TASK 1.1. COINS

Fie date M valori întregi pozitive dintre care una este 1. Există, de asemenea, un număr nelimitat de monede pentru fiecare din valorile date. Se consideră următoarea problemă: O anumită sumă S trebuie plătită folosind un număr minim de monede cu valorile date.

Este cunoscut faptul că această problemă poate fi rezolvată în anumite cazuri folosind următorul algoritm greedy: Se găseşte moneda cu cea mai mare valoare care este mai mică sau egală cu suma S după care se scade valoarea monedei găsite din S. Se contiună în aceeaşi manieră până când S devine 0. Numărul de monede folosit de algoritm pentru a reduce S la 0 pare să fie numărul minim de monede necesare.

În multe cazuri algoritmul de mai sus este eficient, dar pentru anumite seturi de valori ale monedelor şi anumite valori ale lui S algoritmul nu duce la obținerea soluției optime. De exemplu pentru setul de valori {1, 2, 5, 7, 10} şi pentru S = 14, algoritmul greedy dă o soluție cu 3 monede(14=10+2+2) pe cînd soluția minimală este cu 2 monede (14=7+7).

Astfel apare problema: pentru ce set de valori ale monedelor algoritmul greedy nu produce o soluție corectă. Scrieți un program numit **COINS**, care pentru un set dat de valori ale monedelor decide dacă există o sumă S care poate fi obținută folosind un număr de monede mai mic decât cel obținut folosind algoritmul greedy prezentat mai sus.

Programul trebuie să citească datele de la **dispozitivul standard de intrare.** Pe prima linie va fi dat numărul M de valori ale monedelor ($1 < M < 100$). Pe a doua linie de intrare de dau valorile $a_1, a_2, \ldots, a_M$ ($1=a_1 < a_2 < \ldots < a_M \leq 7000000$) separate printr-un singur spațiu. Pe a treia linie de intrare se dau două numere x şi y ($0 < x < y \leq 7000000$) separate printr-un spațiu.

Programul trebuie să tipărească pe prima linie a **dispozitivului standard de ieşire** o valoare S, $x \leq S \leq y$, pentru care algoritmul greedy nu dă soluția optimă. Pe a doua linie programul trebuie să tipărească numerele $b_1, b_2, \ldots, b_M$ de monede din fiecare valoare (în aceeaşi ordine ca la citire) folosite pentru obținerea sumei S ($S = a_1b_1 + a_2b_2 + \ldots + a_M b_M$). Numărul total de monede folosite trebuie să fie mai mic decât numărul obținut de algoritmul greedy. Dacă există mai multe soluții programul va afişa doar una dintre ele.

Se garantează că datele de intrare vor fi astfel date încât pentru cel puțin o valoare S algoritmul greedy prezentat nu va obține soluția optimă.

EXEMPLU

Input	Output
5 1 2 5 7 10 1 100	14 0 0 0 2 0

TASK 1.2. TEAM SELECTION

Olimpiada Interpeninsulară de informatică se apropie şi liderii echipei peninsulei Balkan doresc să aleagă cei mai buni concurenți din peninsulă. Din fericire liderii pot alege echipa dintre N candidați numerotați de la 1 la N ($3 \leq N \leq 500000$). Pentru a selecta cei mai buni candidați liderii au organizat trei competiții. Fiecare dintre cei N candidați a participat la toate competițiile, iar la fiecare competiție nu au existat doi concurenți cu aceleaşi rezultate. Spunem că un concurent A este *mai bun* decât concurentul B dacă A s-a clasat înaintea lui B în toate competițiile. Un concurent A este *excelent* dacă nici un concurent nu este *mai bun* decât A. Liderii echipei peninsulei Balkan doresc să afle numărul de concurenți *excelenți.*

Scrieți un program numit **`TEAM`**, care pentru un N dat şi rezultatele celor trei competiții obține numărul de concurenți *excelenți.*

Datele de intrare sunt date de la **dispozitivul standard de intrare** pe patru linii. Prima linie conține numărul N. Următoarele trei linii conțin clasamentele pentru cele trei competiții. Fiecare dintre aceste linii conține identificatorii numerici ai concurenților, separați printr-un singur spațiu, în ordinea locurilor, de la primul loc la ultimul.

