Algoritmi e Strutture Dati - 10/01/2017

Esercizio 1

Non sono necessarie particolari tecniche per risolvere questo esercizio; è sufficiente fare una scansione di costo lineare $\Theta(n)$.

```
int count01(int [ ] V, int n)
    int count_0 = 0
    int count_1 = 0
    int largest = 0
    for i = 1 to n do
        if V[i] == 0 then
            if count_1 > 0 then
                count_0 = 0
                count_1 = 0
            count_0 = count_0 + 1
        else
            count_1 = count_1 + 1
            largest = max(largest, 2 · min(count_0, count_1))
    return largest
```

Esercizio 2

Sia $DP[i][j]$ la più lunga supersequenza comune contenuta nei prefissi $P(i)$ e $P(j)$, calcolata tramite la seguente espressione ricorsiva:

$$DP[i][j] = \begin{cases} i & \text{se } i \geq 0 \wedge j = 0 \\ j & \text{se } i = 0 \wedge j \geq 0 \\ DP[i-1][j-1] + 1 & \text{se } i > 0 \wedge j > 0 \wedge p_i = t_j \\ \min\{DP[i-1][j], DP[i][j-1]\} + 1 & \text{se } i > 0 \wedge j > 0 \wedge p_i \neq t_j \end{cases}$$

Nel caso una delle stringhe sia vuota, tutti i caratteri dell'altra stringa devono essere presenti nella supersequenza, da cui i due casi base. Se gli ultimi caratteri delle due stringhe sono uguali, si considererà il sottoproblema in cui viene rimosso l'ultimo caratteri di entrambi e si aggiunge $+1$ per contare questo carattere. Nel caso siano diversi, si prenderanno i due sottoproblemi in cui si rimuove l'ultimo carattere di una delle stringhe e si aggiunge $+1$ per contare questo carattere, prendendo il valore più piccolo fra i due.
Un algoritmo basato sulla programmazione dinamica che risolve il problema è il seguente:

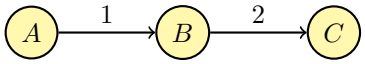
```
int scs(ITEM[ ] P, ITEM[ ] T, int n, int m)
    int[ ][ ] D = new int[0 . . . n][0 . . . m]
    for i = 0 to n do
        DP[i][0] = i
    for j = 0 to m do
        DP[0][j] = j
    for i = 1 to n do
        for j = 1 to m do
            if P[i] == T[j] then
                DP[i][j] = DP[i − 1][j − 1] + 1
            else
                DP[i][j] = min(DP[i − 1][j], DP[i][j − 1]) + 1
    return DP[n][m]
```

La complessità dell'algoritmo è pari a $\Theta(nm)$.

Esercizio 3

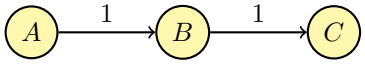
Parte 1

L'arco (A, B) è upward-critical



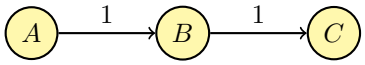
Parte 2

Non esistono archi upward-critical



Parte 3

Tutti gli archi sono downward-critical.



Parte 4

Nella rete seguente, abbassare la capacità dei due archi non cambia il flusso massimo che resta 0.

Tuttavia, assumendo che il flusso massimo sia positivo (assunzione standard nel campo delle reti di flusso), si può dimostrare che non esistono reti che non contengono archi downward-critical. Per assurdo, supponiamo che esista una rete in cui non esiste un arco downward-critical. Quindi, diminuendo di uno la capacità di un qualsiasi arco, il flusso massimo rimane inalterato. Quindi, nessuno degli archi è sfruttato al massimo della capacità. Quindi, nelle rete residua del flusso tutti gli archi della rete originale sono ancora presenti. Poichè il flusso è positivo, esiste un cammino da sorgente a pozzo nel grafo originale. Tale cammino è presente anche nella rete residua e quindi è possibile trovare un cammino aumentante. Quindi il flusso considerato non è massimo.

Esercizio 4

Ordinamento lunghezza crescente L'algoritmo non è corretto. Un possibile controesempio è il seguente

File	Lunghezza	Probabilità
1	1	0.1
2	2	0.9

Ordinati per lunghezza crescente (File 1, File 2), si ottiene un costo pari a:

$$1 \cdot 0.1 + (1 + 2) \cdot 0.9 = 0.1 + 2.7 = 2.8$$

mentre l'ordine corretto è (File 2, File 1):

$$2 \cdot 0.9 + (2 + 1) \cdot 0.1 = 1.8 + 0.3 = 2.1$$

Ordinamento probabilità decrescente L'algoritmo non è corretto. Un possibile controesempio è il seguente

File	Lunghezza	Probabilità
1	10	0.6
2	1	0.4

Ordinati per probabilità decrescente (File 1, File 2), si ottiene un costo pari a:

$$10 \cdot 0.6 + (10 + 1) \cdot 0.4 = 6 + 4.4 = 10.4$$

mentre l'ordine corretto è (File 2, File 1):

$$1 \cdot 0.4 + (1 + 10) \cdot 0.6 = 0.4 + 6.6 = 7.0$$

Soluzione La soluzione corretta è quella di ordinare i file per rapporto lunghezza / frequenza ℓ_i/f_i crescente.
Per assurdo, sia π una permutazione ottima che non rispetta l'ordinamento per rapporto lunghezza/frequenza crescente. Allora esistono due file consecutivi nella permutazione $a = \pi(i)$ e $b = \pi(i+1)$ che non rispettano l'ordine crescente, ovvero $\frac{\ell_a}{f_a} > \frac{\ell_b}{f_b}$. Consideriamo un'altra permutazione in cui i file a e b vengono scambiati. Il costo per accedere ad a aumenta di ℓ_b, mentre il costo per accedere a b diminuisce di ℓ_a. I costi associati a tutti gli altri file sono inalterati. Il costo totale cambia quindi di $f_a\ell_b - f_b\ell_a$, un valore che deve essere maggiore o uguale a 0 perchè π è ottimale. Dalla disequazione si ottiene $\frac{\ell_a}{f_a} \leq \frac{\ell_b}{f_b}$, il che contraddice l'assunzione che a e b non siano ordinati correttamente.