CoderFarm - Corso avanzato

Lezione 2

Lorenzo Ferrari, Davide Bartoli

14 dicembre 2022

Lotteria

Edoardo e la lotteria

Nel lotto da Cesena bisogna scegliere $n \leq 20$ numeri da 1 a $m \leq 200\,000$. Una lista è fortunata se e solo se ogni numero è almeno il doppo del precedente. Contare quante liste fortunate esistono e stampare la risposta modulo $10^9 + 7$.

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

Lotteria

Edoardo e la lotteria

Nel lotto da Cesena bisogna scegliere $n \leq 20$ numeri da 1 a $m \leq 200\,000$. Una lista è fortunata se e solo se ogni numero è almeno il doppo del precedente. Contare quante liste fortunate esistono e stampare la risposta modulo $10^9 + 7$.

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

- idee?

Lotteria

Edoardo e la lotteria

Nel lotto da Cesena bisogna scegliere $n \leq 20$ numeri da 1 a $m \leq 200\,000$. Una lista è fortunata se e solo se ogni numero è almeno il doppo del precedente. Contare quante liste fortunate esistono e stampare la risposta modulo $10^9 + 7$.

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

- idee?
- ~~se lo vediamo in questa lezione evidentemente è una dp~~

Lotteria

Edoardo e la lotteria

Nel lotto da Cesena bisogna scegliere $n \leq 20$ numeri da 1 a $m \leq 200\,000$. Una lista è fortunata se e solo se ogni numero è almeno il doppo del precedente. Contare quante liste fortunate esistono e stampare la risposta modulo $10^9 + 7$.

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

- idee?
- ~~se lo vediamo in questa lezione evidentemente è una dp~~
- sia $dp[i][j]$ il numero di liste di i elementi il cui ultimo elemento è j

Lotteria

Edoardo e la lotteria

Nel lotto da Cesena bisogna scegliere $n \leq 20$ numeri da 1 a $m \leq 200\,000$. Una lista è fortunata se e solo se ogni numero è almeno il doppo del precedente. Contare quante liste fortunate esistono e stampare la risposta modulo $10^9 + 7$.

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

- idee?
- ~~se lo vediamo in questa lezione evidentemente è una dp~~
- sia $dp[i][j]$ il numero di liste di i elementi il cui ultimo elemento è j
- ovviamente $dp[1][j] = 1 \; \forall j$
- la risposta è $dp[n][m] + dp[n][m-1] + \cdots + dp[n][1]$

Lotteria

- supponiamo di avere calcolato $dp[i'][j']$ per ogni $i' < i$ e $j' < j$

Lotteria

- supponiamo di avere calcolato $dp[i'][j']$ per ogni $i' < i$ e $j' < j$
- se l'i-esimo elemento è j e l'$(i-1)$-esimo elemento è j', deve valere $j \geq 2j'$, quindi $j' \leq \lfloor j/2 \rfloor$

Lotteria

- supponiamo di avere calcolato $dp[i'][j']$ per ogni $i' < i$ e $j' < j$
- se l'i-esimo elemento è j e l'$(i-1)$-esimo elemento è j', deve valere $j \geq 2j'$, quindi $j' \leq \lfloor j/2 \rfloor$

$$dp[i][j] = dp[i-1][1] + dp[i-1][2] + \cdots + dp[i-1][j/2]$$

Lotteria

- supponiamo di avere calcolato $dp[i'][j']$ per ogni $i' < i$ e $j' < j$
- se l'i-esimo elemento è j e l'$(i-1)$-esimo elemento è j', deve valere $j \geq 2j'$, quindi $j' \leq \lfloor j/2 \rfloor$

$$dp[i][j] = dp[i-1][1] + dp[i-1][2] + \cdots + dp[i-1][j/2]$$

- implementiamo questa soluzione

```cpp
const int mod = 1e9 + 7;

vector<vector<int>> dp(n+1, vector<int>(m+1));
for (int i = 1; i <= m; ++i) {
    dp[1][i] = 1;
}
for (int i = 2; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        for (int jj = 1; jj <= j/2; ++jj) {
            dp[i][j] = (dp[i][j] + dp[i-1][jj]) % mod;
        }
    }
}

int ans = 0;
for (int i = 1; i <= m; ++i) {
    ans = (ans + dp[n][i]) % mod;
}
```

