Algoritmi e Strutture Dati - 02/02/15

Esercizio 1

È possibile osservare che $T(n) = \Omega(n^2)$, visto il termine n^2 che compare nell'equazione di ricorrenza. Verifichiamo se è anche $O(n^2)$, nel qual caso il limite è stretto e abbiamo terminato.

- Caso base: $T(1) = 1 \leq c \cdot 1^2 \Rightarrow c \geq 1$
- Ipotesi induttiva: $\forall k < n : T(k) \leq ck^2$
- Passo induttivo:

$$\begin{aligned}
T(n) &= T(\lfloor n/\sqrt{2} \rfloor) + T(\lfloor n/2 \rfloor) + T(\lfloor n/2\sqrt{2} \rfloor) + T(\lfloor n/4 \rfloor) + n^2 \\
&\leq T(n/\sqrt{2}) + T(n/2) + T(n/2\sqrt{2}) + T(n/4) + n^2 \\
&\leq cn^2/2 + cn^2/4 + cn^2/8 + cn^2/16 + n^2 \\
&= \frac{15}{16}cn^2 + n^2 \leq cn^2
\end{aligned}$$

L'ultima disequazione è vera per se $15/16c + 1 \leq c$, ovvero se $c \geq 16$. Abbiamo quindi trovato due valori $c = 16$ e $m = 1$ per cui il limite asintotico superiore è soddisfatto. L'equazione di ricorrenza ha quindi complessità $\Theta(n^2)$.
Nota: ad essere precisi, quando $n = 2$, l'equazione di ricorrenza diventa $T(2) = T(1) + T(1) + T(0) + T(0)$ (similmente avviene per $n = 3$). Questo significa che i casi base da considerare sarebbero $T(1)$ e $T(0)$; ma $T(0)$ non è soddisfacibile, e quindi bisogna calcolare anche i casi base $T(2)$ e $T(3)$.

Esercizio 2

Un modo semplice è scrivere una procedura che effettua una visita in profondità da ogni nodo (utilizzando la visita DFS del libro, scrivendo però i valori 0 e 1 al posto di **false** e **true** in *visited*), verificando che tutti siano stati raggiunti. Questa procedura ha costo pari a $O(n(m + n)) = O(mn)$.

```
int count(GRAPH G)
  int count = 0
  int[] visited = new int[1 ... G.n]
  foreach u ∈ G.V() do
      foreach i = 1 to G.n do
          visited[i] = 0
      dfs(G, u, visited)
      if min(visited) == 1 then
          count = count + 1
  return count
```

Volete scrivere ancora meno codice? Utilizzate la procedura isPrincipal() che abbiamo scritto nel compito del 21 Luglio 2014:

```
int count(GRAPH G)
  int count = 0
  foreach u ∈ G.V() do
      if isPrincipal(G, u) then
          count = count + 1
  return count
```

Un modo più efficiente è il seguente: si calcolano le componenti fortemente connesse del grafo utilizzando l'algoritmo di Kosaraju proposto nel libro. Per come è organizzato l'algoritmo, la componente connessa con identificatore più basso (ovvero la prima ad essere scoperta nel grafo trasposto) è l'unica che può contenere nodi che raggiungono tutti gli altri nodi. Non è detto tuttavia che esistano tali nodi. Per verificare, bisogna chiamare isPrincipal() su un nodo contenuto nella componente connessa; se restituisce **true**, è sufficiente restituire il numero di nodi della componente connessa, altrimenti si restituisce 0.

```
int count(GRAPH G)
  int[] id = scc(G)
  int count = 0
  int u = 0
  for i = 1 to G.n do
    if id[i] == 1 then
      count = count + 1
      u = i
  return iif(isPrincipal(G, u), count, 0)
```

Questa procedura ha costo $O(m + n)$.

