Algoritmos y Estructuras de Datos II, Grado Ingeniería Informática

Examen 10 de septiembre de 2011

(OJO, se cuentan las soluciones con todo detalle para que se entienda bien, pero en el examen no hace falta que os extendáis tanto en las explicaciones, ni que deis absolutamente todos los pasos en los cálculos, ni explicar con tanto detalle las implementaciones)

Problema 1:

SOLUCIÓN al ejercicio 1:

Problema 2:

SOLUCIÓN al ejercicio 2:

Parte común de los problemas 3 al 6:

Problema del grafo multietapa escalonado.

Consideramos un "grafo multietapa escalonado con pesos", $G = (V, A)$, donde los vértices están agrupados en etapas disjuntas $V = \{V_1, V_2, \dots V_l\}$ y sólo hay aristas de nodos en un nivel a niveles posteriores (de V_i a V_{i+k}, con $k = 1, 2, \dots, l - i$). Cada arista tiene un peso positivo. Consideramos que la primera y última etapa tienen cada una un único nodo. Se trata de encontrar el camino de longitud mínima desde el nodo de la primera etapa al de la última etapa.

Problema 3:

Diseñar un algoritmo voraz que encuentre una buena forma de resolver el problema. Hay que indicar la heurística que se utiliza para intentar acercarse a la solución óptima. ¿Garantiza ese algoritmo la solución óptima?

SOLUCIÓN al ejercicio 3:

Consideramos un esquema básico de avance rápido:

```
voraz(C: conjunto_candidatos; var S: conjunto_solución)
   S = ∅
   MIENTRAS C ≠ ∅ Y NO solucion(S)
      x=seleccionar(C)
      C = C − {x}
      SI factible(S ∪ {x})
         insertar(S,x)
      FINSI
```

FINMIENTRAS
SI solucion(S)
    solución S, con valor objetivo(S)
EN OTRO CASO
   "No se puede encontrar una solución"
FINSI

Tenemos que definir cada una de las funciones básicas y de los conjuntos que se utilizan:

- En cada paso, el conjunto C de posibles candidatos varía, y está formado por todos los movimientos que se pueden realizar desde el nodo donde nos encontramos. Por ejemplo, si consideramos un grafo con 7 nodos agrupados en cuatro niveles: $N_1 = \{1\}$, $N_2 = \{2, 3\}$, $N_3 = \{4, 5, 6\}$ y $N_4 = \{7\}$, y con pesos los de la matriz de adyacencia

	1	2	3	4	5	6	7
1	0	7	3	6	–	–	9
2	–	0	–	–	–	–	3
3	–	–	0	–	6	4	12
4	–	–	–	0	–	–	2
5	–	–	–	–	0	–	–
6	–	–	–	–	–	0	3
7	–	–	–	–	–	–	0

  donde un - representa que no hay arista entre los nodos origen (en la fila) al destino (en la columna), desde cada nodo se puede ir a los nodos destino para los que el valor en la matriz sea distinto de 0 y de -. Para recorrer estos valores, para el nodo i solo es necesario recorrer la fila i de la matriz desde la columna $i + 1$ en adelante.

- El conjunto S puede implementarse con un array solucion sol:array[1..$maxpasos$] de $1 \ldots N$, con $N = |V|$, donde $maxpasos$ es el máximo número de pasos, que se puede establecer al número de niveles, y $sol[i] = j$ indica que en el paso i se llega al nodo j.

- La función de inicialización del conjunto solución puede consistir en hacer $sol[1] = 1$ (ya que siempre se parte del nodo 1) y el resto de posiciones de sol a cero.

- La función $C \neq \emptyset$ consistirá en comprobar que hay nodos a los que se puede ir desde el que nos encontramos: si estamos en el nodo i ver si para algún j es $m[i, j]$ distinto de cero y de -.

- solucion(S) devuelve si ya hemos encontrado la solución. En este caso se ha encontrado cuando llegamos al nodo N.

- En la función de selección podemos decidir tomar la arista más corta a partir del nodo donde nos encontramos. En el ejemplo, desde el nodo 1 iríamos al 3, con arista (1,3) de peso 3. Esta decisión hace que no obtengamos la solución óptima (por ejemplo si vamos de 1 a 7 el camino tiene peso 9, y si tomamos la arista (1,4) el camino final tiene peso 8). Así, la estrategia de selección no nos asegura que lleguemos a una solución óptima, ni siquiera que lleguemos a solución, lo que pasaría, por ejemplo, si del nodo 3 no saliera ninguna arista.