Lotteria

Complessità

- $O(nm)$ stati
- $O(m)$ per transizione

Lotteria

Complessità

- $O(nm)$ stati
- $O(m)$ per transizione
- $O(nm^2)$ complessità totale

Lotteria

Complessità

- $O(nm)$ stati
- $O(m)$ per transizione
- $O(nm^2)$ complessità totale

La soluzione attuale prende 70/100 punti.

Lottera

Ottimizzazione

Come ottimizzare?

Lottera

Ottimizzazione

Come ottimizzare?

- per calcolare $dp[i][j]$ e $dp[i][j+1]$ la maggior parte dei valori che sommiamo sono uguali

Come ottimizzare?

- per calcolare $dp[i][j]$ e $dp[i][j+1]$ la maggior parte dei valori che sommiamo sono uguali
- in particolare, $dp[i][j+1]$ differisce da $dp[i][j]$ solo per i valori $dp[i-1][x]$ con $j/2 < x \leq (j+1)/2$ (al massimo un valore)

Lottera

Ottimizzazione

Come ottimizzare?

- per calcolare $dp[i][j]$ e $dp[i][j+1]$ la maggior parte dei valori che sommiamo sono uguali
- in particolare, $dp[i][j+1]$ differisce da $dp[i][j]$ solo per i valori $dp[i-1][x]$ con $j/2 < x \leq (j+1)/2$ (al massimo un valore)
- possiamo ridurre il costo della transizione a $O(1)$!

```cpp
vector<vector<int>> dp(n+1, vector<int>(m+1));
for (int i = 1; i <= m; ++i) {
    dp[1][i] = 1;
}
for (int i = 2; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        dp[i][j] = dp[i][j-1];
        for (int jj = (j-1)/2+1; jj <= j/2; ++jj) {
            dp[i][j] += dp[i-1][jj];
            dp[i][j] %= mod;
        }
    }
}
```

Lotteria

Somme prefisse

Un'alternativa è salvarsi le **somme prefisse**

Un'alternativa è salvarsi le **somme prefisse**

- dato un array $a[0], a[1], \ldots, a[n-1]$, l'array delle somme prefisse di a è un array prf tale che $prf[i] = a[0] + a[1] + \cdots + a[i]$

Lotteria

Somme prefisse

Un'alternativa è salvarsi le **somme prefisse**

- dato un array $a[0], a[1], \ldots, a[n-1]$, l'array delle somme prefisse di a è un array prf tale che $prf[i] = a[0] + a[1] + \cdots + a[i]$
- $prf[0] = a[0]$
- $prf[i] = prf_{[}i - 1] + a_{[}i] \; \forall i > 0$

Lotteria

Somme prefisse

Un'alternativa è salvarsi le **somme prefisse**

- dato un array $a[0], a[1], \ldots, a[n-1]$, l'array delle somme prefisse di a è un array *prf* tale che $prf[i] = a[0] + a[1] + \cdots + a[i]$
- $prf[0] = a[0]$
- $prf[i] = prf_[i-1] + a_[i] \; \forall i > 0$

Se a non cambia mai, possiamo trovare la somma degli $a[i]$ con $l \leq i \leq r$ in $O(1)$ come $sum(l, r) = prf[r] - prf[l-1]$

Nel nostro caso $l = 0, r = \lfloor j/2 \rfloor$

```cpp
vector<vector<int>> dp(n+1, vector<int>(m+1));
vector<vector<int>> prf(n+1, vector<int>(m+1));
for (int i = 1; i <= m; ++i) {
    dp[1][i] = 1;
    prf[1][i] = prf[1][i-1] + 1;
}
for (int i = 2; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        dp[i][j] = prf[i-1][j/2];
        prf[i][j] = prf[i][j-1] + dp[i][j];
        prf[i][j] %= mod;
    }
}
```

Matching

Ci sono N uomini e N donne ($N \leq 20$), entrambi numerati $2, \ldots, N$. Data una matrice a, l'uomo i e la donna j sono compatibili sse $a_{i,j} = 1$. Vuoi fare N coppie, ognuna costituita da un uomo e una donna compatibili e inoltre ognuno deve appartenere a esattamente una coppia.
Trova in quando modi puoi formare le N coppie modulo $10^9 + 7$.

