Algoritmos y Estructuras de Datos II

Ejercicios voraces

19 de junio de 2013
UNC cumple 400 años

Ejercicio 1

Se desea mostrar un párrafo justificado, de forma que sus palabras ocupen cada línea completamente (excepto posiblemente la última). Un párrafo está dado por una secuencia de n palabras de $p_1, p_2, \ldots, p_n$ caracteres respectivamente. Cada línea puede tener hasta L caracteres, y entre dos palabras consecutivas de la misma línea debe haber un espacio del tamaño de al menos un carácter. Es decir que una línea con las palabras i a j tendrá $j - i$ espacios entre palabras y ocupará al menos $p_i + p_{i+1} + \cdots + p_j + (j - i)$ caracteres.

Cuando las palabras que caben en una línea y sus espacios intermedios ocupan menos de L caracteres, el espacio sobrante se distribuye ensanchando todos los espacios entre palabras (a menos que sea la última línea del párrafo).

Ejercicio 1, continuación

Por ejemplo, si $L = 14$ y las primeras palabras del párrafo son *Érase una vez*, para completar la línea falta un carácter ($p_1 + p_2 + p_3 + (3 - 1) = 13$), por lo tanto cada uno de los (dos) espacios de la línea ocuparán el espacio de $1 + 1/2 = 1{,}5$ caracteres.

Escribir un algoritmo voraz que determine, por cada línea que ocupará el párrafo, las palabras que caben en la misma y el tamaño de los espacios con que deben separarse para justificarlo.

Ejercicio 1
Solución

En el práctico.

Ejercicio 2

Consideramos ahora una variante del problema de la moneda, que se distingue del anterior en dos aspectos: por un lado, no nos limitamos a pagar el monto dado de manera exacta, también **está permitido superar** dicho monto. Por el otro, en vez de minimizar el número de monedas, interesa minimizar el **importe realmente pagado** de modo de no superar el monto en más de lo necesario.
Asumimos, entonces, que se cuenta con suficientes monedas de cada una de las denominaciones $d_1, \ldots, d_n$ y un monto m y debemos encontrar el mínimo importe que alcance o supere m con las monedas dadas, utilizando **backtracking**. Para ello,

1. Determine una definición adecuada en palabras de $m(i, j)$.
2. Defina recursivamente $m(i, j)$ discriminando los distintos casos posibles y **explicando y justificando** claramente la fórmula que corresponde en cada uno de ellos.

$m(i, j)$ = mínimo importe no menor a j pagable con monedas de denominación $d_1, \ldots, d_i$.

$$m(i,j) = \begin{cases} 0 & j = 0 \\ \infty & i = 0 \wedge j > 0 \\ \min(m(i-1,j), d_i) & i > 0 \wedge d_i > j > 0 \\ \min(m(i-1,j), d_i + m(i, j-d_i)) & i > 0 \wedge j \geq d_i \end{cases}$$

Ejercicio 3

Para un cierto problema se ha hallado la siguiente definición utilizando backtracking.

$$m(i,j) = \begin{cases} 5 & j = 0 \\ 2 & j > 0 \wedge i = 0 \\ j + m(i, j-1) & i > 0 \wedge j > 0 \wedge j > k-3 \\ \min \begin{cases} 1 + m(i-1, j) \\ m(i, j-i) \\ m(i-1, j+3) \end{cases} & i > 0 \wedge j \geq i \wedge j \leq k-3 \\ \min \begin{cases} 1 + m(i-1, j) \\ m(i, 0) \\ m(i-1, j+3) \end{cases} & c.c. \end{cases}$$

Además la llamada principal es $m(n, k)$.
Transformar esta definición en una iterativa utilizando una tabla de valores calculados, es decir, usando programación dinámica.

```
fun cambio(d:array[1..n] of nat, k: nat) ret r: nat
    var m: array[0..n,0..k] of nat
    for i:= 0 to n do m[i,0]:= 5 od
    for j:= 1 to k do m[0,j]:= 2 od
    for i:= 1 to n do
        for j:= 1 to k do
            if j > k-3 then m[i,j]:= j + m[i,j-1]
            else if j ≥ i then m[i,j]:= min(1+m[i-1,j],m[i,j-i],m[i-1,j+3])
            else m[i,j]:= min(1+m[i-1,j],m[i,0],m[i-1,j+3])
            fi
        od
    od
    r:= m[n,k]
end fun
```

Ejercicio 4

Dado un grafo dirigido cuyos vértices representan los puntos más importantes de la ciudad (esquinas, escuelas, iglesias, museos, campos deportivos, centros comerciales, etc.) y cuyas aristas están rotuladas con el costo en tiempo para desplazarse de un vértice a otro le piden determinar cuál es el vértice donde estacionar la ambulancia de que dispone el municipio.
El objetivo es elegir el vértice cuyo punto más alejado esté más cerca. Es decir, si el vértice elegido es v y el vértice más alejado de v es w, entonces para ningún otro vértice v' puede ocurrir que su vértice más alejado, w' esté a menor distancia de v' que w de v.
¿Cuál de los algoritmos que conoce para el cálculo del camino de costo mínimo utilizaría para resolver este problema? ¿Cómo lo utilizaría? Justificar claramente.

Ejercicio 4
Solución

solución 1 Utilizar Floyd y sobre la matriz:

- calcular el máximo de cada fila,
- comparar los máximos entre sí
- elegir la fila i donde se encuentre el menor de ellos
- colocar la ambulancia en i

solución 2 Ejecutar Dijkstra para cada vértice

- para cada de ellos (v) calcular el costo c_v del destino más alejado
- el v con c_v más pequeño es el vértice donde hay que colocar la ambulancia

orden Comparar los órdenes de estos algoritmos.

Ejercicio 5

Recorrer en pre-order, pos-order y bfs.