Programmazione Dinamica

Esercizio 2 Foglio Esercitazione

Dovete affrontare l'esame di Algoritmi e Strutture Dati. L'esame si compone di n domande che valgono rispettivamente $p[1], \ldots, p[n]$ punti. In base alla vostra esperienza, stimate che le domande richiederanno rispettivamente $t[1], \ldots, t[n]$ minuti per essere svolte. Purtroppo il tempo a vostra disposizione è di T minuti, che potrebbe essere inferiore alla somma dei tempi necessari a rispondere a tutte le domande. I punteggi e i tempi sono interi strettamente positivi.

- Scrivere un algoritmo efficiente che, dati i vettori $p[1..n]$, $t[1..n]$ e il valore di T, restituisce il punteggio massimo che potete ottenere rispondendo correttamente ad un opportuno sottoinsieme delle n domande entro il tempo massimo di T minuti.
- Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione
Definiamo $P[i, j]$ come il punteggio massimo che si può ottenere se si risponde correttamente ad un sottoinsieme delle domande $\{1, \ldots, i\}$ avendo a disposizione un tempo massimo di j minuti. Questo significa che i due indici che definiscono P avranno valore $i \in \{1, \ldots, n\}$ e $j \in \{1, \ldots, T\}$.
Per definire la tabella di programmazione dinamica occorre quindi capire come definire $P[i, j]$. Ci sono due possibilità: la prima è quella in cui non possiamo rispondere alla domanda i-esima perchè richiederebbe più tempo di quello che abbiamo a disposizione ($j < t[i]$) e allora in quel caso il punteggio massimo è quello che avevamo ottenuto rispondendo alle domande comprese nell'insieme $\{1, \ldots, i-1\}$, ovvero $P[i-1, j]$. L'altro caso è quello in cui abbiamo effettivamente il tempo per rispondere alla domanda, ma a questo punto dobbiamo capire se conviene farlo, ovvero se rispondere a quella domanda il punteggio massimo aumenta.
Queste informazioni si possono inglobare nella seguente definizione ricorsive della tabella:

$$P[i, j] = \begin{cases} \max\{P[i-1, j], P[i-1, j-t[i]] + p[i]\} & j \geq t[i] \\ P[i-1, j] & \text{else} \end{cases}$$

Ovviamente (per essere precisi) occorrerebbe anche inizializzare la tabella P, nel seguente modo:

$$P[1, j] = \begin{cases} p[1] & j \geq t[1] \\ 0 & \text{else} \end{cases}$$

Possiamo riassumere questo ragionamento nel seguente algoritmo:

Algorithm 1 ASD($p[1 \dots n]$ array **int**; $t[1 \dots n]$ array **int**, T **int**)

```
1: inizializzare P[1,...,n;0,...,T] di int
2: for j = 0,...,T do
3:     if j ≥ t[1] then
4:         P[1,j] = p[1];
5:     else
6:         P[1,j] = 0;
7:     end if
8: end for
9: for i = 2,...,n do
10:    for j = 0,...,T do
11:        if j ≥ t[i] and P[i−1,j] < P[i−1,j−t[i]] + p[i] then
12:            P[i,j] = P[i−1,j−t[i]] + p[i];
13:        else
14:            P[i,j] = P[i−1,j];
15:        end if
16:    end for
17: end for
18: return P[n,T];
```

Esercizi di Allenamento

Di seguito vi lascio altri esercizi di programmazione dinamica per allenarvi in vista dell'esame. Vi scrivo anche la definizione ricorsiva della tabella per poter confrontare se avete risolto l'esercizio in modo corretto.

Esercizio 1

Un distributore di merendine contiene al suo interno n monete i cui valori sono rispettivamente $c[1], c[2], \dots, c[n]$, dove $c[i]$ sono interi positivi ed è possibile che più monete abbiano lo stesso valore.
Studiamo il problema di decidere se sia o meno possibile erogare un resto uguale a R utilizzando un opportuno sottoinsieme delle n monete a disposizione dove, nuovamente, R è un intero positivo.

