostrar que el algoritmo encuentra siempre la solución óptima.

Solución: Supondremos que existen G gasolineras en la ruta que sigue el camionero entre Bilbao y Málaga, incluyendo una en la ciudad destino, y que están numeradas del 0 (gasolinera en Bilbao) a G–1 (la situada en Málaga). Supondremos además que disponemos de un vector con la información que tiene el camionero sobre las distancias entre ellas de forma que el i-ésimo elemento del vector indica los kilómetros que hay entre las gasolineras i–1 e i. Para que el problema tenga solución hemos de suponer que ningún valor de ese vector es mayor que el número n de kilómetros que el camión puede recorrer sin repostar.

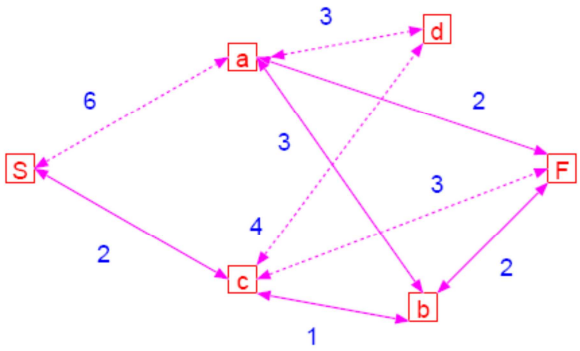
Con todo esto, el algoritmo voraz va a consistir en intentar recorrer el mayor número posible de kilómetros sin repostar, esto es, tratar de ir desde cada gasolinera en donde se pare a repostar a la más lejana posible, así hasta llegar al destino.

Este es el pseudocódigo de una posible solución:

```
TYPE SOLUCION = ARRAY [1..G-1] OF BOOLEAN;
PROCEDURE Deprisa(n:CARDINAL; VAR d:DISTANCIA; VAR
sol:SOLUCION);
        VAR i,numkilometros:CARDINAL;
        BEGIN
        FOR i:=1 TO G-1 DO sol[i]:=FALSE END;
        i:=0;
        numkilometros:=0;
        REPEAT
                REPEAT
                        INC(i);
                        numkilometros:=numkilometros+d[i];
                UNTIL (numkilometros>n) OR (i=G-1);
```

```
            IF numkilometros>n THEN (* si nos hemos pasado... *)
                DEC(i); (* volvemos atras una gasolinera *)
                sol[i]:=TRUE; (* y repostamos en ella. *)
                numkilometros:=0; (* reset contador *)
            END
        UNTIL (i=G-1);
END Deprisa;
```

4.- Un problema que aparece con frecuencia en las redes de ordenadores es el de encontrar la ruta que permite transmitir los datos de un ordenador *S* a otro *F* a mayor velocidad. Las conexiones entre máquinas permiten una velocidad máxima de transmisión que supondremos idéntica en ambos sentidos. La velocidad de transmisión a través de una ruta que utiliza varias conexiones está determinada en la práctica por la conexión más lenta.



(a) Describe cómo resolver este problema utilizando para ello un algoritmo voraz o de programación dinámica. Detalla paso a paso cómo se irían completando las tablas que utiliza el algoritmo elegido.

Solución: En este ejercicio tenemos que encontrar el camino de máxima conexión entre dos nodos de un grafo. El algoritmo de Dijkstra (algoritmo voraz) encuentra el camino mínimo entre un nodo y el resto de nodos del grafo. Por lo tanto en este problema deberemos utilizar una modificación del algoritmo de Dijkstra.
El algoritmo de Dijkstra utiliza dos conjuntos de vértices C (candidatos) y S (solución). S contiene en todo momento el conjunto de vértices cuya distancia es mínima al origen es conocida y C contiene los vértices restantes. Al principio, S contendrá únicamente el origen, y al final, todos los vértices. En cada iteración, cogemos el vértice v de C cuya distancia al origen es mínima. Utilizamos un vector D para mantener la longitud del camino especial más corto desde el origen a cualquier vértice del grafo. Cada vez que seleccionamos un vértice v y lo incluimos en S recalculamos D mediante la siguiente ecuación:

Con L[v,w] el valor del arco que conecta los vértices v y w.

En este problema se pide encontrar el camino de velocidad de conexión máxima, además la velocidad de transmisión de una ruta está determinada por

la conexión más lenta. Por lo tanto en nuestro algoritmo iremos escogiendo los vértices cuya distancia al origen es máxima y cuando recalculamos D lo haremos utilizando la siguiente ecuación:

$$D[w] = \max(D[w], \min(D[v], L[v, w]))$$

El término min(D[v], L[v,w]) expresa que la velocidad de conexión entre el origen y un nodo está determinada por la velocidad más lenta.

