

CoderFarm - Corso avanzato

Lezione 2

Lorenzo Ferrari, Davide Bartoli

28 aprile 2023

Problemi

Removing Digits

Removing Digits

Dato un intero positivo $N \leq 10^6$, in una mossa puoi sottrarre una delle cifre di N ad N . Di quante mosse hai bisogno per rendere N zero?

<https://cses.fi/problemset/task/1637>

Problemi

Removing Digits

- ▶ non funzione sottrarre sempre la cifra maggiore
- ▶ ho un numero limitato di stati e per ogni stato ho un numero limitato di scelte

Problemi

Removing Digits

- ▶ non funzione sottrarre sempre la cifra maggiore
- ▶ ho un numero limitato di stati e per ogni stato ho un numero limitato di scelte

Programmazione dinamica!

Problemi

Removing Digits

- ▶ non funzione sottrarre sempre la cifra maggiore
- ▶ ho un numero limitato di stati e per ogni stato ho un numero limitato di scelte

Programmazione dinamica!

- ▶ sia $dp[i]$ la risposta per $N = i$.
- ▶ $dp[0] = 0$
- ▶ $dp[i] = 1 + \min(dp[i - c_1], \dots, dp[i - c_k])$ dove c_1, \dots, c_k sono le cifre di i .

Problemi

Removing Digits

- ▶ non funzione sottrarre sempre la cifra maggiore
- ▶ ho un numero limitato di stati e per ogni stato ho un numero limitato di scelte

Programmazione dinamica!

- ▶ sia $dp[i]$ la risposta per $N = i$.
- ▶ $dp[0] = 0$
- ▶ $dp[i] = 1 + \min(dp[i - c_1], \dots, dp[i - c_k])$ dove c_1, \dots, c_k sono le cifre di i .

Complessità di tempo

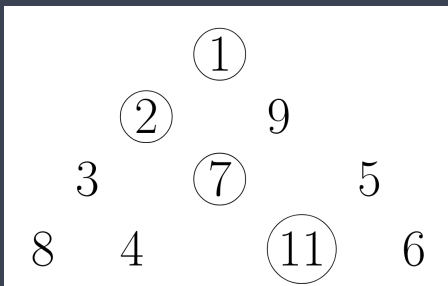
- ▶ $O(N)$ stati
- ▶ $O(\log N)$ per transizione
- ▶ $O(N \log N)$

Problemi

Discesa massima

Discesa massima

Dati $\frac{N(N+1)}{2}$ numeri disposti su una piramide di $N \leq 10$ righe, trovare la somma massima di un percorso dalla cima alla base.



<https://training.olinfo.it/#/task/discesa/statement>

Problemi

Discesa massima

- ▶ idee?
- ▶ quanti percorsi diversi esistono?

Problemi

Discesa massima


- ▶ idee?
- ▶ quanti percorsi diversi esistono?
 - ▶ per N volte devo scegliere se andare a dx o a sx
 - ▶ ogni percorso è univocamente determinato dalla sequenza di scelte, quindi 2^N percorsi diversi

Problemi

Discesa massima

- ▶ idee?
- ▶ quanti percorsi diversi esistono?
 - ▶ per N volte devo scegliere se andare a dx o a sx
 - ▶ ogni percorso è univocamente determinato dalla sequenza di scelte, quindi 2^N percorsi diversi
- ▶ i limiti su N sono abbastanza piccoli da permetterci di eseguire una ricerca completa

Discesa massima



```
int n;  
int v[N][N];  
  
// i: di quanti piani sono sceso  
// j: sul piano i, dove mi trovo  
int solve(int i, int j) {  
    if (i == n) return 0;  
    int sx = solve(i+1, j);  
    int dx = solve(i+1, j+1);  
    return v[i][j] + max(sx, dx);  
}
```

Problemi

Triangolo

Triangolo

Stesso problema di prima, ma $N \leq 100$.

Problemi

Triangolo

Triangolo

Stesso problema di prima, ma $N \leq 100$.

Come si risolve?

Problemi

Triangolo

Triangolo

Stesso problema di prima, ma $N \leq 100$.

Come si risolve?

- memorizziamo la risposta per ogni stato

Problemi

Triangolo

Triangolo


Stesso problema di prima, ma $N \leq 100$.

Come si risolve?

