Povezivanje gradova (gradovi)

Mala Lambda Kviksortić je putovala kroz međudimenzioni procjep i došla u čudan svijet. Došla je u ogromnu pokrajinu sa mnogo velikih gradova. Njen dolazak je svečano dočekan, jer su se svi nadali da će im ona razriješiti dugogodišnji problem.

Naime, u njihovoj velikoj pokrajini gradovi nisu međusobno dobro povezani, a čitav socijalni sistem im je baziran na pisanom zakonu. Jedno od ograničenja koje taj zakon nameće je da svaki grad smije imati samo jedan komunikacioni stub, te da taj stub mora biti postavljen između najbliže dvije kuće u tom gradu. Pored toga, kontrolni broj stuba u regionu se određuje na osnovu daljine dvaju kuća između kojih je postavljen i jednak je toj udaljenosti. Dalje, svaki grad ima i svoj redni broj pod kojim je zaveden u registar gradova.

Problem u organizaciji je taj što se stubovi mogu povezati samo u rastućem redoslijedu kontrolnih brojeva, ali i redoslijedom kojim su zavedeni u registar gradova. Prilikom povezivanja smiju se preskočiti neki gradovi, ali nikada se smiju poremetiti dva redoslijeda.

Mještani hoće da povežu što veći broj gradova, kako bi što bolje mogli međusobno komunicirati. Lambda Kviksortić zna da se može desiti da svi gradovi ne budu povezani, ali ona hoće da poveže što je moguće veći broj njih, a da pri tom ne prekrši mjesni zakon. Pomozite joj da to uradi.

Zadatak

Napisati tri procedure/funkcije kojima građani mogu zadati i riješiti opisani problem.

Procedura *Inicijalizacija*$(n, m[])$ poziva se samo jednom na početku izvršavanja programa. Parametar n je prirodan broj koji predstavlja broj gradova. Redni brojevi gradova u registru gradova su $0, 1, \ldots, n-1$. Drugi parametar $m[] = (m_0, m_1, \ldots m_{n-1})$ je niz dužine n sastavljen od prirodnih brojeva u kojem svaki od elemenata m_i predstavlja broj kuća u i-tom gradu, gdje je i redni broj grada u registru.

Nakon procedure *Inicijalizacija*, poziva se procedura *PostaviKucu* tačno $m = m_0 + m_1 + \cdots + m_{n-1}$ puta, i to m_0 puta sa parametrom $i = 0$, m_1 puta sa parametrom $i = 1$, ..., m_k puta sa parametrom $i = k$. Odnosno, za svaku kuću bit će po jedan poziv ove funkcije. Parametar i je cio broj $(0 \leq i < n)$ koji predstavlja redni broj grada u kojem se kuća nalazi, a parametri x i y su cijeli brojevi $(-10.000 \leq x, y \leq 10.000)$ koji predstavljaju koordinate te kuće. Koordinate su uvijek lokalne za svaki od gradova, odnosno koordinate za jedan grad nemaju nikakve veze sa koordinatama iz drugog grada.

Funkcija *Povezi*() poziva se nakon svih poziva procedure *PostaviKucu* i treba da vrati najveći broj gradova koje je moguće povezati, a da se pri tom ne prekrši mjesni zakon.

Podzadatak 1 (13 bodova): $1 \leq n \leq 10$, $2 \leq m_i \leq 10$

Podzadatak 2 (26 bodova): $1 \leq n \leq 5.000$, $2 \leq m_i \leq 30$

Podzadatak 3 (30 bodova): $1 \leq n \leq 100$, $2 \leq m_i \leq 2.000$

Podzadatak 4 (31 bod): $1 \leq n \leq 30.000$, $2 \leq m_i \leq 15$

Primjer

Primjer izvršavanja programa je sljedeći. Najprije se poziva funkcija *Inicijalizacija*:

- *Inicijalizacija* $(4, [4, 2, 3, 3])$ - postoje 4 grada, koji redom imaju 4, 2, 3 i 3 kuće.

U gradu 0 postoje 4 kuće, pa slijede 4 poziva funkcije *PostaviKucu* sa parametrom $i = 0$:

- *PostaviKucu* $(0, 3, 2)$ - u gradu 0 postoji kuća na koordinatama $(3, 2)$,

- *PostaviKucu* $(0, 3, 4)$ - u gradu 0 postoji kuća na koordinatama $(3, 4)$,
- *PostaviKucu* $(0, 5, 5)$ - ... ,
- *PostaviKucu* $(0, 2, 2)$,

U gradu 1 postoje 2 kuće, pa slijede 2 poziva funkcije *PostaviKucu* sa parametrom $i = 1$:

- *PostaviKucu* $(1, 0, 5)$ - u gradu 1 postoji kuća na koordinatama $(0, 5)$,
- *PostaviKucu* $(1, 5, 0)$.

