Submerging Islands

Innen: Algowiki

Tartalomjegyzék

- 1 Feladat
  - 1.1 Az eredeti feladat
- 2 Megoldási ötletek
  - 2.1 Hibás megoldási ötletek és ellenpéldák
  - 2.2 Helyes, de lassú megoldások
- 3 Megoldás
  - 3.1 Fontos gondolat
  - 3.2 Részletes megoldás
- 4 Komplexitás
- 5 Implementáció

Feladat

Vice City városa N db szigetre épült, melyeket M db híd köt össze. A város lakói attól tartanak, hogy egy nap a víz egyes szigeteket ellep majd. Ha egy sziget víz alá kerül, az megszüntetheti a város egyes részei közti összeköttetést. A feladatunk megállapítani, hogy hány olyan sziget van, amelyek megszüntetnék a város összeköttetettségét, ha víz alá kerülnének. Tudjuk, hogy eredetileg teljes Vice City össze van kötve.

A bemenet egymás után több részfeladatot is tartalmazhat. Egy részfeladat első sorában megkapjuk N $(1 \leq N \leq 10^4)$, majd M $(1 \leq M \leq 10^5)$ értékét, szóközökkel elválasztva. Az ezt követő M sorban két szám, U_i és V_i $(1 \leq U_i, V_i \leq N)$ található, mely az U_i és V_i sorszámú szigetek közti hidat jelöli. A bemenet végét az N=0, M=0 értékpár jelzi. A kimenetre soronként a részfeladatok eredményeit (egy szám) kell írni.

Az eredeti feladat

SPOJ - Submerge (https://www.spoj.com/problems/SUBMERGE/)

Megoldási ötletek

Hibás megoldási ötletek és ellenpéldák

Hibás megközelítés abból kiindulni, hogy ha találunk kört a gráfban, akkor annak bármelyik pontját kivéve a gráf összefüggő marad, vagyis ezeket a szigeteket biztosan nem kell számolni. Gondoljunk csak egy olyan gráfra, amely egyetlen olyan körből áll, melynek minden csúcsából egy további olyan sziget is elérhető, ami csak belőle érhető el.

Helyes, de lassú megoldások

Brute force: minden egyes C csúcsra indítsunk mélységi bejárást C egy tetszőleges szomszédjából, és jegyezzük fel, összesen hány csúcsot értünk el így (C-t eleve látogatottnak állítsuk be, és számoljuk el 1-nek). Ha ez az érték N-nél kevesebb, az azt jelenti, hogy a gráf egyes csúcsait csak C egy másik szomszédjából, vagyis C-n keresztüli további bejárásokkal lehetne elérni, vagyis C egyike a keresett csúcsoknak. N db csúcsra ez a módszer $O(N * (N + M))$ futási időt eredményezne, ennél tudunk jobbat.

Megoldás

Fontos gondolat

A megoldáshoz meg kell számolnunk az elvágó pontokat a gráfban.

Részletes megoldás

A csúcsokat tárolhatjuk tömbök, vektorok, vagy akár struktúrák formájában is. Minden csúcsról tároljuk el a beolvasás során feljegyzett szomszédait (irányítatlan a gráf, ezért mindkét irányban), az elvágó pontok kiszámításához szükséges belépési időt és low-értéket, valamint azt, hogy az adott pontot észleltük-e már elvágó pontként (hogy ne számoljuk többször).

Állítsunk be egy kezdeti számlálót 0-ra, majd indítsuk el az elvágpont-kereső algoritmust egy tetszőleges csúcsból (mondjuk az 1-ből). Minden alkalommal, amikor a bejárás során elvágó ponthoz érünk, vizsgáljuk meg, hogy észleltük-e már az adott csúcsot, és ha nem, állítsuk észleltté és növeljük a számlálónkat. A megoldás a bejárás végén a számlálóban lesz tárolva.

Komplexitás

Az elvágó pontok keresőalgoritmusa $O(N+M)$ futási idejű minden részfeladatban. Az gráf tárolásához $O(N+M)$ tárhely szükséges.

Implementáció

[becsuk]

```
#include <iostream>
#include <vector>
using namespace std;


// Egy varost reprezentalo struktura
struct varos
{
    int be, low;
    bool biztonsagos;
    vector<varos*> szomszedok;
};


int db;
int ido;
// Keressunk elvago pontokat
void keres(varos* forras, varos* v)
{
    // belepesi ido es kezdeti low-ertek beallitasa
    v->low = v->be = ido++;
    int faelek = 0;
    // jarjunk be minden sz szomszedot
    for (varos* sz : v->szomszedok)
    {
        if (sz->be == -1)
        {
            // meg nem jartuk be: v es sz kozott fael van
            keres(v, sz);
            if (sz->low < v->low)
                v->low = sz->low;
            if (sz->low >= v->be && forras != nullptr)
            {
                // v elvago pont: ha meg biztonsagoskent van jelolve, mar tudjuk, hogy nem az
                if (v->biztonsagos)
                {
                    // v egy meg nem eszlelt elvago pont
                    v->biztonsagos = false;
                    ++db;
                }
```

```
41              }
42              ++faelek;
43          }
44          else if (sz != forras)
45          {
46              // mar bejartuk: v es sz kozott vissza-el van
47              if (sz->be < v->low)
48                  v->low = sz->be;
49          }
50      }
51      // ha v a gyoker, kulon vizsgalni kell
52      if (forras == nullptr && faelek > 1 && v->biztonsagos)
53      {
54          // a gyoker is elvago pont
55          v->biztonsagos = false;
56          ++db;
57      }
58  }
59
60
61  int main()
62  {
63      // elso reszfeladat
64      int N, M;
65      cin >> N >> M;
66      while (N != 0 || M != 0)
67      {
68          // letrehozzuk a varosok tombjet
69          varos* varosok = new varos[N];
70          for (int i = 0; i < N; ++i)
71          {
72              varosok[i].be = varosok[i].low = -1;
73              varosok[i].biztonsagos = true;
74          }
75
76          // beolvassuk a hidakat, elkeszitjuk a grafot
77          for (int i = 0; i < M; ++i)
78          {
79              int u, v;
80              cin >> u >> v;
81              --u; --v;
82              varosok[u].szomszedok.push_back(&varosok[v]);
83              varosok[v].szomszedok.push_back(&varosok[u]);
84          }
85
86          // keresunk elvago pontokat es kiirjuk az eredmenyt
87          ido = 0;
88          db = 0;
89          keres(nullptr, &varosok[0]);
90          cout << db << endl;
91
92          // kovetkezo reszfeladat
93          delete[] varosok;
94          cin >> N >> M;
95      }
96      return 0;
97  }
```

A lap eredeti címe: „https://algowiki.miraheze.org/w/index.php?title=Submerging_Islands&oldid=1343”