- ▶ memorizziamo la risposta per ogni stato
 - ▶ due chiamate a `solve(i, j)` ritornano lo stesso risultato
 - ▶ possiamo calcolare ogni stato una sola volta
- ▶ numero di stati: $O(N^2)$
- ▶ costo per transizione $O(1)$
- ▶ complessità di tempo: $O(N^2 \cdot 1) = O(N^2)$

<https://training.olinfo.it/#/task/triangolo/statement>

Triangolo



```
int dp[N][N];
bool vis[N][N];

int solve(int i, int j) {
    if (i == n) return 0;
    if (vis[i][j]) {
        return dp[i][j];
    }
    vis[i][j] = true;
    int sx = solve(i+1, j);
    int dx = solve(i+1, j+1);
    return dp[i][j] = v[i][j] + max(sx, dx);
}
```


Problemi

Grid paths

Grid paths

Considera una griglia $N \times N$ con $N \leq 1000$ dove alcuni blocchi contengono delle trappole. Non è consentito camminare sulle trappole e puoi muoverti solo in basso o a destra. Calcola il numero di percorsi diversi dal quadratino in in alto a sinistra a quello in basso a destra. Stampa la risposta modulo $10^9 + 7$.

<https://cses.fi/problemset/task/1638>

Problemi

Grid Paths

- ▶ sia $dp[i][j]$ il numero di modi per andare da $(1; 1)$ a $(i; j)$.
- ▶ per tutte le celle $(i; j)$ che non contengono trappole:

Problemi

Grid Paths

- ▶ sia $dp[i][j]$ il numero di modi per andare da $(1; 1)$ a $(i; j)$.
- ▶ per tutte le celle $(i; j)$ che non contengono trappole:

$$dp[i][j] = \begin{cases} 1 & \text{se } (i; j) = (1; 1) \\ dp[i-1][j] + dp[i][j-1] & \text{altrimenti} \end{cases}$$

Knapsack

Knapsack (o problema dello zaino)

Vengono dati $N \leq 100$ oggetti, ognuno con un peso e un valore. Si vuole scegliere un sottoinsieme di oggetti di valore massimo che abbia peso totale non superiore a $W \leq 10^5$ (capacità dello zaino). $\text{peso} \leq W$ e $\text{valore} \leq 10^9$ per ogni oggetto.

https://atcoder.jp/contests/dp/tasks/dp_d

Knapsack

Cerchiamo di approcciare il problema anche con soluzioni non ottimali/errate.

- Potremmo provare tutti i sottoinsiemi di oggetti, ma è decisamente troppo lento (sono $O(2^N)$).

Knapsack

Cerchiamo di approcciare il problema anche con soluzioni non ottimali/errate.

- ▶ Potremmo provare tutti i sottoinsiemi di oggetti, ma è decisamente troppo lento (sono $O(2^N)$).
- ▶ Potremmo ordinare gli oggetti per valore/peso decrescente e prendere i migliori fino a quando non superiamo la capacità dello zaino.

Knapsack

Cerchiamo di approcciare il problema anche con soluzioni non ottimali/errate.

- ▶ Potremmo provare tutti i sottoinsiemi di oggetti, ma è decisamente troppo lento (sono $O(2^N)$).
- ▶ Potremmo ordinare gli oggetti per valore/peso decrescente e prendere i migliori fino a quando non superiamo la capacità dello zaino.
 - ▶ non è sempre ottimale! (es. $W = 4$, $N = 3$, $pesi = (1, 2, 2)$, $valori = (2, 3, 3)$).

Knapsack

Semplifichiamo il problema: immaginiamo che gli oggetti ci arrivino uno alla volta. Possiamo scegliere se prendere l'oggetto o no (dobbiamo controllare di non superare W), quindi abbiamo 2 opzioni.

Knapsack

Semplifichiamo il problema: immaginiamo che gli oggetti ci arrivino uno alla volta. Possiamo scegliere se prendere l'oggetto o no (dobbiamo controllare di non superare W), quindi abbiamo 2 opzioni.

Rappresentiamo lo stato con 2 valori:

Knapsack

Semplifichiamo il problema: immaginiamo che gli oggetti ci arrivino uno alla volta. Possiamo scegliere se prendere l'oggetto o no (dobbiamo controllare di non superare W), quindi abbiamo 2 opzioni.

