TASK 1.3. PATH FINDING

The company *Trimoncium Soft* would like to enter the software market with a product for limited-memory pocket computers, which is able to provide a series of useful services via Internet. One such service is to find the shortest path between two crossroads of a given city. Because of technical difficulties the application should use as few Internet requests as possible.

The crossroads are numbered with consecutive integers from 1 to N ($10 \le N \le 7000$) and the location of a crossroad C is defined by a pair of integer coordinates X_C and Y_C in an orthogonal coordinate system ($0 \le X_C, Y_C \le 10000$). It is possible to have two or more crossroads with the same coordinates. If there is no street between two such crossroads it is impossible to pass from one to the other. From each crossroad C it is possible to reach M_C other crossroads ($0 \le M_C \le 5$), called *neighbors* of C, through straight one-way streets (each two-way street is presented as two one-way streets). The total number of streets is not greater than 16000. It is possible that two streets intersect, but the intersection point is not a crossroad, e.g. they could be on different levels (passing on bridges or going through tunnels). The length of a street is the distance between the two crossroads connected by that street.

Write a program named **`PATH`**, which for a given starting crossroad S and a final crossroad F finds one of the shortest paths from S to F, using the subprograms of a provided library. The integers N, S and F are obtained through the three arguments of the subprogram `start`. **This subprogram must be invoked before any other subprogram**. The coordinates X_C, Y_C and the number M_C of the neighbors of the crossroad C are obtained through the last three arguments of the subprogram `getXYM`, invoked with the number C as a first argument. The neighbors of a crossroad C are obtained by calling the subprogram `getAdj` with one argument: the number C. The first call of this subprogram returns the first neighbor of C, the second call returns the second neighbor of C, and so on, the M_C-th call returns the M_C-th neighbor. **You should never call this subprogram more than M_C times**.

When your program has found one of the shortest paths from S to F it has to call the subprogram `done` once, giving it one argument: the number R of crossroads along the path (including S and F). Then the program has to call the subprogram `report` exactly R times, giving it one argument each time: the number of the consecutive crossroad (in the order in which they appear along the path). Then the program has to stop. **After the invocation of the subprogram `done`, no other subprograms except `report` should be called.**

For each test case your program will be graded by the total number K of calls to the `getXYM` and `getAdj` subprograms. This number will be compared with the number J of calls to the same subprograms by a program of the Jury that solves the same task. If $K \le J$, your program will be assigned 10 points for this test case. Otherwise, you will get $2+4*(T-K)/(T-J)$ points, where T is the total number of crossroads and streets. If the program has found a path which is not any of the shortest paths, or if it violates the rules for using the provided library, or if it accidentally outputs the correct answer without assuring its correctness (for example, if it outputs the shortest path without any calls to the subprograms `getXYM` and `getAdj`), then you will be assigned 0 points for the corresponding test case.

The test cases are composed of real-world data, kindly provided to us by DATECS GIS Center.

EXAMPLE 1

Call	Obtained
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)	

EXAMPLE 2

Call	Obtained
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)	

For users of FreePascal

Library (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);
```

The program example.pas demonstrates the usage of the library.

Instructions: To compile the file path.pas include in it the operator

```
uses module;
```

and execute:

```
fpc –So –O2 –XS path.pas
```

For users of GNU C/C++

Library (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);
```

The program example.c demonstrates the usage of the library.

Instructions: To compile the files path.c or path.cpp put

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

in the source code and execute:

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

or

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

If using RHIDE: Assign to Options->Linker configuration the value module.o.

Experiments

In order to be able to test your program, you will be provided with an experimental version of the library. This version reads the map of the city from the file `path.in`. The first line of this file contains the integers N, S and F. The C-th line of the next N lines contains the information for the C-th crossroad. The first three numbers on the line are X_C, Y_C and M_C. They are followed by M_C numbers (on the same line) – the neighbors of C. The corresponding input files for Example 1 and Example 2 are given below. The results of the experiment – whether your program uses the library correctly and whether it reports a valid path – will be printed to the file `path.out`. The experimental version does not check whether your result is optimal and does not evaluate the number of calls to the `getXYM` and `getAdj` subprograms.

Samples for the `path.in` input file

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
```