



Problem Ramen

Input file `stdin`
Output file `stdout`

Considerăm N prieteni F_0, \dots, F_{N-1} și N tipuri de ramen R_0, \dots, R_{N-1} la restaurantul cu ramen. Fiecare prieten F_i are o anumită afinitate A_{ij} pentru tipul j de ramen — cu cât afinitatea este mai mare cu atât prietenului F_i îi place mai mult ramenul R_j . Afinitățile fiecărui prieten sunt distincte — adică $A_{ij} \neq A_{ij'}$ pentru $j \neq j'$. Desigur, este posibil ca $A_{ij} < 0$.

Să presupunem că prietenii vizitează restaurantul cu ramen de mai multe ori. În timpul fiecărei vizite, doi prieteni nu pot mânca același tip de ramen (cantitatea de ingrediente ar fi insuficientă). Să presupunem că prietenii își iau ramenul în ordinea $F_{\pi_0}, \dots, F_{\pi_{N-1}}$, pentru o anumită permutare π_0, \dots, π_{N-1} a $0, \dots, N-1$. Atunci, prietenul F_{π_0} va lua ramenul favorit (adică, cel pentru care are afinitatea cea mai mare), prietenul F_{π_1} va lua ramenul favorit *cu excepția celui luat de F_{π_0}* , și așa mai departe. Cu alte cuvinte, F_{π_i} va lua ramenul favorit dintre cele care nu sunt luate de $F_{\pi_0}, \dots, F_{\pi_{i-1}}$.

Bunătatea unei permutări π este suma afinităților prietenilor pentru tipurile de ramen pe care le iau. Cu alte cuvinte, dacă prietenul i ia ramenul de tip σ_i , atunci bunătatea lui π este $\sum_{i=0}^{N-1} A_{i, \sigma(i)}$.

Scopul tău este să găsești o permutare π cu o bunătate maximă. Poți face acest lucru experimentând cu diferite ordini în care prietenii își iau ramenul (adică, diferite vizite la restaurant). Scopul este de a găsi o permutare optimă fără a fi nevoie de prea multe vizite la restaurant

Protocol de interacțiune

Trebuie să implementezi următoarea funcție:

```
std::vector<int> find_order(int N);
```

Aici, N este numărul de prieteni. Funcția trebuie să returneze o permutare π a numerelor $0, \dots, N-1$ cu bunătatea maximă. Pentru a o implementa, puteți apela în mod repetat următoarea funcție de cel mult 750 ori:

```
std::vector<std::pair<int, int>> query(const std::vector<int>& order);
```

Funcția primește ca date de intrare o permutare π a numerelor $0, \dots, N-1$ (identificată prin parametrul `order`) și returnează o listă de perechi $(\sigma(i), A_{i, \sigma(i)})$, unde $\sigma(i)$ este tipul de ramen luat de prietenul i dacă prietenii își iau ramenul în ordinea dată de π .

Restrictions

- $1 \leq N \leq 75$
- $|A_{ij}| \leq 2\,000\,000$
- Soluția comisiei științifice necesită cel mult cN^k interogări pentru anumite constante $c, k \geq 1$. Pentru a nu suprasolicita infrastructura de testare, puteți apela `query` de cel mult 750 ori, ceea ce reflectă în mod generos performanța practică a soluției modelului.



#	Points	Restrictions
1	5	$N = 5$
2	15	$N = 15$
3	20	$N = 30$
4	20	$N = 45$
5	20	$N = 60$
6	20	$N = 75$

Examples

Programul tău	Programul comisiei
	Calls <code>find_order(2)</code> .
<code>query({0, 1})</code>	
	<code>{{0, 9}, {1, 0}}</code>
<code>query({1, 0})</code>	
	<code>{{1, 5}, {0, 5}}</code>
<code>find_order(2)</code> returns <code>{1, 0}</code> .	

În acest exemplu, sunt $N = 2$ prieteni cu următoarele afinități $A_{i,j}$ pentru $N = 2$ tipuri de ramen:

A	0	1
0	9	5
1	5	0

Interacțiunea începe cu apelul comisiei la funcția ta: `find_order(2)`. Funcția ta interoghează apoi cele două permutări posibile: `{0, 1}` și `{1, 0}`. Prima obține o bunătate de $0 + 9 = 9$, în timp ce a doua obține o bunătate de $5 + 5 = 10$. Funcția returnează apoi cea mai bună dintre cele două, care este `{1, 0}`. Permutarea returnată are bunătatea maximă posibilă, conform cerințelor.