Algoritmi e Strutture Dati - 16/06/14

Esercizio 1

Utilizzando il master theorem, è facile vedere che $T(n) = \Theta(\sqrt[3]{n}\log n)$. Dimostriamo che $T(n) = O(\sqrt[3]{n}\log n)$ per sostituzione. Coinvolgendo il logaritmo, il caso base è fra quelli problematici:

$$T(1) = 1 \not\leq c\log 1 = 0$$

Per questo motivo, consideriamo i valori i compresi fra 2 e 15, estremi inclusi; $\lfloor i/8 \rfloor$ in questo caso è pari a 0 o 1; scriviamo quindi

$$T(i) = 2T(\lfloor i/8 \rfloor) + \sqrt[3]{i} = 2 \cdot 1 + \sqrt[3]{i} \leq c\sqrt[3]{i}\log i \qquad \forall i : 2 \leq i \leq 15$$

da cui si ottiene:

$$c \geq \frac{2 + \sqrt[3]{i}}{\sqrt[3]{i}\log i} \qquad \forall i : 2 \leq i \leq 15$$

Per $i = 16$, $\lfloor i/8 \rfloor$ è pari a 2 e rientra nei casi base già risolti. Possiamo quindi fermarci a 15.
Nel passo induttivo, dobbiamo dimostrare che $T(n) \leq c\sqrt[3]{n}\log n$ e supponiamo che la relazione $T(n') \leq c\sqrt[3]{n'}\log n'$ sia già stata dimostrata per $2 \leq n' < n$.

$$\begin{aligned} T(n) &\leq 2c\sqrt[3]{\lfloor n/8 \rfloor}\log\lfloor n/8 \rfloor + \sqrt[3]{n} \\ &\leq 2c\sqrt[3]{n/8}\log n/8 + \sqrt[3]{n} \\ &= c\sqrt[3]{n}\log n/8 + \sqrt[3]{n} \\ &= c\sqrt[3]{n}(\log n - \log 8) + \sqrt[3]{n} \\ &= c\sqrt[3]{n}\log n - 3c\sqrt[3]{n} + \sqrt[3]{n} \leq c\sqrt[3]{n}\log n \end{aligned}$$

L'ultima disequazione è soddisfatta se $c \geq 1/3$. Poiché questa disequazione per c e tutte quelle derivanti dal caso base sono di tipo $\geq$, è sufficiente prendere il valore più alto fra questi valori come valore per c.

Esercizio 2

Per risolvere questo esercizio, è sufficiente modificare l'algoritmo che visita le componenti connesse, in modo da verificare, per ogni componente, se la componente contiene cicli oppure no. Bisogna prestare attenzione al fatto che tutti gli archi sono sdoppiati, quindi quando si visita un nodo v per la prima volta arrivando dal nodo u bisogna evitare di considerare l'arco $[v, u]$. La complessità è $O(m+n)$.

Esercizio 3

Esercizio 3.1 Si consideri un problema con due file F_1, F_2, dove il file F_1 ha lunghezza 100 e probabilità di accesso 0.51, mentre F_2 ha lunghezza 1 e probabilità di accesso 0.49.

- Ordinando F_1 prima di F_2, come richiesto dall'algoritmo, si ottiene: $0.51 \cdot 100 + 0.49 \cdot 101 = 100.49$
- Ordinando F_2 prima di F_1, diversamente dell'algoritmo, si ottiene: $0.49 \cdot 1 + 0.51 \cdot 101 = 52$.

Quindi l'ordinamento proposto non è ottimale.

Esercizio 3.2 Si consideri un problema con due file F_1, F_2, dove il file F_1 ha lunghezza 50 e probabilità di accesso 0.25, mentre il file F_2 ha lunghezza 100 e probabilità di accesso 0.75.

- Ordinando F_1 prima di F_2, come richiesto dall'algoritmo, si ottiene: $0.25 \cdot 50 + 0.75 \cdot 150 = 125$
- Ordinando F_2 prima di F_1, diversamente dell'algoritmo, si ottiene: $0.75 \cdot 100 + 0.25 \cdot 150 = 112.5$.