Si resolvemos el problema utilizando el algoritmo propuesto esto serían los pasos:

Etapa	V	C	S	D
Inic		{a,b,c,d,F}		[6 -1 2 -1 -1]
1	a	{b,c,d,F}	{a}	[6 3 2 -3 2]
2	b	{c,d,F}	{a,b}	[6 3 2 3 2]
3	d	{c,F}	{a,b,d}	[6 3 3 3 2]
4	c	{F}	{a,b,c,d}	[6 3 3 3 3]
5	F		{a,b,c,d,F}	[6 3 3 3 3]

Como tenemos que calcular el máximo cuando no hay conexión entre dos vértices ponemos el valor -1.
Para conocer los vértices por los que pasa el camino máximo tenemos que añadir una nueva tabla P. Tendremos que modificar nuestro algoritmo y añadir:

```
Para cada w de C hacer
        Si D[w] < min(D[v], L[v,w]) entonces
                D[w] = min(D[v], L[v,w])
                P[w] = v
        Finsi
FinPara
```

En este caso la tabla tendrá los siguiente valores al final del proceso:
P = [S a d a c]

Otra opción para resolver el problema es utilizar un algoritmo de programación dinámica. El algoritmo de Floyd calcula la longitud del camino más corto entre cualquier par de vértices. Se construye una matriz D que almacena el camino más corto entre cada par de vértices. Se inicializa D con la matriz de adyacencia L y a continuación se hacen n iteraciones, de manera que tras cada iteración k, D almacenará las longitudes de los caminos mínimos que utilizan como vértices intermedios únicamente {1, 2, .., k}. La matriz Dk de la k-esima iteración se calcula de la siguiente forma:

$$D_k[i, j] = \min(D_{k-1}[i, j], D_{k-1}[i, k] + D_{k-1}[k, j])$$

Como en este problema estamos buscando la ruta con máxima velocidad de conexión y además la conexión de la ruta viene determinada por la conexión de

menor velocidad modificamos el algoritmo de Floyd de tal forma que la ecuación con la que calculamos la matriz Dk es la siguiente:

$$D_k[i,j] = \max(D_{k-1}[i,j], \min(D_{k-1}[i,k], D_{k-1}[k,j]))$$

Si queremos saber por donde pasa el camino de conexión máxima debemos utilizar una tabla P.
Si resolvemos el problema utilizando la modificación del algoritmo de Floyd obtendríamos los siguientes valores para la matriz D

D0 =

0	6	-1	2	-1	-1
6	0	3	-1	3	2
-1	3	0	1	-1	2
2	-1	1	0	4	3
-1	3	-1	4	0	-1
-1	2	2	3	-1	0

D1 =

0	6	-1	2	-1	-1
6	0	3	2	3	2
-1	3	0	1	-1	2
2	2	1	0	4	3
-1	3	-1	4	0	-1
-1	2	2	3	-1	0

D2 =

0	6	3	2	3	2
6	0	3	2	3	2
3	3	0	2	3	2
2	2	2	0	4	3
3	3	3	4	0	2
2	2	2	3	2	0

D3 =

0	6	3	2	3	2
6	0	3	2	3	2
3	3	0	2	3	2
2	2	2	0	4	3
3	3	3	4	0	2
2	2	2	3	2	0

D4 =

0	6	3	2	3	2
6	0	3	2	3	2
3	3	0	2	3	2
2	2	2	0	4	3
3	3	3	4	0	3
2	2	2	3	3	0

D5 =

0	6	3	3	3	3
6	0	3	3	3	3
3	3	0	3	3	3
3	3	3	0	4	3
3	3	3	4	0	3
3	3	3	3	3	0

D6 =

0	6	3	3	3	3
6	0	3	3	3	3
3	3	0	3	3	3
3	3	3	0	4	3
3	3	3	4	0	3
3	3	3	3	3	0

P =

0	0	a	d	a	d
0	0	0	d	0	d
a	0	0	d	a	d
d	d	d	0	0	0
a	0	a	0	0	c
d	d	d	0	c	0

De forma que la ruta de máxima velocidad de conexión entre S y F será:
s→a→d→c→F

(b) Describe cómo resolver este problema utilizando para ello un algoritmo de ramificación y poda.

Solución: Para resolver este problema primero debo plantear como serán los nodos del árbol de búsqueda. En cada nodo tendré un vector en el que representaré la posible solución (es decir los ordenadores por los que pasa la ruta de máxima conexión). Por ejemplo el nodo inicial será: [S]
Después puedo ir explorando las diferentes combinaciones. Los nodos sucesores del nodo inicial serán [S a] y [S c]. Y así puedo ir explorando el árbol hasta encontrar la solución. Si no utilizo ninguna información extra puedo explorar el árbol en profundidad o en anchura. Si quiero podar el árbol necesito un valor de coste que me estime la velocidad de conexión del nodo que estoy explorando. De esta forma iré explorando los nodos más prometedores analizando cada vez el nodo abierto (que no ha sido analizado) cuya conexión estimada sea máxima.