  Se puede intentar mejorar la heurística de selección sumando al peso de cada arista la media de los pesos de todas las aristas del grafo multiplicada por el número mínimo de pasos a dar desde ese nodo, lo que es la diferencia entre el último nivel del grafo y el nivel donde se encuentra el nodo. En el ejemplo, como tenemos 10 aristas con peso total 55 la media es 5.5, y para decidir el nodo al que ir desde 1, consideramos valores: para el nodo 2 el valor 7+11 (pues quedan dos niveles para llegar al nivel 4), para el 3 el 3+11, para el 4 el valor 6+5.5 (queda un nivel), y para el 7 el valor 9 (no queda ningún nivel). La decisión sería ir directamente al nodo 7. Esta heurística, aunque es mejor que la anterior al considerar más información, tampoco nos da la solución óptima.

  Se puede mejorar si desde cada nivel se hace la media de las aristas a partir de ese nivel. Por ejemplo, una vez que llegamos al nivel 2 (nodos 2 y 3), hay 6 aristas, con un peso total 25, y con peso medio 4.16. A partir del nivel 3 hay 2 aristas, con peso medio 2.5. Para decidir donde ir desde el nodo 1 consideramos los valores de los posibles destinos: 7+8.32 para el nodo 2, 3+8.32 para el 3, 6+2.5 para el 4, y 9 para el 7. La decisión sería ir al nodo 4, que en este caso nos lleva a la solución óptima: 1-4-7, pero que no nos la garantiza en general. Por ejemplo, si tuvieramos las aristas (3,5) y (5,7), cada una con peso 1, el camino 1-3-5-7 tiene peso 5, mientras que en el primer paso descartaríamos ir al nodo 3.

- La operación $C = C - \{x\}$ no se programará explícitamente, ya que una vez que hemos decidido ir al nodo j desde el nodo i pasamos a analizar los valores en la fila j.
- No utilizamos función factible, pues sólo seleccionamos movimientos que nos llevan a nodos a los que podemos ir. La función de selección devolverá un valor especial cuando desde un nodo no se puede seguir avanzando.
- La inclusión en el conjunto solución ($S \cup \{x\}$) será almacenar el nodo seleccionado.
- La función que devuelve el valor objetivo simplemente devuelve el número de nodos almacenados en *sol* menos 1.

Un esquema de avance rápido para este problema puede ser:

```
grafo-voraz(sol: array[1..niveles] de 0...N)
    sol ← 0
    sol[1] = 1
    paso = 1
    factible=VERDADERO
// cuando factible sea falso es porque no se ha podido
// ir a ningún nodo
    MIENTRAS factible Y pos[paso] ≠ N
        seleccionar nodo destino el de mínimo valor desde sol[paso]
        si desde sol[paso] no se llega a ningún nodo factible=FALSO
        SI factible=VERDADERO
            paso++
            sol[paso] = destino
        FINSI
    FINMIENTRAS
    SI factible=VERDADERO
        solución sol con paso pasos
    EN OTRO CASO
      "No se puede encontrar una solución"
    FINSI
```

Problema 4:

Resolver el problema mediante programación dinámica. Describir en detalle la función de recurrencia, los casos base, las tablas, la forma de rellenarlas y la forma de reconstruir la solución.

SOLUCIÓN al ejercicio 4:

Queremos llegar al nodo N en un número de pasos que como mucho será el de niveles del grafo menos 1. Habrá que obtener el camino mínimo en función de soluciones óptima de subproblemas: soluciones llegando a otros nodos en un número menor de pasos. La recurrencia puede ser:

$$C(p,d) = min_{i=1,\ldots,N}\{C(p-1,i) + D(i,d)\}$$

donde se obtiene que el camino óptimo para llegar al nodo d en un máximo de p pasos es el mínimo de la solución óptima de llegar a otro nodo i en un máximo de $p-1$ pasos e ir directamente ($D(i,d)$ es el peso de la arista (i,d)) de i a d.

Usaremos una tabla C con un número de filas igual al número de niveles (la primera fila corresponderá a $p = 0$ y contendrá casos base) y una columna por cada nodo del grafo (N columnas).