Quindi l'ordinamento proposto non è ottimale.

```
integer[] countTrees(GRAPH G)
    boolean[] visited ← new boolean[1 ... G.n]
    foreach u ∈ G.V() do visited[u] ← false

    integer counter ← 0
    foreach u ∈ G.V() do
        if not visited[u] then
            if countTreesDFS(G, u, nil, visited) then
                counter ← counter + 1
    return counter

    countTreesDFS(GRAPH G, NODE u, NODEp, boolean[] visited)
        boolean isAcyclic ← true
        visited[u] ← true
        foreach v ∈ G.adj(u) do
            if v ≠ p and not visited[v] then
                if not countTreesDFS(G, conta, v, u, visited) then
                    isAcyclic ← false
            else
                isAcyclic ← false
        return isAcyclic
```

Esercizio 3.3 Per risolvere il problema in maniera greedy, è sufficiente ordinare i file per rapporto $L[i]/P[i]$ crescente. Dobbiamo quindi dimostrare che tale ordinamento gode della scelta greedy.
Per prima cosa, ragioniamo sulla sottostruttura ottima. Si consideri un ordinamento ottimo $f_1, f_2, \ldots, f_n$ (permutazione degli indici $1, \ldots, n$), e si consideri il sottoproblema dato dai file $f_2, f_3, \ldots f_n$. Vogliamo dimostrare che l'ordinamento $f_2, f_3, \ldots, f_n$ è ottimo per tali file, e non esiste una permutazione che comporta un costo minore. Per assurdo, sia $g_2, g_3, \ldots, g_n$ un tale permutazione; ma allora, potremmo ottenere una permutazione $f_1, g_2, g_3, \ldots, g_n$ che ha costo inferiore a $f_1, f_2, f_3, \ldots, f_n$, in quanto:

- il costo associato al file f_1 è pari a $L[f_1] \cdot P[f_1]$ in entrambi i casi;
- una volta messo in testa il file f_1, la sua lunghezza viene aggiunta al tempo di attesa di tutti i file seguenti, indipendentemente dal loro ordine

A questo punto, dimostriamo la scelta greedy. Si consideri un ordinamento ottimo e sia m l'indice di un file con il più basso rapporto lunghezza/probabilità. In altre parole,

$$\frac{L[m]}{P[m]} \leq \frac{L[i]}{M[i]}, \qquad \forall i, 1 \leq i \leq n \tag{1}$$

Vogliamo dimostrare che è possibile costruire un altro ordinamento ottimo in cui il file m si trova in prima posizione. Ovviamente, se $m = 1$ la dimostrazione è completa. Supponiamo quindi che $m > 1$ e costruiamo tale soluzione passo-passo.
Si scambino il file m–1 e il file m, ovvero si consideri l'ordinamento $1, \ldots, m$–$2, m, m$–$1, m+1, \ldots n$; vogliamo dimostrare che tale ordinamento ha tempo atteso totale di accesso minore o uguale all'ordinamento $1, \ldots, m$–$2, m$–$1, m, m+1, \ldots, n$. Tutti i file diversi da m–1 e m restano nella stessa posizione; il loro contributo al tempo atteso totale di accesso resta invariato. Dobbiamo quindi dimostrare che

$$(T[m\text{–}2] + L[m]) \cdot P[m] + (T[m\text{–}2] + L[m] + L[m\text{–}1]) \cdot P[m\text{–}1] \leq (T[m\text{–}2] + L[m\text{–}1]) \cdot P[m\text{–}1] + (T[m\text{–}2] + L[m\text{–}1] + L[m]) \cdot P[m]$$

dove

- $(T[m\text{–}2] + L[m]) \cdot P[m]$ è il costo del file m quando si trova dopo i file $1 \ldots m$–2 e prima del file m–1;
- $(T[m\text{–}2] + L[m] + L[m\text{–}1]) \cdot P[m\text{–}1]$ è il costo del file m–1 quando si trova dopo il file $1 \ldots m$–2 e dopo il file m;
- $(T[m\text{–}2] + L[m\text{–}1]) \cdot P[m\text{–}1]$ è il costo del file m–1 quando si trova dopo i file $1 \ldots m$–2 e prima del file m;
- $(T[m\text{–}2] + L[m\text{–}1] + L[m]) \cdot P[m]$ è il costo del file m quando si trova dopo il file $1 \ldots m$–2 e dopo il file m–1;

