Solución al examen de Septiembre 2003

Programación III

Cuestión 1

El procedimiento `uno` tiene una instrucción condicional de coste constante. Dentro de la condición tiene o bien una instrucción constante o bien dos llamadas recursivas. Ambas invocan la función con un tamaño $n/3$. En el segundo caso hay otra instrucción simple añadida. La expresión queda $T(n) = 1 + T(n/3) + 1 + T(n/3)$ lo que equivale a que $T(n) = 2T(n/3) + c$ con c constante. Aplicando la resolución genérica de las expresiones del tipo $T(n) = aT(n/b) + cn^k$ con $k = 0$, $a = 2$ y $b = 3$ queda que $2 > 3^0$ y por tanto la función T(n) tiene un coste $O(n^{log_3{}^2})$.

De forma análoga, el segundo algoritmo `dos` está formado por una instrucción condicional cuyo cuerpo incluye secuencialmente (i) una llamada recursiva de tamaño $n/4$, (ii) una instrucción simple, (iii) otra llama recursiva de tamaño $n/4$, (iv) un bucle en el que se repite n veces un cálculo consistente en llamar dos veces a una función $h(n, r, i)$ de coste lineal, más una instrucción simple, y por último (v) una instrucción simple y otra llamada recursiva de tamaño $n/4$. Sumando los 5 términos nos sale $T(n) = T(n/4)+1+T(n/4)+n*(2n+1)+1+T(n/4)$, lo que equivale a $T(n) = 3T(n/4) + 4n^2$ y siendo $3 < 4^2$ el coste es $O(n^2)$.

Cuestión 2

Un procedimiento parcial de ordenación por selección nos da los m elementos más pequeños con coste $O(m.n)$. Una ordenación eficiente sin embargo lo haría en $O(n.log\, n)$. Si $m \approx n$, el coste vendría a ser cuadrático, lo cual nos haría desechar el procedimiento de selección, sin embargo con $m << n$ el orden $O(n.m)$ se puede considerar lineal, y en este caso el algoritmo de selección puede ser más eficiente.

Cuestión 3

Cada casilla se asimila a un nodo. Casillas *libres* adyacentes tendrían aristas dirigidas en ambas direcciones. El peso sería unitario para cada arista. Las casillas de tipo *ocupada* no tienen aristas origen ni destino.

De esta forma, un algoritmo de *Dijkstra* puede hallar el camino más corto de un nodo LLEGADA a todos los demás, incluyendo el nodo SALIDA.

En términos de coste, sin embargo es necesario tener en cuenta que si el laberinto es un cuadrado de lado n, el grafo tendrá $v = n^2$ nodos y alrededor de $a = 4n^2$ aristas. En el análisis de coste de la resolución de *Dijkstra* si v (número de nodos del grafo) es lo suficientemente grande hace cierta la expresión $a << v^2$ y por tanto podemos aproximarnos al coste $O((a + v).log\, v)$.

Problema

Se trata de un grafo no dirigido de 10 nodos $n_1, n_2, \ldots, n_{10}$ con una matriz simétrica de costes:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	-	3	4	5	6	7	0	1	2	3
2	3	-	5	6	7	0	1	2	3	4
3	4	5	-	7	0	1	2	3	4	5
4	5	6	7	-	1	2	3	4	5	6
5	6	7	0	1	-	3	4	5	6	7
6	7	0	1	2	3	-	5	6	7	0
7	0	1	2	3	4	5	-	7	0	1
8	1	2	3	4	5	6	7	-	1	2
9	2	3	4	5	6	7	0	1	-	3
10	3	4	5	6	7	0	1	2	3	-

Se trata de conseguir minimizar el coste de los enlaces asegurando únicamente la conectividad de la red.

El enunciado describe un problema de optimización en el que se nos pide que el grafo sea conexo ("asegurar la conectividad de la red") y contenga un árbol de expansión mínimo ("que el coste sea mínimo"), ya que la conectividad se asegura no dejando subgrafos no conexos.

ELECCIÓN DEL ESQUEMA: Con las condiciones descritas podemos usar algoritmos que resuelvan el problema del árbol de expansión mínimo, dentro de la familia de los algoritmos voraces.

