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 $\mathsf{maxdepth}(T)$. 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).

```
integer 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 $M[i, j]$ 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 $M[1, n]$. La funzione di ricorrenza che definisce la sottostruttura ottima del problema è la seguente:

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

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

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

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

```
stampaPar(integer[] C, OPERATOR[] O, integer[][] S, integer i, integer 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 $V[1 \ldots k]$.

- *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 $V[1 \ldots n']$. 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 $V[1 \ldots n-1]$.

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 $V[i \ldots j]$ 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 $V[i \ldots j]$ in un vettore $V'[1 \ldots n]$, 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 $V[i \ldots m]$; per la dimostrazione di cui sopra, esiste in double-gap in $V[i \ldots m]$ e possiamo restringerci a controllare questo sottovettore.

- $V[j] - V[m] \geq j - m + 1$, dove $j - m + 1$ è la lunghezza del sottovettore $V[m \ldots j]$; per la dimostrazione di cui sopra, esiste in double-gap in $V[m \ldots j]$ 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 $j = i + 1$.

```
doublegap(integer[] V, integer i, integer j)
  if j = i + 1 then
    return i
  integer 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.
Si consideri invece un algoritmo che ordina i job per valori crescenti della deadline.

```
maxgain(integer[] D, integer[] G, integer n )
  { ordina i vettori D, G per deadline crescente; ovvero, ∀i, j, 1 ≤ i < j ≤ n : D[i] < D[j] }
  integer t ← 0
  for i ← 1 to n do
    if t + 1 ≤ D[i] then
      print i
      t ← t + 1
```

Vogliamo dimostrare che scegliere il primo job (in ordine crescente di deadline) la cui deadline sia rispettata è una scelta greedy corretta.
Sia $S = s_1 s_2 \ldots s_k$ una sequenza di job che massimizza il guadagno. Vogliamo dimostrare che esiste una sequenza $S' = s'_1 \ldots s'_k$ con guadagno uguale ad S e in cui s'_1 è il job la cui deadline è minima.

Se s_1 è il job con deadline minima, la dimostrazione è già completa. Supponiamo invece che questo job sia in posizione s_m, $m > 1$. Si consideri la sequenza $S' = s_m s_2 s_3 \ldots s_{m-1} s_1 s_{m+1} \ldots s_k$, in cui abbiamo scambiato s_1 con s_m. Tutti i job con indici diversi da s_m e s_1 vengono eseguiti allo stesso istante temporale, quindi non hanno problemi di deadline. Il job s_m ha deadline $D[s_m] \geq m$, e quindi eseguirlo come primo job non comporta problemi; il job s_1 ha deadline $D[s_1] > D[s_m] \geq m$, in quanto s_m ha deadline minima e quindi eseguirlo all'istante m non comporta problemi. In altre parole, S' è una sequenza con guadagno uguale ad S e quindi ottima, che rispetta la scelta greedy.