Esercizio 3

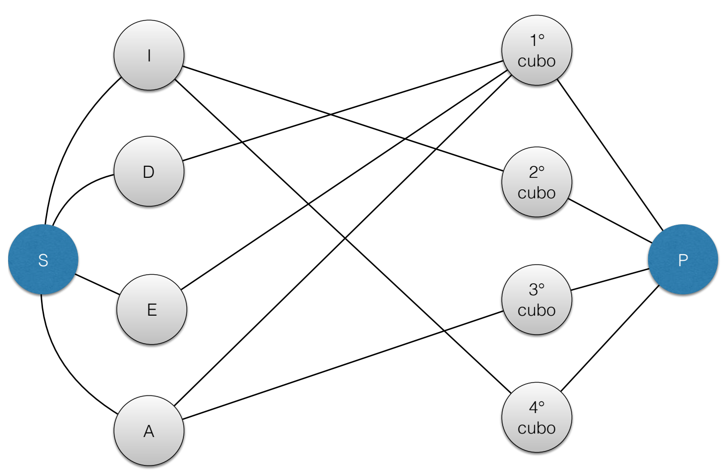
Si può utilizzare una rete di flusso, così organizzata:

- Si aggiunge una sorgente e un pozzo
- Per ogni lettera della parola da realizzare, si aggiunge un nodo
- Per ognuno degli n cubi, si aggiunge un nodo
- Si collega ogni lettera della parola da realizzare con la sorgente
- Si collega ogni lettera della parola da realizzare con i cubi che contengono tale lettera
- Si collega ogni cubo con il pozzo

Tutti gli archi hanno peso pari a 1 (non mostrato in figura), gli archi sono orientati da sinistra a destra (non mostrato in figura). La parola è realizzabile se e solo si ottiene un flusso pari a t.
La complessità si può ottenere dalle seguenti informazioni:

- $t \leq n$ (ovviamente il numero di caratteri della parola non deve essere più grande dei cubi disponibili)
- $|V| = t + n + 2$ (caratteri+cubi+pozzo e sorgente); poichè $t \leq n$, ne viene che $|V| \leq 2n + 2$;
- $|E| \leq t + n + 6n$, in quanto ci sono t archi sorgente-lettere, n archi cubi-pozzo, e ogni cubo può avere al più 6 archi; da cui $|E| \leq 8n$.
- Se ne deduce che $|V| + |E| \leq 10n + 2$.
- Poiché il flusso non può essere superiore a t, ne viene che $|f*| \leq t \leq n$.

Utilizzando il limite di Ford e Fulkerson, si ottiene una complessità $O(|f*|(V + E)) = O(n^2)$.



Esercizio 4

Per risolvere il problema, utilizziamo memoization. Sia $DP[t][i]$ il minimo numero valore in centesimi utilizzando le prime i monete e dovendo pesare t grammi. $DP[t][i]$ può essere calcolato ricorsivamente come segue:

$$DP[t][i] = \begin{cases} 0 & t = 0 \\ +\infty & t < 0 \\ +\infty & i = 0 \wedge t > 0 \\ \min\{DP[t-p[i]][i] + v[i], DP[t][i-1] & t > 0 \wedge i > 0\} \end{cases}$$

I quattro casi corrispondono alle situazioni seguenti:

- Se siamo nei casi particolari generati dal quarto caso in cui uno dei due parametri è minore di zero, ritorna $+\infty$ ad indicare che questo caso non può essere considerato.
- Se non abbiamo più monete ma abbiamo ancora grammi da pesare, ritorna $+\infty$ ad indicare che questo caso non può essere considerato.
- Se $t = 0$, allora 0 grammi sono sufficienti.
- Altrimenti, considera due possibilità: scegliere l'i-esima moneta e continuare ad utilizzarla, o non sceglierla e smettere di utilizzarla. Nel primo caso il peso totale diminuisce di $p[i]$ e il valore ottenuto aumenta di $v[i]$; nel secondo caso diminuisce l'indice i.

Il costo della procedura è pari a $O(nT)$, dovuto all'inizializzazione del vettore DP.

```
int PiggyBank(int[] p, int[] v, int T, int n)
    int[][] DP = new int[1...n][1...T] = { -1 }          % Initialized to −1
    return pbRec(p, v, T, n, DP)
```

```
int pbRec(int[] p, int[] v, int t, int i, int[][] DP)
    if t < 0 then
        return +∞
    if i == 0 and t > 0 then
        return +∞
    if t == 0 then
        return 0
    if DP[i][t] < 0 then
        DP[i][t] = min(pbRec(p, v, t − p[i], i, DP) + v[i], pbRec(p, v, t, i − 1, DP))
    return DP[i][t]
```