UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA	**Programación III** Código de asignatura (marcar con círculo): Sistemas: __ 402048 (plan viejo) __ 532044 (plan nuevo) Gestión: __ 41204- (plan viejo) __ 542046 (plan nuevo)	**Prueba Presencial** **Segunda Semana** Febrero de 2004 Duración: **2 horas** Material permitido: **NINGUNO**

Apellidos: _____ **DNI:** _____

Nombre: _____ **Centro donde entregó la práctica:** _____ **e-mail:** _____

Cuestión 1 (1 puntos). ¿En qué orden está el tiempo de ejecución del siguiente algoritmo? Justifica tu respuesta.

```
Procedimiento h(n,i,j)
     si n>0 entonces
          h(n-1,i,6-i-j);
          escribir i "->" j;
          h(n-1,6-i-j,j);
     fsi;
```

T(n)=2·T(n-1)+1, luego

número de llamadas: a=2

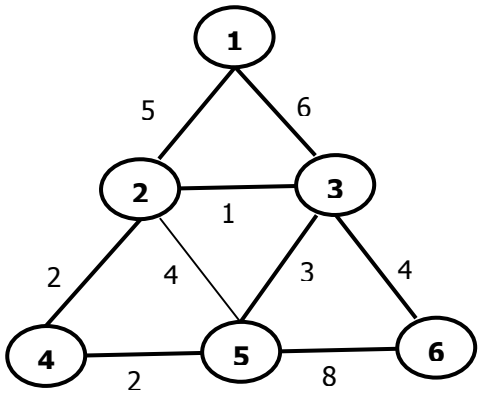
reducción por sustracción: b=1

$c \cdot n^k=1$, k=0

Aplicando la recurrencia para el caso cuando a>1: $T(n) \in \Theta(a^{n \text{ div } b})$; $T(n) \in \Theta(2^n)$

Cuestión 2 (2 punto).

Aplica el algoritmo de Kruskal al siguiente grafo indicando claramente en cada paso qué arista se selecciona, la evolución de las componentes conexas y la evolución de la solución.



Se ordenan las aristas de menor a mayor coste: {(2,3),(4,5),(2,4),(3,5),(3,6),(2,5),(1,2),(1,3),(5,6)}		
Arista seleccionada	**Evolución de las componentes conexas**	**Evolución de la solución**
	{1} {2} {3} {4} {5} {6}	∅
(2,3)	{1} {2,3} {4} {5} {6}	{(2,3)}
(4,5)	{1} {2,3} {4,5} {6}	{(2,3), (4,5)}
(2,4)	{1} {2,3,4,5} {6}	{(2,3), (4,5), (2,4)}
(3,5)	Se rechaza porque los extremos de la arista ya están en la misma comp. conexa	

La resolución del problema debe incluir, por este orden:

1. Elección razonada del esquema algorítmico
2. Descripción del esquema usado e identificación con el problema
3. Estructuras de datos
4. Algoritmo completo a partir del refinamiento del esquema general
5. Estudio del coste

(3,6)	{1} {2,3,4,5,6}	{(2,3), (4,5), (2,4), (3,6)}
(2,5)	Se rechaza porque los extremos de la arista ya están en la misma comp. conexa	
(1,2)	{1,2,3,4,5,6}	{(2,3), (4,5), (2,4), (3,6), (1,2)}
Proceso terminado porque sólo queda una única componente conexa		

Cuestión 3 (2 puntos). ¿Cuándo resulta más apropiado utilizar una exploración en anchura que en profundidad?

- Cuando el grafo tiene ramas infinitas
- Cuando se busca el camino de menor número de nodos o aristas
- Cuando los nodos finales se encuentran cerca del nodo raíz

¿En que casos puede que la exploración en anchura no encuentre solución aunque ésta exista?

- Que exista una solución quiere decir que existe un camino desde el nodo inicial al nodo final. No tiene sentido, entonces, hablar de que la "solución" esté en una componente conexa diferente. Por otro lado, el caso en que un nodo genere infinitos hijos es un caso excepcional que no debería darse. Por todo ello, y salvo esta última excepción, la exploración en anchura siempre encuentra la solución, si ésta existe.

¿Y la exploración en profundidad? Razona tu respuestas.

