Gráfalgoritmusok: összefüggőség

Horváth Gyula

horvath@inf.elte.hu

1. Elvágópontok és hidak

1.1. definíció. *Egy* $G = (V,E)$ *összefüggő irányítatlan gráf* $p \in V$ *pontját elvágópontnak nevezzük, ha* p*-t (és a hozzá kapcsolódó minden élet) törölve, a gráf nem lesz összefüggő.*

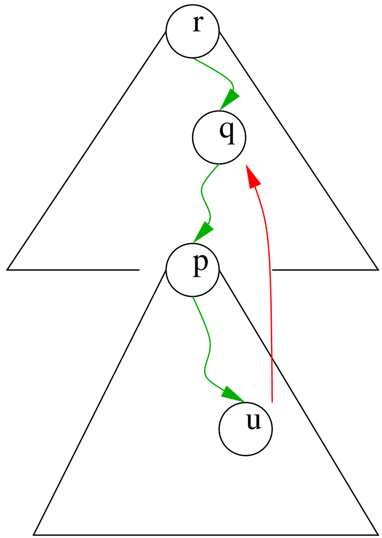
1.1. Elvágópontok

Legyen $G = (V,E)$ irányítatlan gráf.
Feladat: határozzuk meg G mindazon pontját, amelyek benne vannak legalább egy körben!

1.2. Állítás. *Irányítatlan gráf bármely mélységi bejárása esetén minden él vagy faél, vagy visszaél.*

1.3. Állítás. *Egy* $G = (V,E)$ *irányítatlan gráf* $p \in V$ *pontja akkor és csak akkor van körben, ha van olyan* $u \in V$ *és* $v \in V$ *pont, hogy* $p = u$ *vagy* u *leszármazottja* p*-nek, és* (u,v) *visszaél* G *valamely* MFF *mélységi feszítőfájában.*

Legyen $G = (V, E)$ irányítatlan gráf, és tekintsük egy r-gyökerű MFF mélységi feszítőfáját. Legyen $D(p)$ a p elérési ideje a mélségi bejárás során. Definiáljuk az $L : V \to \mathbb{N}$ függvényt a következőképpen:

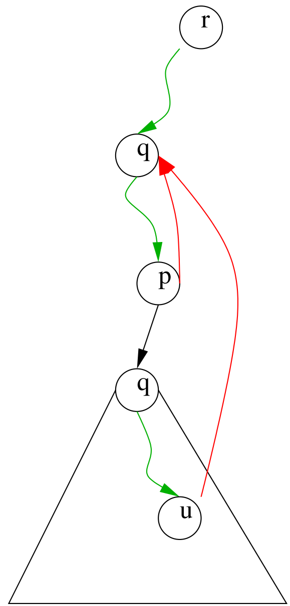


1. ábra.

$$L(p) = \min \begin{cases} D(p) \\ D(q): \text{ van olyan } u \text{ leszármazottja } p\text{-nek, hogy } u \to q \text{ visszaél} \end{cases}$$

1.4. Állítás. *Az L függvény kiszámítható az élek számával arányos időben ($O(E)$) a mélységi bejárás során.*

$$L(p) = \min_{p \to q} \begin{cases} D(p) & \\ D(q): & p \to q \textit{ visszaél} \\ L(q): & p \to q \textit{ faél} \end{cases}$$



2. ábra.

```
void MelyBejar(int p){
//Global: G, ido, D,Szin,Apa,L
    D[p]=ido++;
    L[p]=ido;
    Szin[p]=Szurke;
    for (int q:G[p])
        if (Szin[q]==Feher){ //p->q faél
            Apa[q]=p;
            MelyBejar(q);
            L[p]=min(L[p],L[q]);
        }else if(Apa[p]!=q && D[q]<L[p]) //visszaél
            L[p]=D[q];
}
```

1.5. Állítás. *Irányítatlan gráf bármely p pontja akkor és csak akkor van benne legalább egy körben, ha $L(p) < D(p))$ vagy p-nek van olyan u fia a mélységi feszítőfában, hogy $L(u) \leq D(p)$.*