Se va afişa la **dispozitivul standard de ieşire,** pe o singură linie, numărul de concurenți *excelenți.*

EXEMPLUL 1

Input	**Output**
3 2 3 1 3 1 2 1 2 3	3

Notă: Nici un concurent nu e mai bun decât ceilalți, deci toți sunt excelenți.

EXEMPLUL 2

Input	**Output**
10 2 5 3 8 10 7 1 6 9 4 1 2 3 4 5 6 7 8 9 10 3 8 7 10 5 4 1 2 6 9	4

Notă: Concurenții excelenți sunt 1, 2, 3 şi 5.

TASK 1.3. PATH FINDING

Compania *Trimoncium Soft* vrea să intre în piața de software cu un produs pentru calculatoare de buzunar cu memorie limitată, care este capabil să presteze o serie de servicii utile via Internet. Un astfel de serviciu este să găsească cea mai scurtă cale între două intersecții a unui oraş. Din cauza dificultăților tehnice aplicația ar trebui să facă cît mai puține cereri prin Internet.

Intersecțiile sunt numerotate cu numere naturale consecutive de la 1 la N ($10 \le N \le 7000$) şi poziția intersecției C este dată de o pereche de coordonate întregi X_C şi Y_C într-un sistem de coordonate ortogonal ($0 \le X_C, Y_C \le 10000$). Pot exista două sau mai multe intersecții diferite cu aceleaşi coordonate. Dacă între asemenea intersecții nu există stradă, atunci nu se poate trece direct de la o intersecție la cealaltă. De la fiecare intersercție C se poate ajunge la alte M_C intersecții ($0 \le M_C \le 5$), numite *vecini* ai lui *C,* prin străzi unidirecționale (fiecare stradă bidirecțională este prezentată ca două străzi unidirecționale). Numărul total de străzi nu depăşeste 16000. Este posibil ca două străzi să se intersecteze, dar punctul de intersecție să nu fie o intersecție de străzi, de exemplu ele pot fi plasate pe diferite nivele (fiind separate prin poduri sau prin tunele). Lungimea unei străzi este distanța dintre cele două intersecții legate de această stradă.

Scriteți un program numit **`PATH`**, care pentru o intersecție dată S şi o intersecție finală F găseşte cea mai scurtă cale de la S la F, folosind subprogramele unei biblioteci date. Numerele întregi N, S şi F sunt obținute prin cele trei argumente ale subprogramului `start`. **Acest subprogram trebuie apelat înainte de orice alt subprogram**. Coordonatele X_C, Y_C şi numărul M_C ale vecinilor intersecției C sunt obtinute prin ultimele trei argumente ale subprogramului `getXYM`, apelat cu numărul C ca prim argument. Vecinii unei intersecții C sunt obtinuți prin apelarea subprogramului `getAdj` cu un argument: numărul C. Primul apel al acestui subprogram întoarce primul vecin al lui C, al doilea apel întoarce al doilea vecin al lui C, ..., al M_C-lea apel întoarce al M_C-lea vecin. **Nu apelați acest subprogram mai mult de M_C ori**.

După ce programul vostru a găsit unul dintre cele mai scurte drumuri de la S la F el trebuie să apeleze o singură dată subprogramul `done`, dându-i un singur argument: numărul R de intersecții de-a lungul drumului (incluzând S şi F). Apoi programul trebuie să apeleze subprogramul `report` exact de R ori, transmițându-i de fiecare dată câte un argument: numărul intersecției (în ordinea în care ele apar de-a lungul drumului). Ulterior, programul trebuie să se oprească. **După apelarea subprogramului `done`, nu se poate apela nici un alt subprogram cu excepția subprogramului `report`**.

Pentru fiecare test programul va fi punctat în funcție de numărul total K de apeluri ale subprogramelor `getXYM` şi `getAdj`. Acest număr va fi comparat cu numărul J de apeluri ale aceloraşi subprograme de către pogramul juriului care rezolvă aceeaşi problemă. Dacă $K \le J$, programul va fi punctat cu 10 puncte pentru acest test. În caz contrar, va primi $2+4*(T-K)/(T-J)$ puncte, unde T este numărul total de intersecții şi străzi. Dacă programul nu găseşte una din cele mai scurte căi, dacă încalcă regulile de folosire a bibliotecii date, sau dacă răspunde corect accidental fără să asigure corectitudinea (de exemplu, dacă programul defineşte una din cele mai scurte căi fără nici un apel al subprogramelor `getXYM` şi `getAdj`), va fi punctat cu 0 puncte pentru testul respectiv.

