Algoritmi e Strutture Dati - 26/01/16

Esercizio 1

È possibile stimare la prima ricorrenza con $T(n) = \Theta(n)$; è facile vedere che $T(n) = \Omega(n)$, visto che la parte non ricorsiva ha complessità $\Omega(n)$. Dimostriamo che $T(n) = O(n)$; dobbiamo quindi dimostrare che $\exists c > 0, \exists m \geq 0 : T(n) \leq cn, \forall n \geq m$. Procediamo per induzione:

- Caso base: Per $n = 1$, $T(1) = 1 \leq c$, ovver $c \geq 1$;
- Ipotesi induttiva: $T(n') \leq cn', \forall n' < n$;
- Passo induttivo:

$$\begin{aligned}
T(n) &= \frac{1}{2}(T(n-1) + T(3n/4)) + n \\
&\leq \frac{1}{2}(cn - c + \frac{3}{4}cn) + n \\
&= \frac{7}{8}cn - c/2 + n \\
&\leq \frac{7}{8}cn + n \\
&\leq cn
\end{aligned}$$

  L'ultima disequazione è vera per $c \geq 8$.

Abbiamo così dimostrato che $T(n) = \Theta(n)$

Anche la seconda ricorrenza è lineare; per quanto riguarda il limite inferiore $T(n) = \Omega(n)$, dimostriamo che $\exists c > 0, \exists m \geq 0 : T(n) \geq cn, \forall n \geq m$. Procediamo per induzione:

- Caso base: Per $n = 1$, $T(1) = 1 \geq c$, ovvero $c \leq 1$;
- Ipotesi induttiva: $T(n') \geq cn', \forall n' < n$;
- Passo induttivo:

$$\begin{aligned}
T(n) &= T(\frac{1}{2}n) + T(\frac{1}{4}n) + T(\frac{1}{6}n) + T(\frac{1}{12}n) + 1 \\
&\geq \frac{1}{2}cn + \frac{1}{4}cn + \frac{1}{6}cn + \frac{1}{12}cn + 1 \\
&= cn + 1 \\
&\geq cn
\end{aligned}$$

  L'ultima disequazione è vera per qualunque valore di c; abbiamo così dimostrato che $T(n) = \Omega(n)$.

È facile vedere che provare a dimostrare che $T(n) \leq cn$ condurrebbe a una disequazione $cn + 1 \leq cn$, ovviamente falsa. Proviamo quindi a dimostrare che $T(n) \leq cn - b$.

- Caso base: Per $n = 1$, $T(1) = 1 \leq c - b$, ovvero $c \geq b + 1$;
- Ipotesi induttiva: $T(n') \leq cn' - b, \forall n' < n$;
- Passo induttivo:

$$\begin{aligned}
T(n) &= T(\frac{1}{2}n) + T(\frac{1}{4}n) + T(\frac{1}{6}n) + T(\frac{1}{12}n) + 1 \\
&\leq \frac{1}{2}cn - b + \frac{1}{4}cn - b + \frac{1}{6}cn - b + \frac{1}{12}cn - b + 1 \\
&= cn - 4b + 1 \\
&\leq cn - b
\end{aligned}$$

  L'ultima disequazione è vera per $b \geq 1/3$ e qualunque valore di c; abbiamo quindi dimostrato che $T(n) = O(n)$.

Esercizio 2

È sufficiente utilizzare il grafo (diretto, con due archi per ogni coppia di nodi) che rappresenta una mappa come rete di flusso, utilizzando la casa come sorgente e la scuola come pozzo, e costruendo quindi una funzione di capacità tale per cui $c(u, v) = 1$ per ogni $(u, v) \in E$. A questo punto, applicando l'algoritmo per il calcolo del flusso, si ottiene il numero k di cammini "edge-disjoint" esistenti nel grafo; se $k \geq 2$, i figli potranno evitarsi lungo il percorso.
Si noti che il fatto che il grafo originale non sia orientato non ha influenza sul risultato; se esiste un flusso di valore k, esiste un taglio di valore k che separa la sorgente dal pozzo. Ovvero esistono k archi distinti che vanno dalla partizione del taglio contenente la sorgente alla partizione del taglio contenente il pozzo.
Per quanto riguarda la complessità, poichè ci possono essere al più $n-1$ cammini distinti, la complessità è $O(mn)$; ma è sempre possibile limitare il numero di visite effettuate a 2, in modo da limitare la complessità a $O(m+n)$.