1.6. Állítás. *Irányítatlan gráf bármely p pontja akkor és csak akkor elvágópont, ha $p \neq r$ és van olyan q fia a mélységi feszítőfában, hogy $L(q) \geq D(p))$.*

1.7. Állítás. *Irányítatlan gráf esetén a mélységi feszítőfa gyökere akkor és csak akkor elvágópont, ha egynél több fia van a feszítőfában.*

```
void Elvágopntok(int p){
//Global: G, ido, D,Szin,L, ElvagoP
    D[p]=ido++;
    L[p]=ido;
    Szin[p]=Szurke;
    int fiuk=0;
    for (int q:G[p])
        if (Szin[q]==Feher){ //p->q faél
            fiuk++;
            Apa[q]=p;
            MelyBejar(q);
            L[p]=min(L[p],L[q]);
            if (D[p]>1 && L[q]>=D[p])
                ElvagoP[p]=true;
        }else if(Apa[p]!=q && D[q]<L[p]) //visszaél
            L[p]=D[q];

    if(D[p]==1 && fiuk>1)
        ElvagoP[p]=true;
}
```

1.2. Hidak

1.8. definíció. *Egy irányítatlan* $G = (V, E)$ *gráf* $(p, q) \in E$ *éle akkor és csak akkor híd, ha az élet elhagyva a gráf nem lesz összefüggő.*

1.9. Állítás. *Egy irányítatlan* $G = (V, E)$ *gráf* $(p, q) \in E$ *éle akkor és csak akkor híd, ha nincs benne egyetlen körben sem.*

1.10. Állítás. *Egy irányítatlan* $G = (V, E)$ *gráf* $(p, q) \in E$ *éle akkor és csak akkor híd, ha valamely mélységi feszítőfa szerint faél, és számított* L *fügvénnyel teljesül, hogy* $L(q) = D(q)$.

1.11. következmény. *Egy gráf minden hídja az élek számával arányos időben kiszámítható.*

```
//Irányítatlan gráf hidjai
#include <bits/stdc++.h>
#define maxN 10001
using namespace std;
struct El{
    int p,q;
    El(){};
    El(int x, int y){p=x; q=y;}
};
int n;//pontok száma
vector<int> G[maxN];

void BeOlvas(){
//Global:n,G
    int m,p,q;
    scanf("%d %d", &n,&m);
    for (int i=0; i<m; i++){
        scanf("%d %d", &p,&q);
        G[p].push_back(q);
        G[q].push_back(p);
    }
}
```

```
enum Paletta {Feher, Szurke, Fekete};
Paletta Szin[maxN];
int D[maxN];
int L[maxN];
int Apa[maxN];
vector<El> Hidak;
int ido;

void MelyBejar(int p){
//Global: G, ido, D,Szin,L, Hidak
    D[p]=++ido;
    L[p]=ido;
    Szin[p]=Szurke;
    for (int q:G[p])
        if (Szin[q]==Feher){ //p->q faél
            Apa[q]=p;
            MelyBejar(q);
            L[p]=min(L[p],L[q]);
            if (L[q]==D[q])
                Hidak.push_back(El(p,q));
        }else if(Apa[p]!=q && D[q]<L[p]) //visszaél
            L[p]=D[q];
}
```

```
int main (){
    BeOlvas();
    ido=0;
    for(int p=1;p<=n;p++) Szin[p]=Feher;
    for(int p=1;p<=n;p++)
        if(Szin[p]==Feher)
            MelyBejar(p);

    printf("%d\n",Hidak.size());
    for (El e: Hidak)
            printf("%d %d\n",e.p,e.q);
}
```

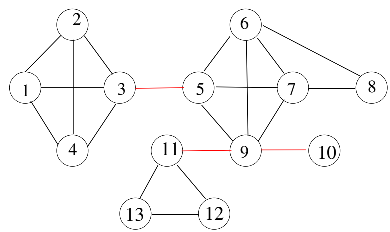
2. Kétszeresen összefüggő komponensek

2.1. definíció. *Azt mondjuk, hogy egy* $G = (V,E)$ *irányítatlan gráf 2-pont-összefüggő (2-pö), ha* $|V| > 1$*, összefüggő és bármely pontját törölve a gráf továbbra is összefüggő marad.*