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

Bitmask dp

Matching

Matching

Ci sono N uomini e N donne ($N \leq 20$), entrambi numerati $2, \ldots, N$. Data una matrice a, l'uomo i e la donna j sono compatibili sse $a_{i,j} = 1$. Vuoi fare N coppie, ognuna costituita da un uomo e una donna compatibili e inoltre ognuno deve appartenere a esattamente una coppia.
Trova in quando modi puoi formare le N coppie modulo $10^9 + 7$.

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

- quali potrebbero essere gli stati della nostra dp?

Matching

Ci sono N uomini e N donne ($N \leq 20$), entrambi numerati $2, \ldots, N$. Data una matrice a, l'uomo i e la donna j sono compatibili sse $a_{i,j} = 1$. Vuoi fare N coppie, ognuna costituita da un uomo e una donna compatibili e inoltre ognuno deve appartenere a esattamente una coppia.
Trova in quando modi puoi formare le N coppie modulo $10^9 + 7$.

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

- quali potrebbero essere gli stati della nostra dp?
- un set S di uomini "assegnati" alle prime j donne
- $dp[S][j]$: numero di modi per farlo

Matching

Ci sono N uomini e N donne ($N \leq 20$), entrambi numerati $2, \ldots, N$. Data una matrice a, l'uomo i e la donna j sono compatibili sse $a_{i,j} = 1$. Vuoi fare N coppie, ognuna costituita da un uomo e una donna compatibili e inoltre ognuno deve appartenere a esattamente una coppia.
Trova in quando modi puoi formare le N coppie modulo $10^9 + 7$.

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

- quali potrebbero essere gli stati della nostra dp?
- un set S di uomini "assegnati" alle prime j donne
- $dp[S][j]$: numero di modi per farlo
- per le transizioni possiamo pensare a chi va in coppia la donna j, l'ultima del prefisso

Bitmask dp

Matching

In particolare

$$dp[S][j] = \sum_{i=1}^{N} (dp[S \setminus \{i\}][j-1] \text{ se } a_{i,j} = 1)$$

Bitmask dp
Matching

In particolare

$$dp[S][j] = \sum_{i=1}^{N} (dp[S \backslash \{i\}][j-1] \text{ se } a_{i,j} = 1)$$

Stiamo contando tutti gli assegnamenti esattamente una volta: se assegnamo la donna j a diversi i non abbiamo sovrapposizioni

Bitmask dp

Matching

In particolare

$$dp[S][j] = \sum_{i=1}^{N} (dp[S \backslash \{i\}][j-1] \text{ se } a_{i,j} = 1)$$

Stiamo contando tutti gli assegnamenti esattamente una volta: se assegnamo la donna j a diversi i non abbiamo sovrapposizioni

La risposta è $dp[\{1, 2, \ldots, N\}][N]$

Bitmask dp

Matching

- come si implementa?

Bitmask dp

Matching

- come si implementa?
- ci serve un modo per rappresentare un set S con un intero

Bitmask dp

Matching

- come si implementa?
- ci serve un modo per rappresentare un set S con un intero
- possiamo rappresentare S come un intero *mask* a N bit dove il bit i-esimo è 1 se e solo se $i \in S$

Bitmask dp

Matching

- come si implementa?
- ci serve un modo per rappresentare un set S con un intero
- possiamo rappresentare S come un intero *mask* a N bit dove il bit i-esimo è 1 se e solo se $i \in S$
- le operazioni bitwise ci consentono di effettuare efficientemente operazioni come inserire/rimuovere un elemento e controllare se un elemento è presente

