# Elvágó pontok és hídélek

Innen: Algowiki

## Tartalomjegyzék



## Bevezető

Nézzük a következő feladatot:

Minden számítógépes hálózat csomópontokból és bizonyos csomópontpárok között kiépített közvetlen kétirányú adatátvitelt biztosító kommunikációs vonalakból épül fel. Adott A és B csomópont esetén az olyan P csomópontot (P□A és P□B), amely meghibásodása esetén nem lehet adatátvitel A és B között, A−B-kritikus csomópontnak nevezzük. Írj programot, amely adott A és B csomópontra kiszámítja a hálózat összes A−B-kritikus csomópontját!

(Mester, Haladó feladatok, Mélységi bejárás 27. Kritikus csomópontok)

Bizonyos problémáknál előfordulhat, hogy egy összefüggő irányítatlan gráfban olyan pontokat keresünk, amelyek eltávolításával a gráf egynél több komponensre szétesik. Van, hogy az élek között keresünk ugyanilyet.

# Ötlet

Egy gráf mélységi bejárásakor a bejárás során "használt" (faélek) és a "nem használt" (vissza-él, előre-él és kereszt-él) éleket vizsgálva, az alábbi állításokat könnyen meggondolhatjuk:

- Mélységi bejárás esetén, ha egy A csúcsból fa-éleken haladva út vezet egy B csúcsba (B gyermeke A-nak), és ebből fa-éleken út vezet egy C csúcsba, akkor biztosan nincsen más olyan fa-élekből álló út, ami A-ból C-be vezet.

- Mivel a mélységi bejárás során a fa éleket kiválasztva valóban fa gráfot kapunk, ez körmentes.

- (Mélységi bejárásnál) Értsük egy A csúcs leszármazottai alatt az összes olyan csúcsot, amit egy adott bejárásnál az A csúcs "után" fedeztünk fel és vezet beléjük út A-ból! Ekkor, ha A egy leszármazottjából vezet út egy A-nak NEM leszármazottjába, akkor azt a csúcsot már vizsgáltuk.

- Egy mélységi bejárás kezdőcsúcsa elvágó pont a gráfban, ha legalább 2 fia van.

Ezeket vizsgálva látható, hogy egy A csúcsot (nem a mélységi bejárás kezdőcsúcsa) elhagyva akkor esik részekre a gráf (A elvágó pont), ha A-nak van olyan leszármazott családja, ahonnan nem vezet út A ősébe.

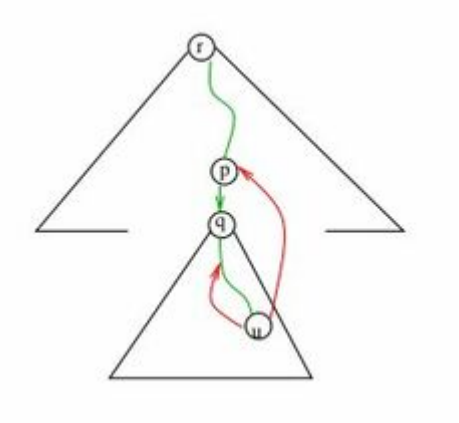
Figyeljük még meg, hogy egy A csúcs attól még lehet elvágó pont, hogy az egyik fiából vezet út A ősébe, ha egy másik fiából nincs ilyen, akkor azt a fiút és leszármazottait elvágja a gráf többi részétől!

# Formálisan

Legyen MFF = (V,F) a G = (V,E) irányítatlan (összefüggő) gráf egy mélységi feszítőfája.

MFF r gyökere akkor és csak akkor elvágó pont, ha legalább két fia van (MFF-ben).

Legyen MFF gyökerétől különböző pontja G-nek. Ekkor p akkor és csak akkor elvágó pont, ha van olyan q fia MFF-ben, hogy a q gyökerű MFF-ban lévo minden u pontra és minden u → w vissza-élre w leszármazottja p-nek (MFF-ben).



Elvágó pont

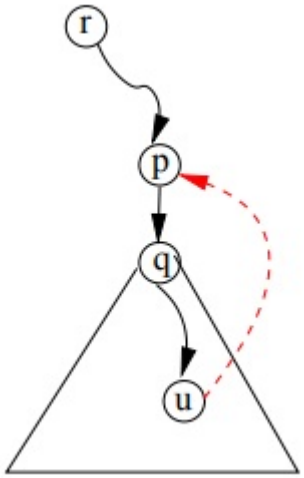
# Elvágó pontok meghatározása

Ahhoz, hogy az elvágó pontokat megtaláljuk, a legegyszerűbb módszer, ha bevezetünk a gráf csúcsaira valamilyen L(x) függvényt, ennek segítségével fogjuk figyelni a vissza-éleket.