Testele sunt formate din date reale, oferite cu bunăvoință de *DATECS GIS Center*.

EXEMPLUL 1

Apel	Rezultat
`start(N,S,F)`	`N=3 S=1 F=3`
`getXYM(1,…)`	X_1=0 Y_1=0 M_1=1
`getXYM(3,…)`	X_3=1 Y_3=1 M_3=0
`getAdj(1)`	2
`getXYM(2,…)`	X_2=0 Y_2=1 M_2=1
`getAdj(2)`	3
`done(3)`	
`report(1)`	
`report(2)`	
`report(3)`	

EXEMPLUL 2

Apel	Rezultat
`start(N,S,F)`	`N=4 S=1 F=4`
`getXYM(1,…)`	X_1=0 Y_1=0 M_1=2
`getXYM(4,…)`	X_4=1 Y_4=1 M_4=1
`getAdj(1)`	2
`getAdj(1)`	3
`getXYM(2,…)`	X_2=0 Y_2=1 M_2=1
`getXYM(3,…)`	X_3=9 Y_3=0 M_3=2
`getAdj(2)`	4
`done(3)`	
`report(1)`	
`report(2)`	
`report(4)`	

Pentru utilizatorii FreePascal

Biblioteca (`module.ppu`, `module.o`)

```
procedure start(var N,S,F: LongInt);
procedure getXYM(I: LongInt; var XI,YI,MI: LongInt);
function getAdj(I: LongInt): LongInt;
procedure done(R: LongInt);
procedure report(V: LongInt);
```

Programul `example.pas` demonstrează utilizarea bibliotecii.

Instrucțiuni: Pentru compilarea fişierului `path.pas` includeți în el operatorul

```
uses module;
```

şi executați:

```
fpc -So -O2 -XS path.pas
```

Pentru utilizatorii GNU C/C++

Biblioteca (`module.h, module.o`)

```
void start(long* N, long* S, long* F);
void getXYM(long I, long* XI, long* YI, long* MI);
long getAdj(long I);
void done(long R);
void report(long V);
```

Programul `example.c` demonstreaza utilizarea bibliotecii.

Instrucțiuni: Pentru compilarea fişierului `path.c` sau `path.cpp` puneți

```
#include "module.h"
```

în sursă şi executați:

```
gcc -O2 -static path.c module.o -lm -o path.exe
```

sau

```
gXX -O2 -static path.cpp module.o -lm -o path.exe
```

Dacă folosiți RHIDE: Atribuiți la Options->Linker configuration valoarea `module.o`.

Experimente

Pentru testarea programului, veți dispune de o versiune experimentală a bibliotecii. Această versiune citeşte harta oraşului din fişierul `path.in`. Primul rând al acestui fişier va conține numerele întregi *N*, *S* şi *F*. Al *C*-lea rând din următoarele *N* linii trebuie să conțină informația pentru intersecția *C*. Primele trei numere ale liniei trebuie vor fi X_C, Y_C şi M_C. Ele vor fi urmate de M_C numere (pe aceeaşi linie): indicii vecinilor lui C. Fişierele de intrare corespunzătoare celor două exemple sunt date mai jos. Rezultatul experimentului – dacă programul foloseşte biblioteca corect şi dacă programul returnează o cale validă – va fi afişat în fişierul `path.out`. Versiunea experimentală nu verifică dacă rezultatul este optim şi nici nu evaluează numărul de apeluri ale subprogramelor `getXYM` şi `getAdj`.

Exemple pentru fişierul de intrare `path.in`

Sample 1

```
3 1 3
0 0 1 2
0 1 1 3
1 1 0
```

Sample 2

```
4 1 4
0 0 2 2 3
0 1 1 4
9 0 2 1 4
1 1 1 3
```