Programmazione Dinamica

Come risolvere esercizi di programmazione dinamica.

Tipologia Es 1

Dovete pianificare una dieta giornaliera. Avete a disposizione n alimenti che forniscono rispettivamente $c[1], \ldots, c[n]$ calorie. In base alle vostre preferenze, stimate che ogni alimento richieda rispettivamente $t[1], \ldots, t[n]$ minuti per essere preparato. Purtroppo il tempo a vostra disposizione per cucinare è T minuti, che potrebbe essere inferiore alla somma dei tempi necessari a preparare tutti gli alimenti. Le calorie e i tempi sono numeri interi strettamente positivi.

- Scrivere un algoritmo efficiente che, dati i vettori $c[1..n]$, $t[1..n]$ e il valore di T, restituisce il massimo numero di calorie che potete ottenere selezionando un opportuno sottoinsieme dei n alimenti entro il tempo massimo di T minuti.
- Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione
Iniziare spiegando come si definisce la matrice e cosa rappresentano le entrate della matrice.
Definiamo $C[i, j]$ come il massimo numero di calorie che si possono ottenere selezionando un sottoinsieme degli alimenti $\{1, \ldots, i\}$ avendo a disposizione un tempo massimo di j minuti. Questo significa che i due indici che definiscono C avranno valore $i \in \{1, \ldots, n\}$ e $j \in \{1, \ldots, T\}$.
Spiegare e ARGOMENTARE!!!!! come definire ricorsivamente le entrate della matrice di programmazione dinamica.
Per definire la tabella di programmazione dinamica occorre capire come definire $C[i, j]$. Ci sono due possibilità:

- Se il tempo necessario a preparare l'alimento i-esimo eccede il tempo a disposizione ($j < t[i]$), allora non possiamo includerlo, e il massimo numero di calorie è quello che si poteva ottenere selezionando un sottoinsieme degli alimenti $\{1, \ldots, i-1\}$, cioè $C[i-1, j]$.
- Se invece abbiamo il tempo per preparare l'alimento, possiamo scegliere se includerlo o meno. Includendolo, il numero di calorie aumenta di $c[i]$, e il nuovo tempo a disposizione diventa $j - t[i]$. Pertanto, il massimo numero di calorie sarà $\max\{C[i-1, j], C[i-1, j - t[i]] + c[i]\}$.

Queste informazioni si possono inglobare nella seguente definizione ricorsiva della tabella:

$$C[i, j] = \begin{cases} \max\{C[i-1, j], C[i-1, j - t[i]] + c[i]\} & \text{se } j \geq t[i] \\ C[i-1, j] & \text{altrimenti} \end{cases}$$

Spiegare e ARGOMENTARE!!!!! come definire I CASI BASE DELLA MATRICE!!! Stiamo definendo per ricorsione, servono dei casi base!!!!
Per completare la definizione dell'algoritmo, occorre inizializzare la tabella C nel seguente modo:

$$C[1,j] = \begin{cases} c[1] & \text{se } j \geq t[1] \\ 0 & \text{altrimenti} \end{cases}$$

Una volta fatto questo, riassumere il procedimento in un algoritmo. L'algoritmo solo mette insieme quello che avete scritto sopra. SCRIVERE ESPLICITAMENTE I CASI BASE ANCHE NELLO PSEUDOCODICE!!!!!!!

Possiamo riassumere il ragionamento nel seguente algoritmo:

Algorithm 1 MaxCalories($c[1 \dots n]$ array **int**; $t[1 \dots n]$ array **int**; T **int**)

```
1:  Inizializzare C[1...n; 0...T] di int
2:  for j = 0, ..., T do
3:      if j ≥ t[1] then
4:          C[1, j] = c[1]
5:      else
6:          C[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 C[i − 1, j] < C[i − 1, j − t[i]] + c[i] then
12:             C[i, j] = C[i − 1, j − t[i]] + c[i]
13:         else
14:             C[i, j] = C[i − 1, j]
15:         end if
16:     end for
17: end for
18: return C[n, T]
```

Ricordare il costo comptuazionale!!!

Costo computazionale: L'algoritmo ha complessità temporale $O(n \cdot T)$ e complessità spaziale $O(n \cdot T)$ poiché la tabella C ha dimensioni $n \times T$.

Tipologia Es 2

Un bartender ha a disposizione n ingredienti liquidi i cui volumi sono rispettivamente $v[1], v[2], \ldots, v[n]$, dove $v[i]$ sono interi positivi ed è possibile che più ingredienti abbiano lo stesso volume.
Studiamo il problema di decidere se sia o meno possibile preparare un cocktail con un volume totale esattamente pari a V utilizzando un opportuno sottoinsieme degli n ingredienti a disposizione dove, nuovamente, V 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 preparare il cocktail, *false* altrimenti). Attenzione: l'algoritmo non richiede di minimizzare il numero di ingredienti utilizzati, ma chiede semplicemente se sia possibile raggiungere il volume corretto.
- Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione

Iniziare spiegando come si definisce la matrice e cosa rappresentano le entrate della matrice.

Definiamo la matrice booleana $M[1, \ldots, n; 0, \ldots, V]$ dove $M[i, j] = \textit{true}$ se è possibile preparare un cocktail con volume esattamente pari a j utilizzando un sottoinsieme degli ingredienti $\{1, \ldots, i\}$.

Spiegare e ARGOMENTARE!!!!! come definire ricorsivamente le entrate della matrice di programmazione dinamica.
La tabella si definisce in modo ricorsivo nel seguente modo:

- Se il volume del i-esimo ingrediente è maggiore di j, l'ingrediente non può essere usato, quindi $M[i, j] = M[i-1, j]$.
- Se invece il volume del i-esimo ingrediente è minore o uguale a j, possiamo decidere se includerlo o meno. Se decidiamo di includerlo, allora il nuovo volume residuo sarà $j - v[i]$. Pertanto:
$$M[i, j] = M[i-1, j] \vee M[i-1, j - v[i]]$$

Spiegare e ARGOMENTARE!!!!! come definire I CASI BASE DELLA MATRICE!!! Stiamo definendo per ricorsione, servono dei casi base!!!! ATTENZIONE QUA CI SONO DUE CASI BASE IN CUI METTERE TRUE!!!

Per inizializzare correttamente la tabella, abbiamo:

$$M[1,j] = \begin{cases} \mathit{true} & \text{se } j = 0 \text{ o } j = v[1] \\ \mathit{false} & \text{altrimenti} \end{cases}$$

Infatti è sempre possibile ottenere un volume pari a 0 (non utilizzando nessun liquido) oppure se $j = v[1]$ (usando solo il primo liquido, se il suo valore è pari a V).

Lo pseudocodice segue direttamente da queste definizioni, vedi es. precedente.

Anche qua, basta copiare in psuedocodice quello che avete scritto sopra!!!! Ricordare di mettere nello pseudocodice CASI BASE!!!!

Costo computazionale: L'algoritmo ha complessità temporale $O(n \cdot V)$ e complessità spaziale $O(n \cdot V)$, poiché la matrice M ha dimensioni $n \times V$.