Los casos base son: $C(0,1) = 0$ (el coste de llegar al nodo 1 sin ningún paso es 0) y $C(0,i) = \infty$ si $i \neq 1$ (no se puede llegar del nodo 1 a otro nodo sin dar ningún paso).

La tabla para el ejemplo dado, quedaría:

	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	7	3	6	∞	∞	9
2	0	7	3	6	9	7	9
3	0	7	3	6	9	7	8

La solución óptima la tenemos en $C(3,7) = 8$. Vemos que no es necesario almacenar los datos en un array bidimensional, ni recalcular las entradas de todos los nodos para todos los pasos: en cada paso se han encontrado los caminos mínimos de los nodos que se encuentran en ese nivel, por lo que para ellos no es necesario calcular C. Así, la tabla se reduce a un vector que en el paso p contiene los valores de la fila p de la tabla.

Falta determinar cual es el camino de longitud mínima, para eso, por cada cálculo del mínimo de C con la ecuación de recurrencia se guarda el valor con el que se obtiene el mínimo. Si usamos una tabla auxiliar de las mismas dimensiones que C los valores son:

	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	1	1	1	1	3	3	1
3	1	1	1	1	3	3	4

Igual que con C, es suficiente almacenar un array donde se van actualizando los valores para los nodos de los sucesivos niveles. Para recomponer la solución se accede a la última posición de esta tabla auxiliar (la 7 en el ejemplo), el valor en esa posición nos dice desde que nodo se llega, se toma esta posición como nuevo nodo destino y se busca en la tabla desde donde se llega a él, y así hasta llegar al nodo 1. En el ejemplo el camino (obtenido en orden inverso en que se recorre) es 7-4-1.

Problema 5:

Resolver el problema de forma óptima por backtracking. Se deberán utilizar los esquemas vistos en clase. Definir la forma de representar la solución, el tipo de árbol usado, el esquema y programar las funciones genéricas del esquema. Hay que justificar el criterio que se utiliza para intentar eliminar nodos.

SOLUCIÓN al ejercicio 5:

Es un problema de optimización minimizando, y utilizaremos el esquema de backtracking no recursivo para este tipo de problemas. Utilizaremos un array sol:array[0..$niveles$] de $0 \ldots N$, donde la posición 0 se inicializa a 1, que es el nodo del que se sale, y $sol[p] = j$ indica que en el paso p se llega al nodo j. La longitud del camino por el que vamos se almacena en $long$, que se utilizará para eliminar nodos cuando la longitud sea mayor que la de la mejor solución encontrada hasta ese momento.

El número de niveles del árbol coincide con el del grafo, pero no todos los nodos terminarles están al mismo nivel. Un nodo en el árbol que corresponda a un nodo en el nivel i del grafo tendrá hijos correspondientes a los nodos en los niveles del grafo del $i + 1$ en adelante. Son nodos terminales los que corresponden al nodo destino (el N) del grafo.

El esquema será:

$nivel = 1$

$VOA = max$

// max es un valor máximo que puede ser por ejemplo

// la suma de todos los pesos de las aristas
$SOA \leftarrow 0$ // inicialmente no tenemos solución
$sol \leftarrow \{1, 0 \ldots\}$ // se parte del nodo 1
$long = 0$
REPETIR
    generar(nivel,sol,long)
    SI solucion(nivel,sol) Y $long < VOA$
        $VOA = long$
        $SOA \leftarrow sol$
    FINSI
    SI criterio(nivel,sol,long)
        $nivel++$
    EN OTRO CASO
        MIENTRAS $nivel \neq 0$ Y NO mashermanos(nivel,sol)
            retroceder(nivel,sol,long)
        FINMIENTRAS
    FINSI
HASTA $nivel = 0$

Si es el primer descendiente se genera el primer nodo del nivel siguiente en el grafo, y si no se pasa al siguiente nodo:

generar(nivel,sol,long):
    SI $sol[nivel] == 0$
        $sol[nivel]$ = primer nodo en siguiente nivel
    EN OTRO CASO SI
// se quita la longitud de la arista que teníamos
        $long- = D[sol[nivel-1], sol[nivel]]$
        $sol[nivel]++$
    FINSI
// sumamos la longitud de la arista que hemos incluido
    $long+ = D[sol[nivel-1], sol[nivel]]$