DESCRIPCIÓN DEL ESQUEMA: Se elige cualquiera de los algoritmos expuestos en el temario *Kruskal* o *Prim*, por ejemplo éste último.

función Prim(G:grafo):T:conjunto de aristas
$T \leftarrow \emptyset$
$B \leftarrow 1$
mientras $B \neq N$ **hacer** {
  min $e = \{u, v\}$ tq. $u \in B$ y $v \in N \backslash B$
  $T \leftarrow T \bigcup \{e\}$
  $B \leftarrow B \bigcup \{v\}$
}
dev T

Hemos tomado 1 como nodo arbitrario. El conjunto B va a ir conteniendo los nodos del subgrafo ya conexo y el conjunto T irá teniendo en cada iteración aquellas aristas del árbol de expansión mínimo que contiene los nodos de B. El conjunto de candidatos es B, la condición de finalización es que $B = N$ y la función de optimización es elegir aquella arista del subgrafo

B que conecte con algún nodo de $N \backslash B$ con menor coste.

Estructuras de datos: El grafo se representará mediante una matriz de costes. La estructura de datos tendrá un método que implementa el cálculo de la distancia entre dos nodos. En el caso de esta implementación la distancia entre dos nodos i y j es el valor de la matriz de distancias, y su coste $O(1)$.

Algoritmo completo a partir del esquema general:

```
función Prim(G:grafo):T:conjunto de aristas
T ← ∅
B ← 1
para i ← 2 hasta n hacer
        masProximo[i] ← 1
        distanciaMin[i] ← G.distancia[i, 1]
repetir n − 1 veces{
   min ← ∞
   para j ← 2 hasta n hacer {
  \* Selecciona la mejor arista entre B y N\B *\
        si (distanciaMin[j] ≥ 0) ∧ (distanciaMin[j] < min) ent {
                        distanciaMin[k] ← G.distancia(j, k)
                        masProximo[j] ← k
        }
   }
   T ← T ⋃{masProximo[k], k} \* Añade la arista *\
   distanciaMin[k] ← −1
   para j ← 2 hasta n hacer {
        si G.distancia(j, k) < distanciaMin[j] ent {
                        distanciaMin[k] ← G.distancia(j, k)
                        masProximo[j] ← k
        }
  }
}
dev T
```

Coste: El coste del algoritmo de Prim es $O(n^2)$, que puede mejorarse utilizando una representación de montículos para el vector $distanciaMin[]$

Optimalidad: El algoritmo de Prim encuentra la solución óptima. Se puede demostrar por inducción sobre T que añadir la arista más corta $\{e\}$ que sale de T (Lema 6.3.1 del texto base) forma en $T \bigcup \{e\}$ un árbol de recubrimiento mínimo que contendrá al final $n-1$ aristas y todos los nodos del grafo G.

Aplicación al problema: Tenemos los siguientes conjuntos inicialmente $B = \{1\}$ y la arista mínima entre un nodo de B y otro de $N \backslash B$ es $u = (1, 7)$ con valor 0.

Los valores de B y la u elegida en cada momento evolucionan como sigue:

$B = \{1, 7\}$ $u = (7, 9)$ Coste: 0

$B = \{1, 7, 9\}$ $u = (7, 2)$ Coste: 1

$B = \{1, 2, 7, 9\}$ $u = (2, 6)$ Coste: 0

$B = \{1, 2, 6, 7, 9\}$ $u = (6, 10)$ Coste: 0

$B = \{1, 2, 6, 7, 9, 10\}$ $u = (6, 3)$ Coste: 1

$B = \{1, 2, 3, 6, 7, 9, 10\}$ $u = (3, 5)$ Coste: 0

$B = \{1, 2, 3, 5, 6, 7, 9, 10\}$ $u = (6, 3)$ Coste: 0

$B = \{1, 2, 3, 5, 6, 7, 8, 9, 10\}$ $u = (9, 8)$ Coste: 1

$B = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ $u = (9, 8)$ Coste: 1

Coste del árbol de expansión mínimo: 4