Svolgendo le moltiplicazioni, si ottiene:

$$T[m\text{–}2] \cdot P[m] + L[m] \cdot P[m] + T[m\text{–}2] \cdot P[m\text{–}1] + L[m] \cdot P[m\text{–}1] + L[m\text{–}1] \cdot P[m\text{–}1] \leq$$
$$T[m\text{–}2] \cdot P[m\text{–}1] + L[m\text{–}1] \cdot P[m\text{–}1] + T[m\text{–}2] \cdot P[m] + L[m\text{–}1] \cdot P[m] + L[m] \cdot P[m]$$

Semplificando i termini uguali da entrambi lati si ottiene:

$$\cancel{T[m\text{–}2] \cdot P[m]} + \cancel{L[m] \cdot P[m]} + \cancel{T[m\text{–}2] \cdot P[m\text{–}1]} + L[m] \cdot P[m\text{–}1] + \cancel{L[m\text{–}1] \cdot P[m\text{–}1]} \leq$$
$$\cancel{T[m\text{–}2] \cdot P[m\text{–}1]} + \cancel{L[m\text{–}1] \cdot P[m\text{–}1]} + \cancel{T[m\text{–}2] \cdot P[m]} + L[m\text{–}1] \cdot P[m] + \cancel{L[m] \cdot P[m]}$$

da cui si ottiene:

$$L[m] \cdot P[m\text{–}1] \leq L[m\text{–}1] \cdot P[m] \quad \Leftrightarrow \quad \frac{L[m]}{P[m]} \leq \frac{L[m\text{–}1]}{P[m\text{–}1]}$$

che è una condizione vera per la disequazione 1.
Con questa operazione, abbiamo dimostrato che scambiando m con il suo predecessore, è possibile ottenere un nuovo ordinamento che ha costo inferiore o uguale all'ordinamento corrente; ma essendo l'ordinamento ottimo, anche il nuovo ordinamento è ottimo. Ripetendo questa operazione più volte, fino a portare il file con indice m in prima posizione, si ottiene un ordinamento ottimo che rispetta la nostra scelta greedy.
Il costo dell'algoritmo è dominato dall'ordinamento, ed è quindi pari a $O(n \log n)$.

Esercizio 4

Il tempo necessario a risolvere il problema a partire dal distributore i con un'autonomia residua a è rappresentato da $T[i, a]$. Il problema iniziale è rappresentato da $T[1, 0]$. Il caso base è rappresentato da $T[n, a] = 0, \forall a$. Altrimenti, al distributore i-esimo si considerano tutti i valori di ricarica g compresi fra 0 (niente ricarica) e $100 - a$ (ricarica completa a partire dall'autonomia residua a) che sono sufficienti ad arrivare al prossimo distributore ($a + g \geq d[i + 1]$).
Il tempo necessario per risolvere il problema può essere calcolato ricorsivamente in questo modo:

$$T[i, a] = \begin{cases} 0 & i = n \\ \min\{T[i + 1, a + g] + g \cdot t[\iota] : 0 \leq g \leq 100 - a \ \wedge \ a + g \geq d[i + 1]\} & i < n \end{cases}$$

E' possibile risolvere l'algoritmo tramite programmazione dinamica nel modo seguente:

```
mintime(integer[] t, integer[] d, integer n)
    integer[][] T ← newinteger[1...n][0...100]
    for a ← 0 → 100 do
        T[n, a] ← 0
    for i ← n − 1 downto 1 do
        for a ← 0 to 100 do
            T[i, a] ← +∞
            for g ← 0 to 100 − a do
                integer t ← T[i + 1, a + g] + g · t[i]
                if a + g ≥ d[i] and t ≤ T[i, a] then
                    T[i, a] ← t
    return T[1, 0]
```

Il ciclo sulla variabile i viene ripetuto $n - 1$ volte; il costo dell'esecuzione dei due cicli su a e g è pari ad A^2. Il costo è quindi pari a $O(nA^2)$.