Rappresentiamo lo stato con 2 valori:

- ▶ *pos*: l'indice dell'oggetto che stiamo considerando
- ▶ *peso*: il peso totale che abbiamo preso fino ad ora

Knapsack

Semplifichiamo il problema: immaginiamo che gli oggetti ci arrivino uno alla volta. Possiamo scegliere se prendere l'oggetto o no (dobbiamo controllare di non superare W), quindi abbiamo 2 opzioni.

Rappresentiamo lo stato con 2 valori:

- ▶ pos : l'indice dell'oggetto che stiamo considerando
- ▶ $peso$: il peso totale che abbiamo preso fino ad ora

$dp[pos][peso] :=$ valore massimo che possiamo ottenere considerando gli oggetti da 0 a pos e avendo peso totale $peso$. La risposta è $\max(dp[N - 1][j])$ per $j \leq W$.

Knapsack

Semplifichiamo il problema: immaginiamo che gli oggetti ci arrivino uno alla volta. Possiamo scegliere se prendere l'oggetto o no (dobbiamo controllare di non superare W), quindi abbiamo 2 opzioni.

Rappresentiamo lo stato con 2 valori:

- pos : l'indice dell'oggetto che stiamo considerando
- $peso$: il peso totale che abbiamo preso fino ad ora

$dp[pos][peso] :=$ valore massimo che possiamo ottenere considerando gli oggetti da 0 a pos e avendo peso totale $peso$. La risposta è $\max(dp[N - 1][j])$ per $j \leq W$.

Ci sono in totale $O(N \cdot W)$ stati, e per ognuno abbiamo 2 opzioni. La complessità è $O(N \cdot W)$.

Longest Common Subsequence

Problema

LCS

Dati due stringhe S e T di lunghezza $N, M \leq 3000$, calcola la lunghezza della sottosequenza piu lunga che sia presente sia in S che in T .

https://atcoder.jp/contests/dp/tasks/dp_f

Longest Common Subsequence

Soluzione

$dp[i][j] :=$ lunghezza della LCS tra $S[i \dots N]$ e $T[j \dots M]$.
La risposta è $dp[0][0]$.

Longest Common Subsequence

Soluzione

$dp[i][j] :=$ lunghezza della LCS tra $S[i \dots N]$ e $T[j \dots M]$.

La risposta è $dp[0][0]$.

Ci troviamo in $dp[i][j]$. Abbiamo due casi possibili:

- ▶ $S[i] \neq T[j]$, allora $dp[i][j] = \max(dp[i+1][j], dp[i][j+1])$;
- ▶ $S[i] = T[j]$, allora $dp[i][j] = 1 + dp[i+1][j+1]$.

Longest Common Subsequence

Soluzione

$dp[i][j] :=$ lunghezza della LCS tra $S[i \dots N]$ e $T[j \dots M]$.

La risposta è $dp[0][0]$.

Ci troviamo in $dp[i][j]$. Abbiamo due casi possibili:

- ▶ $S[i] \neq T[j]$, allora $dp[i][j] = \max(dp[i+1][j], dp[i][j+1])$;
- ▶ $S[i] = T[j]$, allora $dp[i][j] = 1 + dp[i+1][j+1]$.

Il numero di stati è $O(N \cdot M)$, e per ogni stato abbiamo 2 opzioni, quindi la complessità è $O(N \cdot M)$.

Longest Common Subsequence

Soluzione

$dp[i][j] :=$ lunghezza della LCS tra $S[i \dots N]$ e $T[j \dots M]$.

La risposta è $dp[0][0]$.

Ci troviamo in $dp[i][j]$. Abbiamo due casi possibili:

- ▶ $S[i] \neq T[j]$, allora $dp[i][j] = \max(dp[i+1][j], dp[i][j+1])$;
- ▶ $S[i] = T[j]$, allora $dp[i][j] = 1 + dp[i+1][j+1]$.

Il numero di stati è $O(N \cdot M)$, e per ogni stato abbiamo 2 opzioni, quindi la complessità è $O(N \cdot M)$.

Problemi aggiuntivi

https://training.olinfo.it/#/task/ois_nonna/statement

https://training.olinfo.it/#/task/ois_police3/statement

https://training.olinfo.it/#/task/ois_police4/statement

<https://training.olinfo.it/#/task/lotteria/statement>

<https://cses.fi/problemset/>

<https://atcoder.jp/contests/dp/tasks>