Bitmask dp

Matching

- l'elemento i-esimo è rappresentato dalla potenza 2^i con $0 \leq i \leq N-1$

Bitmask dp

Matching

- l'elemento i-esimo è rappresentato dalla potenza 2^i con $0 \leq i \leq N-1$
- `if (mask & (1 << i))` controlla se i è presente in *mask*
- `mask |= (1 << i)` inseriesce i in *mask*
- `mask &= ~(1 << i)` rimuove i da *mask*
- `mask ^= (1 << i)` cambia lo stato di i in *mask*: se i è presente lo rimuove, altrimenti lo inserisce

```cpp
vector<vector<int>> good(n, vector<int>(n));
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        cin >> good[i][j];
    }
}
vector<vector<int>> dp(1 << n, vector<int>(n+1));
dp[0][0] = 1; // 1 modo di assegnare 0 elementi
for (int mask = 1; mask < (1 << n); ++mask) {
    for (int i = 0; i < n; ++i) {
        if (!(mask & (1 << i))) continue;
        int old_mask = mask ^ (1 << i);
        for (int j = 1; j <= n; ++j) {
            if (!good[i][j-1]) continue;
            dp[mask][j] += dp[old_mask][j-1];
            dp[mask][j] %= mod;
        }
    }
}
cout << dp[(1 << n) - 1][n] << "\n";
```

Matching

Complessità

Un problema (praticamente) identico è ois_dristor2[1]

[1] che nessuna squadra ha risolto in gara (neanch'io, Lorenzo) :P

Matching

Complessità

Un problema (praticamente) identico è ois_dristor2[1]

Lost in Dristor 2

Dati $N \leq 14$, $M \leq 100$ e una matrice $N \times M$ che indica se elementi del primo insieme sono a coppie compatibili con elementi del secondo insieme, conta quanti max matching (matching con il massimo numero di coppie) esistono.

[1] che nessuna squadra ha risolto in gara (neanch'io, Lorenzo) :P

Matching

Complessità

Un problema (praticamente) identico è ois_dristor2[1]

Lost in Dristor 2

Dati $N \leq 14$, $M \leq 100$ e una matrice $N \times M$ che indica se elementi del primo insieme sono a coppie compatibili con elementi del secondo insieme, conta quanti max matching (matching con il massimo numero di coppie) esistono.

- in questo caso dobbiamo anche contare il numero di elementi in *mask*

[1] che nessuna squadra ha risolto in gara (neanch'io, Lorenzo) :P

Matching

Complessità

Un problema (praticamente) identico è ois_dristor2[1]

Lost in Dristor 2

Dati $N \leq 14$, $M \leq 100$ e una matrice $N \times M$ che indica se elementi del primo insieme sono a coppie compatibili con elementi del secondo insieme, conta quanti max matching (matching con il massimo numero di coppie) esistono.

- in questo caso dobbiamo anche contare il numero di elementi in *mask*
- si può fare in $O(1)$ con `__builtin_popcount(mask)`, che restituisce il numero di bit 1 nella rappresentazione binaria di *mask*

[1] che nessuna squadra ha risolto in gara (neanch'io, Lorenzo) :P

Matching

Complessità

Un problema (praticamente) identico è ois_dristor2[1]

Lost in Dristor 2

Dati $N \leq 14$, $M \leq 100$ e una matrice $N \times M$ che indica se elementi del primo insieme sono a coppie compatibili con elementi del secondo insieme, conta quanti max matching (matching con il massimo numero di coppie) esistono.

- in questo caso dobbiamo anche contare il numero di elementi in *mask*
- si può fare in $O(1)$ con `__builtin_popcount(mask)`, che restituisce il numero di bit 1 nella rappresentazione binaria di *mask*
- divertitevi a implementarlo (è molto pulito)

[1] che nessuna squadra ha risolto in gara (neanch'io, Lorenzo) :P

Problemi addizionali

Implementateli che vi fa bene

https://training.olinfo.it/#/task/ois_dristor2/statement
https://training.olinfo.it/#/task/ois_police4/statement
https://atcoder.jp/contests/dp/tasks
https://cses.fi/problemset/