Tenemos una solución cuando hemos llegado a N con un camino factible:

solucion(nivel,sol):
    DEVOLVER $sol[nivel] == N$ Y $long < max$

Se cumple el criterio (se sigue hacia abajo en el árbol) cuando: no estamos en un nodo terminal, el camino es factible, y la longitud del camino no excede la de la mejor solución encontrada:

criterio(nivel,sol):
    DEVOLVER $sol[nivel] \neq N$ Y $long < VOA$

Hay más hermanos si no hemos probado hasta el nodo N:

mashermanos(nivel,sol):

DEVOLVER $sol[nivel] \neq N$

Antes de retroceder se actualiza *long* quitando el peso de la arista que se elimina y se deja la solución a su estado inicial:

retroceder(nivel,sol,mov,recorridos):

$long- = D[sol[nivel-1], sol[nivel]]$

$sol[nivel] = 0$

$nivel--$

Problema 6:

Resolver el problema de forma óptima por ramificación y poda. Se deberán utilizar los esquemas vistos en clase. Definir la forma de representar la solución, la forma de calcular las cotas y la estimación de beneficio (y justificar por qué se consideran que esas cotas y estimación pueden ser satisfactorias), así como las estrategias de ramificación y poda.

SOLUCIÓN al ejercicio 6:

Se utiliza el mismo tipo de árbol que en el problema anterior, por lo que cada nodo estará representado por los valores de *nivel*, *sol* y *long*. Además, para almacenar la información del nodo en la lista de nodos vivos hay que almacenar también las cotas (CI y CS) y la estimación del coste (EC). Hablamos de coste en vez de beneficio porque estamos en un problema de minimización.

La CI se puede calcular como la longitud del camino que llevamos generado (variable *long*), que es lo que se usa en el backtracking para podar si $long \geq VOA$. Se podría tomar un valor mayor sumando a *long* la longitud mínima de las aristas del grafo, o de las aristas que salen a partir del nivel del grafo en el que estamos.

La CS se puede obtener haciendo un avance rápido (ejercicio 3) a partir del nodo del grafo en que estamos y sumando el valor a la longitud *long*. En este problema puede que el avance rápido no encuentre solución aunque la haya. En caso de no encontrar solución el avance rápido nos devolvería un valor especial muy alto (el valor max).

Dado que el avance rápido suponemos que nos da una solución cercana a la óptima, podemos tomar $EC = CS$. Pero en este problema el avance rápido puede que no encuentre solución aunque la haya, por lo que puede ser mejor tomar la estimación como media de las dos cotas. Cuál es la mejor decisión en cuanto a la estimación depende de la entrada particular, y no podemos decirlo a priori. Lo único que podemos hacer es tomar decisiones que parezcan razonables (heurísticas). Además, como a partir de un nodo no

sabemos si hay solución, el criterio de poda será $CI(nodo) \geq C$, y la variable C se actualizará solo con valores de nodos solución (no con los valores de las cotas superiores de los nodos recorridos, pues las cotas superiores son de "posibles"soluciones). Cuando la CS corresponda a un valor obtenido por avance rápido sí hemos encontrado solución, por lo que en este caso se puede actualizar el valor de C sin estar ya en un nodo solución.

La estrategia de ramificación será la LC, pues conviene analizar primero los nodos de menor coste estimado. A igualdad de EC podemos usar una estrategia FIFO o LIFO, que no podemos determinar cual es mejor. Si usamos una LC-FIFO habrá que insertar los nodos en la lista de nodos vivos ordenados de menor a mayor EC, y a igualdad de valor un nuevo nodo se incluirá a continuación de los demás nodos que tienen su misma EC.

El esquema de ramificación y poda quedaría:

$LNV = \{NodoRaiz\}$
$C = max$
$SOA \leftarrow 0$ // inicialmente no tenemos solución
MIENTRAS $LNV \neq \emptyset$
  x=extraer(LNV)
  SI $CI(x) \leq C$
    PARA cada nodo del grafo en nivel posterior al del nodo de x
      generar y, hijo de x correspondiente a ese nodo
      SI factible(y)
        SI solucion(y) Y valor(y)$< C$)
          C =valor(y)
          SOA =solucion(y)
        EN OTRO CASO SI CS obtenida con avance rápido
          $C = CS(y)$
        FINSI
      FINSI
    FINPARA
  FINSI
FINMIENTRAS