Algoritmi e Strutture Dati - Prova d'esame
27/05/11

Esercizio 1

L'esercizio si risolve semplicemente con la seguente procedura ricorsiva, la cui chiamata iniziale è maxdepth(T). Essendo una visita, la complessità è ovviamente $O(n)$. Per quanto riguarda la correttezza, è semplice notare che raggiungeremo solo i nodi che fanno parte di cammini crescenti, senza proseguire (e restituendo -1 quando si incontrano il primo valore non monotono crescente).

```
int maxdepth(TREE T)
    if T ≠ nil and (T.parent() == nil or T.parent().value < T.value) then
        return 1 + max(maxdepth(T.left()), maxdepth(T.right()))
    else
        return −1
```

Esercizio 2

Sia $DP[i][j]$ il minimo valore che può essere ottenuto dalla sottoespressione $c_i O_i c_{i+1} \ldots c_{j-1} O_{j-1} c_j$; il problema originale consiste nel calcolare $DP[1][n]$. La funzione di ricorrenza che definisce la sottostruttura ottima del problema è la seguente:

$$DP[i][j] = \begin{cases} \min_{i \le k < j} \{DP[i][k] O_k DP[k+1][j]\} & j > i \\ c_i & j = i \end{cases}$$

Questa ricorrenza porta ad un programma molto simile a quello della moltiplicazione fra catene di matrici:

```
parentesizzazione(int[] C, OPERATOR[] O, int[][] M, int[][] S, int n)
    int i, j, h, k, t
    for i = 1 to n do
        DP[i][i] = C[i]
    for h = 2 to n do
        for i = 1 to n − h + 1 do
            j = i + h − 1
            DP[i][j] = +∞
            for k = i to j − 1 do
                t = DP[i][k]O[k]DP[k + 1][j]
                if t < DP[i][j] then
                    DP[i][j] = t
                    S[i][j] = k
```

Una procedura che stampa l'espressione utilizzando i valori memorizzati in S è la seguente:

```
stampaPar(int[] C, OPERATOR[] O, int[][] S, int i, int j)
    if i == j then
        print C[i]
    else
        print "("; stampaPar(S, i, S[i][j]); print O[S[i][j]]; stampaPar(S, S[i][j] + 1, j); print ")"
```

Esercizio 3

È possibile dimostrare per induzione che se $V[n] - V[1] \geq n$, allora esiste un double gap nel sottovettore $V[1 \dots k]$.

- *Caso base*. Per $n = 2$, l'ipotesi è che $V[2] - V[1] \geq 2$, che è ovviamente un double gap.
- *Passo induttivo*. Supponiamo che sia stato dimostrato che $\forall n' < n : V[n'] - V[1] \geq n'$, allora esiste un double gap nel sottovettore $V[1 \dots n']$. Vogliamo dimostrare che la proprietà è vera anche per n.

  Si consideri $V[n-1]$; possono darsi due casi. Se $V[n] - V[n-1] \geq 2$, allora esiste un double gap agli indici $(n-1, n)$. Altrimenti, se $V[n] - V[n-1] \leq 1$, si ottiene che

  $$V[n-1] - V[1] \geq V[n] - 1 - V[1] \geq n - 1$$

  (in quanto $V[n-1] \geq V[n] - 1$ e $V[n] - V[1] \geq n$). Per induzione, esiste quindi un double gap in $V[1 \dots n-1]$.

Questa dimostrazione tuttavia non aiuta a trovare un algoritmo che funzioni in tempo $O(\log n)$, in quanto suggerisce di controllare tutte le coppie consecutive in tempo $O(n)$. Ovviamente, dovremmo utilizzare un algoritmo basato sulla ricerca binaria.
Sia $V[i \dots j]$ un sottovettore tale per cui $V[j] - V[i] \geq j - i + 1$, dove $j - i + 1$ è la lunghezza del sottovettore. Per la dimostrazione di cui sopra, sappiamo che esiste un double gap all'interno del sottovettore (se vi confonde il fatto che stiamo parlando di sottovettori e non di vettori interi, copiate mentalmente i valori in $V[i \dots j]$ in un vettore $V'[1 \dots n]$, con $n = j - i + 1$).
Sia $m = \lfloor (i+j)/2 \rfloor$ e si consideri $V[m]$. Possono darsi due casi:

- $V[m] - V[i] \geq m - i + 1$, dove $m - i + 1$ è la lunghezza del sottovettore $V[i \dots m]$; per la dimostrazione di cui sopra, esiste in double-gap in $V[i \dots m]$ e possiamo restringerci a controllare questo sottovettore.
- $V[j] - V[m] \geq j - m + 1$, dove $j - m + 1$ è la lunghezza del sottovettore $V[m \dots j]$; per la dimostrazione di cui sopra, esiste in double-gap in $V[m \dots j]$ e possiamo restringerci a controllare questo sottovettore.

Dobbiamo tuttavia dimostrare che almeno una delle due condizioni sia vera; supponendo per assurdo che $V[m] - V[i] \leq m - i$ e che $V[j] - V[m] \leq j - m$, allora sommandoli assieme abbiamo che $V[j] - V[i] \leq j - i$, che è assurdo rispetto all'ipotesi che $V[j] - V[j] \geq j - i + 1$.
Questo suggerisce il seguente algoritmo, il cui caso base avviene quando $j = i + 1$.

```
int doublegap(int[] V, int i, int j)
    if j == i + 1 then
        return i
    int m = ⌊(i + j)/2⌋
    if V[m] − V[i] ≥ m − i + 1 then
        return doublegap(V, i, m)
    else
        return doublegap(V, m, j)
```

Essendo una ricerca binaria, la complessità è $O(\log n)$.

Esercizio 4

Si considerino due job, uno con guadagno 2 e deadline 2 e uno con guadagno 1 e deadline 1. Eseguendo prima il primo job, come da algoritmo, si può eseguire solo quello e il guadagno è 2; eseguendo invece prima il secondo e poi il primo, si ottiene un guadagno di 3. Questo dimostra che l'algoritmo greedy non è corretto.