Задача 1.3. НАМИРАНЕ НА ПЪТ

Компанията *Тримонциум Софт* иска да пусне на пазара джобен компютър с ограничена памет, който, обаче, чрез връзка с Интернет да може да предоставя полезни услуги. Една такава услуга е намиране на най-къс път между две кръстовища на даден град. По технически причини приложението трябва да прави колкото може по-малко запитвания по Интернет.

Кръстовищата са номерирани последователно с числата от 1 до N ($10 \le N \le 7000$), а местоположението на кръстовището C се задава с целочислени координати X_C и Y_C в правоъгълна координатна система ($0 \le X_C, Y_C \le 10000$). Възможно е да има две или повече кръстовища с едни и същи координати. Ако няма улица между две такива кръстовища, не е възможно да се премине от едното на другото. От всяко кръстовище C може да се стигне до M_C други кръстовища ($0 \le M_C \le 5$), наричани *съседи* на C, чрез прави еднопосочни улици. (двупосочните улици са представени като две еднопосочни). Общият брой на улиците не надвишава 16000. Възможно е две улици да се пресичат, без пресечната им точка да е кръстовище, например те могат да бъдат на различни нива (когато едната минава по мост или през тунел). Дължина на една улица е разстоянието между кръстовищата, които тя свързва.

Напишете програма **`PATH`**, която по зададени начално кръстовище S и крайно кръстовище F намира най-къс път от S до F, използвайки предоставените библиотечни подпрограми. Числата N, S и F се получават при извикване на подпрограмата `start` с три аргумента. **Тя трябва да бъде извикана преди всяка друга подпрограма.** Координатите X_C и Y_C и броят M_C на съседите на кръстовището C, се получават в последните три аргумента на подпрограмата `getXYM`, като първият аргумент е номерът на кръстовището C. Номерата на съседите на кръстовище C се получават като резултат от извикване на подпрограмата `getAdj` с единствен аргумент C. Първото такова извикване връща номера на първия съсед на C, второто извикване – на втория съсед на C и т.н., M_C-тото извикване връща номера на M_C-тия съсед. **Никога не извиквайте тази подпрограма повече от M_C пъти.**

Когато програмата намери най-късия път от S до F, тя трябва да извика само един път подпрограмата `done` с единствен аргумент броя R на кръстовищата по намерения път (включително S и F). След това тя трябва да извика подпрограмата `report` точно R пъти, като всеки път задава в единствения си аргумент номера на поредното кръстовище (по реда, в който се срещат в най-късия път). Накрая програмата трябва да прекрати изпълнението си.
След извикването на подпрограмата `done`, не трябва да се извикват други библиотечни подпрограми, освен `report`.

За всеки тест програмата ще бъде оценена според общия брой K на извикванията на подпрограмите `getXYM` и `getAdj`. Той ще бъде сравнен с броя J на извикванията на тези подпрограми от програма на журито за същата задача. Ако $K \le J$, ще бъдат присъдени 10 точки за съответния тест. В противен случай ще бъдат присъдени $2+4*(T-K)/(T-J)$ точки, където T е общият брой на улиците и кръстовищата. Ако програмата не намира най-къс път, не спазва правилата за работа с библиотеката или случайно даде правилен отговор, чиято оптималност не е проверила (например, извежда най-къс път без да е извиквала подпрограмите `getXYM` и `getAdj`), тя ще получи 0 точки за съответния тест.

Тестовете са съставени с реални данни, любезно предоставени ни от ДАТЕКС ГИС Център.

ПРИМЕР 1

ПРИМЕР 2

Извиквате

Получавате

Извиквате Получавате

```
start(N,S,F) N=3 S=1 F=3    start(N,S,F) N=4 S=1 F=4

getXYM(1,…) X1=0 Y1=0 M1=1     getXYM(1,…)    X1=0 Y1=0 M1=2

getXYM(3,…) X3=1 Y3=1 M3=0   getXYM(4,…)    X4=1 Y4=1 M4=1

getAdj(1)      2         getAdj(1)      2

getXYM(2,…) X2=0 Y2=1 M2=1   getAdj(1)      3

getAdj(2)    3         getXYM(2,…)  X2=0 Y2=1 M2=1

done(3)                              getXYM(3,…)  X3=9 Y3=0 M3=2

report(1)                          getAdj(2)    4

report(2)                          done(3)

report(3)                          report(1)

               report(2)

               report(4)
```

Библиотека за FreePascal (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);
```

Програмата example.pas е пример за използване на тази библиотека.

Инструкции: За да компилирате вашия файл path.pas, включете оператора

uses module;

в изходния текст и компилирайте с

fpc –So –O2 –XS path.pas

Библиотека за GNU C/C++ (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);

Програмата example.c е пример за използване на тази библиотека.

Инструкции: За да компилирате вашия файл path.c или path.cpp, използвайте #include "module.h" в изходния текст и компилирайте с:

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

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

За пишещите на C/C++ в RHIDE: Непременно задайте на Options->Linker configuration стойност module.o.

Експерименти: За да тествате програмата си ще разполагате с експериментална версия на действителния модул. Тя чете картата на града, с която искате да проверите програмата си от файла `path.in`. На първия ред на файла се записани трите цели числа *N*, *S* и *F*. Следват *N* реда, като на *C*-тия от тях е записана информацията за кръстовището с номер *C*. Първите три числа на реда са X_C, Y_C и M_C, а следващите M_C числа на реда са номерата на "съседите" на *C*. Входните файлове за двата примерни теста са дадени по-долу. Резултатът от експеримента – дали програмата ви борави валидно с библиотеката и дали извежда валиден път – ще бъде изведен във файла `path.out`. Модулът за експерименти не проверява минималността на вашия път и не оценява броя на извикванията на подпрограмите `getXYM` и `getAdj`.

Примерен вход 1

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

Примерен вход 2

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

```
1 1 1 3
```