2.2. Állítás. ***Alternatív definíció:*** $G = (V,E)$ *irányítatlan gráf 2-pö, ha* $|V| > 1$ *és bármely két éle között van két pont-diszjunkt út.*

2.3. definíció. *Azt mondjuk, hogy egy* $G = (V,E)$ *irányítatlan gráf 2-él-összefüggő (2-éö), ha* $|V| > 1$*, összefüggő és bármely élét törölve a gráf továbbra is összefüggő marad.*

2.4. Állítás. ***Alternatív definíció:*** $G = (V,E)$ *irányítatlan gráf 2-éö, ha* $|V| > 1$ *és bármely két pontja között van két él-diszjunkt út.*



3. ábra.

Példa bemenet és kimenet

bemenet

```
13 18
1 2
1 3
1 4
2 3
3 4
3 5
5 6
5 7
5 9
6 7
6 9
6 8
7 8
7 9
9 10
9 11
11 13
11 12
12 13
```

kimenet

```
3
12-11 13-12 11-13
9-6 9-5 7-9 7-5 6-7 5-6 6-8
4-1 3-4 3-1 2-3 1-2
```

```
//Irányítatlan gráf kétszeresen összefüggő (biconnected) komponensek
#include <bits/stdc++.h>
#define maxN 10001
using namespace std;
struct El{
    int p,q;
    El(){};
    El(int x, int y){p=x; q=y;}
};
int n;//pontok száma
vector<int> G[maxN];
void BeOlvas();

enum Paletta {Feher, Szurke, Fekete};
Paletta Szin[maxN];
int D[maxN];
int L[maxN];
int Apa[maxN];
stack<El> V;
vector<vector<El>> H;
int ido;
```

```
void MelyBejar(int p){
//Global: G, ido, D,Szin,L, V,H
      D[p]=++ido;
      L[p]=ido;
      Szin[p]=Szurke;
     for (int q:G[p]){
        if (Szin[q]==Feher){ //p->q faél
           V.push(El(p,q));
           Apa[q]=p;
           MelyBejar(q);
           L[p]=min(L[p],L[q]);
           if (L[q]>=D[p]){ //találtunk egy komponenst
              if (V.top().p==p && V.top().q==q){ //p-q híd
                 V.pop();
              }else{
                 El pq;
                 vector<El> Hp;
                 do{
                    pq=V.top(); V.pop();
                    Hp.push_back(pq);
                 }while(pq.p!=p || pq.q!=q);
                 H.push_back(Hp);
              }
           }
```

```
        }else if (q!=Apa[p] && Szin[q]==Szurke){ //p->q visszaél
            L[p]=min(L[p],D[q]);
            V.push(El(p,q));
        }
    }
    Szin[p]=Fekete;
}
```

```
void Komponensek(){
//global: G, H
    for (int p=1; p<=n; p++) Szin[p]=Feher;
    ido=0;
    for (int p=1; p<=n; p++)
        if (Szin[p]==Feher) MelyBejar(p);
    vector<El> Hp;
    //a veremben maradt komponens kiírása
    while(V.size()>0){
        El pq=V.top(); V.pop();
        Hp.push_back(pq);
    }
    if (Hp.size()>0) H.push_back(Hp);
}
```

```
int main (){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    BeOlvas();
    Komponensek();
    cout<<H.size()<<endl;
    for (vector<El> komp: H){
        for(El e : komp)
            cout<<e.p<<"-"<<e.q;
        cout<<endl;
    }
}
void BeOlvas(){
//Global:n,G
    int m,p,q;
    cin>>n>>m;
    for (int i=0; i<m; i++){
        cin>>p>>q;
        G[p].push_back(q);
        G[q].push_back(p);
    }
}
```