Az algoritmus leírásához szükséges ismerni a mélységi bejárásban használt elérési idő és a szülő fogalmat, amelyeket 2 tömbben fogunk tárolni, mindkettő hossza a gráf csúcsainak száma lesz, ezek a d és a p lesznek. Ekkor a mélységi bejárás végén d[i] az i. csúcs elérési ideje, p[i] pedig az i. csúcs szülőjének indexe. (A MFF gyökerénél ez legyen 0 érték!). Az elérési idő minden új csúcs vizsgálatakor az addigi legnagyobb elérési időnél eggyel nagyobb érték lesz.

A korábban meggondolt állítások segítségével, egy X csúcs minden leszármazottjának a feszítőfában nagyobb elérési ideje lesz, mint az X csúcsnak. Ez azt is jelenti, hogy egy X csúcs akkor lesz elvágó pont, ha egy gyermekének leszármazottai között NINCS olyan csúcs, amiből (vissza-)él vezetne egy csúcsba, melynek elérési ideje kisebb mint az X elérési ideje (azaz X-nek őse). Ahhoz, hogy ezt meghatározhassuk egy adott X csúcsra, minden közvetlen fiára tudnunk kell a leszármazottakból kivezető élek közül azt, amelyik a legkisebb elérési idejű csúcsba vezet (elég a legkisebbet tárolni).

Az említett L(X) függvény a fenti minimumot fogja adni, amit ez alapján így számolhatunk: L(x) = min {d[x], min{ d[q] : ha x -> q visszaél }, min { L(q) : ha x -> q fa-él} }



A közvetlen gyermekeket kell vizsgálni

# Algoritmus

Mindezt a mélységi bejárás módosításával O(N + M) időben tudjuk vizsgálni, az alábbi algoritmus szerint. Az elvágó pont eldöntéséhez:

- gyökér esetén Elvágó[x] = x gyermekeinek száma > 1,

- nem a gyökeret nézzük: akkor ha L(x) >= d[ p[x] ] igaz, akkor Elvágó[ p[x] ] igaz.

```
list <int> G[maxN];
int p [maxN], d[maxN], L[maxN];
int n, ido = 0;

// Global : G, p, d, L , ido
void MelyBejar(int i) {
    d[i] = ++ido;
    L[i] = ido;
    for(int q : G[i]){
        if (d[q] == 0) { // p->q fa él
            p[q]=i;
            MelyBejar(q);
            if (L[q] < L[i]){
                L[i] = L[q];
            }
        } else if (q != p[i] && d[q] < L[i] ) { // p->q visszaél
            L[i] = d[q];
        }
    }
}
```

A mélységi bejárást az 1 indexű csúcsból indítva, megkapjuk a megfelelő L(x) értékeket, majd vizsgálhatjuk, őket a csúcsokon végighaladással.

```
MelyBejar(1) ;
int rfiai = 0;
for(int q = 2; q<=n ; q++ ) {
    int par = p[q] ;
    if( par == 1 ){
        rfiai++;
    }else{
        ElvagoP[par] = ElvagoP[par] || L[q] >= d[par];
    }
}
ElvagoP[1] = rfiai > 1;
```

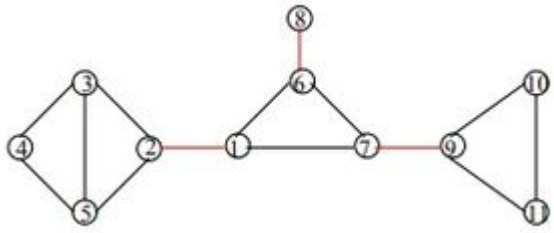
# Komplexitás

Az algoritmus (szomszédsági listás tárolás esetén), úgyanúgy mint a mélységi bejárás O(N + M)-es komplexitású.

# Hídélek

Összefüggő gráfban keresünk olyan éleket melyek elhagyásával a gráf több komponensre esik szét.

Egy (p, q) él akkor és csak akkor híd, ha nincs benne egyetlen körben sem. Ebből következik, hogy a mélységi feszítőfának mindenképpen fa-éle kell, hogy legyen.



Pirossal jelölve a hídélek

Ami ezen túl szükséges, azt könnyű meggondolni az elvágó pontok elve alapján.

Ekkor ha a MFF-ban p csúcs megelőzi q csúcsot, szükséges, hogy L(q) = d[q], azaz a q leszármazottaiból ne mutasson él q ősébe. A fenti algoritmuson keveset kell csak módosítani a hídél megtalálásához:

```
bool Hid[maxN];
void MelyBejarHid(int i){
    d[i] = ++ido;
    L[i] = ido;
```

```
    for( int q : G[i]){
        if (d[q] == 0) { // p->q f a él
            p[q]=i;
            MelyBejar (q);
            if (L[q] < L[i]){
                L[i] = L[q];
            }
            Hid[q] = L[q] == d[q];
        } else if (q != p[i] && d[q] < L[i] ) { // p->q viss zaél
            L[i] = d[q];
        }
    }
}
```

A változtatással az i -> q élet el tudjuk tárolni, hiszen p[q] = i lesz. Az algoritmus itt is O(N + M)