- Descrivere lo **pseudo-codice** di un algoritmo efficiente per decidere se il problema ammette una soluzione. L'algoritmo, quindi, dovrà tornare un valore di verità (*true* se si può erogare il resto, *false* altrimenti).
  Attenzione: l'algoritmo non richiede di trovare il numero minimo di monete da erogare, chiede semplicemente se sia possibile erogare il resto corretto.

- Determinare il costo computazionale dell'algoritmo descritto al punto precedente, mo-

tivando adeguatamente la risposta.

Soluzione
Definiamo la matrice booleana $M[1,\ldots,n;0,\ldots,R]$ dove $M[i,r] = \mathit{true}$ solo se esiste un sottoinsieme tra le prime i monete di valore complessivo uguale a r.
La tabella si definisce in modo ricorsivo nel seguente modo:

$$M[i,j] = \begin{cases} M[i-1,r] \vee M[i-1,r-c[i]] & r \geq c[i] \\ M[i-1,r] & \text{else} \end{cases}$$

Ovviamente (per essere precisi) occorrerebbe anche inizializzare la tabella M, nel seguente modo:

$$M[1,r] = \begin{cases} \mathit{true} & r = 0 \text{ or } r = c[1] \\ \mathit{false} & \text{else} \end{cases}$$

La scrittura dello pseudo-codice diventa, a questo punto, molto semplice, ed è lasciata al lettore.

Esercizio 2

Si consideri un insieme di n persone che devono salire su un ascensore. L'ascensore è in grado di sostenere un peso massimo pari a C kg, ma è sufficientemente ampio da ospitare un numero qualsiasi di persone. Siano $p[1],\ldots,p[n]$ i pesi in kg delle n persone. Il vettore dei pesi non è ordinato. Tutti i pesi sono interi.

- Scrivere lo **pseudo-codice** di un algoritmo che, dati in input il valore di C e il vettore $p[1,\ldots,n]$, restituisca il numero massimo di persone che possono salire sull'ascensore contemporaneamente, senza superare il peso massimo complessivo di C kg. In altre parole, si chiede di determinare la cardinalità di un sottoinsieme massimale delle n persone il cui peso complessivo risulti minore o uguale a C.
- Determinare il costo computazionale dell'algoritmo descritto al punto precedente, motivando adeguatamente la risposta.

Soluzione
Definiamo la matrice booleana $N[i,c]$ che indica il numero massimo di persone, scelte tra le prime i, che è possibile trasportare in un ascensore che ha capienza massima c.
La tabella si definisce in modo ricorsivo nel seguente modo:

$$M[i,j] = \begin{cases} \max\{N[i-1,c], N[i-1,c-p[i]]+1\} & c \geq p[i] \\ N[i-1,c] & \text{else} \end{cases}$$

Ovviamente (per essere precisi) occorrerebbe anche inizializzare la tabella N, nel seguente modo:

$$N[1,c] = \begin{cases} 1 & c \geq p[1] \\ 0 & \text{else} \end{cases}$$

La scrittura dello pseudo-codice diventa, a questo punto, molto semplice, ed è lasciata al lettore.

Esercizio 3

Si consideri una scacchiera quadrata rappresentata da una matrice $M[1,\ldots,n;1,\ldots,n]$. Lo scopo del gioco è spostare una pedina dalla casella di partenza in alto a sinistra di coordinate $(1,1)$ alla casella di arrivo in basso a destra di coordinate (n,n). Ad ogni mossa la pedina può essere spostata di una posizione verso il basso oppure verso destra (senza uscire dai bordi della scacchiera). Quindi, se la pedina si trova in (i,j) potrà essere spostata in $(i+1,j)$ oppure $(i,j+1)$, se possibile. Ogni casella $M[i,j]$ contiene un numero reale; man mano che la pedina si muove, il giocatore accumula il punteggio segnato sulle caselle attraversate, incluse quelle di partenza e di arrivo.
Scrivere lo **pseudo-codice** algoritmo efficiente che, data in input la matrice $M[1,\ldots,n;1,\ldots,n]$ restituisce il massimo punteggio che è possibile ottenere spostando la pedina dalla posizione iniziale a quella finale con le regole descritte.