Esercizio 3

Sia DP una matrice booleana $(m+1) \cdot (n+1)$, tale per cui $DP[i][j]$ è **true** se e solo se il prefisso $Z(i+j)$ è mescolato a partire dai prefissi $X(i)$ e $Y(j)$. Una definizione ricorsiva di $DP[i][j]$ è la seguente:

$$DP[i][j] = \begin{cases} \textbf{true} & i = 0 \land j = 0 \\ Y[j] = Z[i+j] \land DP[i][j-1] & i = 0 \land j > 0 \\ X[i] = Z[i+j] \land DP[i-1][j] & i > 0 \land j = 0 \\ (X[i] = Z[i+j] \land DP[i-1][j]) \lor (Y[j] = Z[i+j] \land DP[i][j-1]) & i > 0 \land j > 0 \end{cases}$$

La funzione assume che $|Z| = |X| + |Y|$; è semplice intercettare il caso in cui questo non sia vero e restituire **false**.
Nel caso $i = 0$ e $j = 0$, stiamo considerando prefissi vuoti di X e Y; visto che stiamo considerando il prefisso $Z(0)$, anch'esso vuoto, $DP[0][0] = \textbf{true}$. Se $i = 0$ e $j > 0$, dobbiamo verificare che il carattere $Z[i+j] = Z[0+j] = Z[j]$ sia uguale ad $Y[j]$; dopodichè, arretriamo di un carattere in Y e verifichiamo ricorsivamente che la proprietà sia valida anche in $DP[i][j-1]$. Nel caso $i > 0$ e $j = 0$, la condizione opera sulla stringa X e sull'indice i, simmetricamente alla precedente. Nel caso $i > 0$ e $j > 0$, dobbiamo mettere in **or** le condizioni precedenti.
Questo dà origine al codice seguente:

```
boolean isShuffle(ITEM[] X, ITEM[] Y, ITEM[] Z, int n, int m)
    boolean[][] DP = new boolean[0...n][0...m]
    DP[0][0] = true
    for j = 1 to m do
        DP[0][j] = Y[j] = Z[j] and B[0, j-1]
    for i = 1 to n do
        DP[i][0] = X[i] = Z[i] and B[i-1, 0]
    for i = 1 to n do
        for j = 1 to m do
            DP[i][j] = (X[i] = Z[i+j] and DP[i-1][j]) or (Y[j] = Z[i+j] and DP[i][j-1])
    return DP[n][m]
```

La complessità è ovviamente $\Theta(nm)$.

Esercizio 4

Per risolvere la prima parte del problema, è importante notare che un approccio greedy che cerca di utilizzare il quadrato più grande non funziona. Ad esempio, $18 = 3^2 + 3^2$ (due quadrati), ma utilizzando 4^2 si ottiene $18 = 4^2 + 1^2 + 1^2$ (3 quadrati). Ci affidiamo quindi alla programmazione dinamica.
Sia $DP[n]$ il minimo numero di quadrati necessari per esprimere n. Ovviamente il caso base è $DP[0] = 0$; per quanto riguarda la parte ricorsiva, consideriamo tutti i valori t tali per cui $1 \leq t^2 \leq n$; consideriamo quindi tutti i sottoproblemi $DP[n-t^2]$ e scegliamo fra questi quello che richiede il minor numero di quadrati, aggiungendo 1 per il quadrato considerato:

$$DP[n] = \begin{cases} 0 & n = 0 \\ \min_{1 \leq t \leq \lfloor \sqrt{n} \rfloor} \{DP[n-t^2] + 1\} & n > 1 \end{cases}$$

È semplice scrivere questo codice tramite programmazione dinamica:

```
int squareSum(int n)
    int[] DP = new int[0...n]
    DP[0] = 0
    DP[1] = 1
    for i = 2 to n do
        DP[i] = +∞
        for t = 1 to ⌊√i⌋ do
            DP[i] = min(DP[i], DP[i − t²] + 1)
    return DP[n]
```

Questo codice ha complessità $\Theta(n\sqrt{n})$. È interessante notare che il numero di quadrati necessari è sempre inferiore o uguale a 4, come è stato dimostrato da Lagrange (1798).
Per risolvere la seconda parte del problema, dobbiamo invece utilizzare la tecnica del backtrack. Una versione semplice per risolvere il problema potrebbe scegliere tutti i possibili valori di t tali per cui $1 \leq t^2 \leq n$, e provare ricorsivamente tutte le possibilità:

```
psRec(int n, int[] S, int i)
    if n == 0 then
        print S[1...i − 1]
    else
        for t = 1 to ⌊√n⌋ do
            s[i] = t
            psRec(n − t², S, i + 1)
```

```
printSquare(int n)
    int[] S = new int[1...n]
    psRec(n, S, 1)
```

Il vettore S delle scelte ha dimensione n (somma di tutti 1). La complessità è superpolinomiale.
L'algoritmo appena visto, tuttavia, stampa anche tutte le permutazioni dei quadrati. È accettabile per il compito, ma una soluzione migliore evita le permutazioni imponendo un ordine ai valori che vengono sommati.

```
psRec(int n, int[] S, int i, int limit)
    if n == 0 then
        print S[1...i − 1]
    else
        for t = 1 to min(⌊√n⌋, limit) do
            s[i] ← t
            printSquare(n − t², S, i + 1, t)
```

```
printSquare(int n)
    int[] S = new int[1...n]
    psRec(n, S, 1, ⌊√n⌋)
```

È stato aggiunto il parametro *limit*, che indica il valore massimo che può essere utilizzato per formare un quadrato; in questo modo, una stringa che inizia con 1^2 può essere seguita solo da valori 1^2, una stringa che inizia con 2^2 può essere seguita solo da valori $2^2, 1^2$, etc.
Anche in questo caso è possibile stimare la complessità come superpolinomiale.