Slično je i za gradove 2 i 3:

- *PostaviKucu* $(2, 1, 1)$,
- *PostaviKucu* $(2, 0, 0)$,
- *PostaviKucu* $(2, 2, 2)$,
- *PostaviKucu* $(3, 0, 0)$,
- *PostaviKucu* $(3, 2, 1)$,
- *PostaviKucu* $(3, 5, 10)$.

Na kraju se poziva Vaša funkcija *Povezi*, koja, ako radi ispravno, treba da vrati broj 3. *Obrazloženje.* Nije teško vidjeti da su u gradu 0 najbliže kuće sa koordinatama $(3, 2)$ i $(2, 2)$ čija je udaljenost $d_{0,min} = 1$. U gradu 1, postoje samo dvije kuće sa koordinatama $(0, 5)$ i $(5, 0)$, pa je $d_{1,min} = \sqrt{50} = 5\sqrt{2} \approx 7,071$. U gradu 2, kuće sa koordinatama $(1, 1)$ i $(0, 0)$ jednako su udaljene kao i kuće sa koordinatama $(1, 1)$ i $(2, 2)$, pa je $d_{2,min} = \sqrt{2} \approx 1,414$. I konačno, u gradu 3, najbliže su kuće sa koordinatama $(0, 0)$ i $(2, 1)$ čija je udaljenost $d_{3,min} = \sqrt{5} \approx 2,236$. Komunikacioni tornjevi moraju se postaviti između ovih najbližih kuća, a njihovi kontrolni brojevi će biti jednaki ovim najmanjim udaljenostima. Na taj način dobijemo niz kontrolnih brojeva $1\,;\, 7,071\,;\, 1,414\,;\, 2,236$ redom za gradove $0\,;\, 1\,;\, 2\,;\, 3$. Gradove moramo povezati u rastućem redoslijedu kontrolnih brojeva, ali i redom kojim su zavedeni u registar. Treba povezati što je moguće više gradova. Jasno je da je najbolje rješenje povezati gradove $0 \to 2 \to 3$, odnosno mogu se povezati najviše 3 grada.

Detalji implementacije

Sa servera za takmičenje možete preuzeti pripremljena okruženja (`gradovi_c.zip`, `gradovi_cpp.zip` ili `gradovi_pas.zip`) sa osnovnim fajlovima za C/C++ i Pascal.

Ukoliko koristite C ili C++ napišite funkcije sa prototipovima:

```
void Inicijalizacija(int n, int m[]);
void PostaviKucu(int i, int x, int y);
int Povezi();
```

u fajlu `gradovi.[c/cpp]`.

Ukoliko koristite Pascal napišite dvije procedure i funkciju sa prototipovima:

```
procedure Inicijalizacija(n : LongInt; var m : Array of LongInt);
procedure PostaviKucu(i : LongInt; x : LongInt; y : LongInt);
function Povezi() : LongInt;
```

u fajlu `gradovi.pas`.

Samo unutar ovog fajla treba da implementirate svoje rješenje. Pri tome smijete koristiti i druge pomoćne funkcije koje ste vi napisali, te standardna zaglavlja/biblioteke odabranog programskog jezika i funkcije iz ovih biblioteka,

kao i globalne varijable. Ne smijete ni na koji način vršiti interakciju sa standardnim ulazom/izlazom niti sa bilo kojom datotekom.

U pripremljenom okruženju nalazi se fajl `grader.[c/cpp/pas]` koji testira ispravnost rada[1] funkcije koju ste napisali na javni primjer. Kada šaljete svoje rješenje, šaljete samo fajl `gradovi.[c/cpp/pas]`, dok komisija koristi svoj `grader.[c/cpp/pas]` koji nije javni. U skladu s tim, slobodni ste da modifikujete `grader.[c/cpp/pas]` i prilagođavate ga svojim potrebama u svrhu testiranja na lokalnom računaru.

Ukoliko koristite *Code::Blocks* u pripremljenim okruženjima možete naći i odgovarajuće projekte sa podešenim parametrima za prevođenje. "*Release build*" u potpunosti odgovara parametrima za prevođenje koji su na serveru za takmičenje, dok "*Debug build*" ima isključene optimizacije i uključene simbole za debagiranje.

Ukoliko koristite *FreePascal IDE*, dovoljno je da pokrenete prevođenja fajla `grader.pas` dok je u istom folderu fajl `gradovi.pas`. Na serveru za takmičenje postavljeni su sljedeći parametri za prevođenje: `-dEVAL -vw -XS -O2`.

Ukoliko ne koristite *Code::Blocks*, odnosno *FreePascal IDE*, u okruženjima se nalaze i fajlovi `prevedi_[c/cpp/pas].sh` koje možete koristiti za prevođenje svojih programe, a koje pozivate iz terminala komandom `sh prevedi_[c/cpp/pas].sh` iz odgovarajućeg foldera.

[1] Fajl grader.[c/cpp/pas] koji je javno dostupan testira samo ispravnost bez postavljanja ograničenja na vrijeme izvršavanja i iskorištenu memoriju.