- Si existen ramas infinitas sin nodos finales puede que la exploración en profundidad quede atrapada en una de ellas y no encuentre solución aunque ésta exista.

Problema (5 puntos). Dado un grafo dirigido, finito, con ciclos y con todas las aristas de coste unitario, implementar un algoritmo que devuelva el número de nodos que contiene el camino más largo sin ciclos que se puede recorrer desde un determinado nodo.

Elección del esquema.

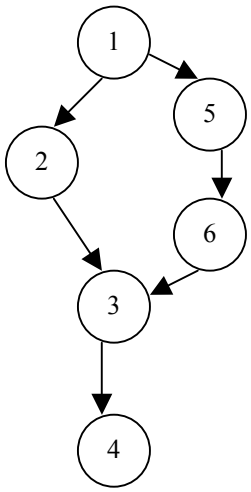
Hallar el camino más largo desde un nodo en un grafo finito puede verse como un problema de determinar cuál es la profundidad máxima del árbol. Para ello, será necesario recorrer todos los caminos sin ciclos del grafo, almacenando la longitud del más largo encontrado hasta el momento. El esquema de búsqueda exhaustiva en profundidad nos permite resolver este problema.

Descripción del esquema e identificación con el problema.

En la página 330 del libro de texto (Brassard, 1997) se presenta el esquema de recorrido en profundidad:

```
procedimiento rp(nodo)
        {nodo no ha sido visitado anteriormente}
        visitado[nodo]:=cierto;
        para cada hijo en adyacentes[nodo] hacer
                si visitado[nodo]=falso entonces
                        rp(hijo)
                fsi
        fpara
```

Así planteado, un nodo no se exploraría 2 veces aunque llegáramos a él desde una rama distinta. Esto es incorrecto y se debe a que el vector de nodos visitados se plantea de forma global. Veamos un ejemplo:



La exploración llega al nodo 1 y lo marca como visitado. Análogamente sigue marcando como visitados los nodos 2, 3 y 4, hallando un camino de longitud 4. Cuándo el control regresa al nodo 1 para explorar ramas alternativas, se explora el nodo 5, el 6, pero cuando llega al nodo 3, se detiene la exploración porque 3 está marcado como visitado. El nuevo camino encontrado tiene 4 nodos en vez de 5 que sería lo correcto.

Este problema lo podemos corregir si el vector de visitados tiene un ámbito local en vez de global, y se pasa una copia de padres a hijos (paso de parámetro por valor).

Estructuras de datos

Necesitaremos una estructura para grafo. La más compacta y sencilla de usar es un vector en el que la componente i tiene un puntero a lista de nodos adyacentes al nodo i. Por tanto, también necesitaremos implementar una lista de nodos que puede ser, simplemente, una lista de enteros.

Además, necesitaremos un vector de nodos visitados por rama.

Algoritmo completo

```
procedimiento longitud(nodo,visitados) dev long
        {nodo no ha sido visitado anteriormente}
        visitados[nodo]:=cierto;
        long_max:=0;
        para cada hijo en adyacentes[nodo] hacer
                si visitados[nodo]=falso entonces
                        long:=longitud(hijo,visitados);
                        si long>long_max entonces
                                long_max:=long
                        fsi;
                fsi
        fpara
        devolver long_max+1;
```

La llamada inicial se realiza con el nodo inicial, y los elementos del vector de visitados inicializados a falso.

Estudio del coste

Sea n el número de nodos del grafo. Si suponemos un grafo denso (en el que todos los nodos están interconectados entre sí), tenemos que la longitud del camino máximo será n. Cada

llamada recursiva, entonces, supondrá una reducción en uno del problema. Así mismo, el número de llamadas recursivas por nivel también ira disminuyendo en uno puesto que en cada nivel hay un nodo adyacente más que ya ha sido visitado. Por último, dentro del bucle se realizan al menos n comparaciones. Por tanto, podemos establecer la siguiente recurrencia:

$T(n)=(n-1)\cdot T(n-1)+n$

número de llamadas: $a=n-1$

reducción por sustracción: $b=1$

$c\cdot n^k=n$, $k=1$

Aplicando la recurrencia para el caso cuando a>1: $T(n)\in O(a^{n \text{ div } b})$; $T(n)\in O(n^n)$

Al mismo resultado se puede llegar razonando sobre la forma del árbol de exploración.