Gráfalgoritmusok: összefüggőség, párosítás páros gráfban

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.*

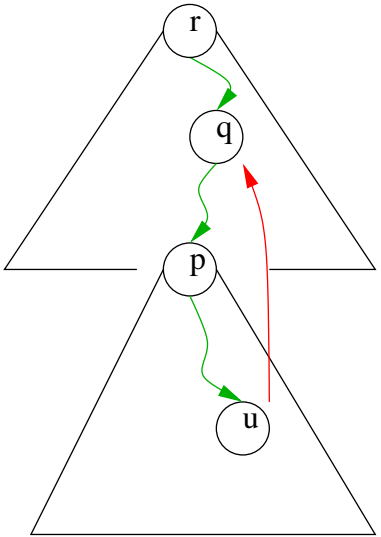
1.1. Elvágópontok

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:

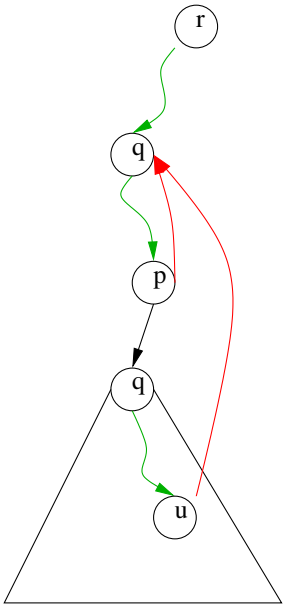


1. ábra.

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

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

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



2. ábra.

```
1  void MelyBejar(int p){
2  //Global: G, ido, D,Szin,Apa,L
3     D[p]=ido++;
4     L[p]=ido;
5     Szin[p]=Szurke;
6     for (int q:G[p])
7        if (Szin[q]==Feher){//p->q faél
8           Apa[q]=p;
9           MelyBejar(q);
10          L[p]=min(L[p],L[q]);
11       }else if(Apa[p]!=q && D[q]<L[p])//visszaél
12          L[p]=D[q];
13 }
```

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.*

```
1  void Elvágopntok(int p){
2  //Global: G, ido, D,Szin,L, ElvagoP
3     D[p]=ido++;
4     L[p]=ido;
5     Szin[p]=Szurke;
6     int fiuk=0;
7     for (int q:G[p])
8        if (Szin[q]==Feher){//p->q faél
9           fiuk++;
10          Apa[q]=p;
11          MelyBejar(q);
12          L[p]=min(L[p],L[q]);
13          if (D[p]>1 && L[q]>=D[p])
14             ElvagoP[p]=true;
15       }else if(Apa[p]!=q && D[q]<L[p])//visszaél
16          L[p]=D[q];
17
18    if(D[p]==1 && fiuk>1)
19       ElvagoP[p]=true;
20 }
```

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 függvénnyel teljesül, hogy $L(q) = D(q)$.*

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

```cpp
//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);
    }
}
```

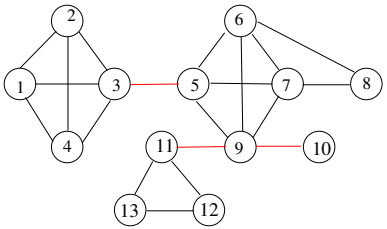
```
23 enum Paletta {Feher, Szurke, Fekete};
24 Paletta Szin[maxN];
25 int D[maxN];
26 int L[maxN];
27 int Apa[maxN];
28 vector<El> Hidak;
29 int ido;
30
31 void MelyBejar(int p){
32 //Global: G, ido, D,Szin,L, Hidak
33    D[p]=++ido;
34    L[p]=ido;
35    Szin[p]=Szurke;
36    for (int q:G[p])
37       if (Szin[q]==Feher){//p->q faél
38          Apa[q]=p;
39          MelyBejar(q);
40          L[p]=min(L[p],L[q]);
41          if (L[q]==D[q])
42             Hidak.push_back(El(p,q));
43       }else if(Apa[p]!=q && D[q]<L[p])//visszaél
44          L[p]=D[q];
45 }
```

```
46 int main (){
47    BeOlvas();
48    ido=0;
49    for(int p=1;p<=n;p++) Szin[p]=Feher;
50    for(int p=1;p<=n;p++)
51       if(Szin[p]==Feher)
52          MelyBejar(p);
53
54    printf("%d\n",Hidak.size());
55    for (El e: Hidak)
56          printf("%d-%d\n",e.p,e.q);
57 }
```

2. Kétszeresen összefüggő komponensek

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

2.2. Állítás. *Egy* $G = (V, E)$ *irányítatlan gráf akkor és csak akkor kétszeresen összefüggő, ha* $|verV| > 1$ *vagy bármely éle benne van egy körben.*



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
```

Belátható, hogy a kétszeresen összefüggő komponenseket úgy kapjuk, hogy töröljük a gráf élei közül a hidakat.

```
1  //Irányítatlan gráf kétszeresen összefüggő (biconnected) komponensek
2  #include <bits/stdc++.h>
3  #define maxN 10001
4  using namespace std;
5  struct El{
6     int p,q;
7     El(){};
8     El(int x, int y){p=x; q=y;}
9  };
10 int n;//pontok száma
11 vector<int> G[maxN];
12 void BeOlvas();
13
14 enum Paletta {Feher, Szurke, Fekete};
15 Paletta Szin[maxN];
16 int D[maxN];
17 int L[maxN];
18 int Apa[maxN];
19 stack<El> V;
20 vector<vector<El>> H;
21 int ido;
```

```
22 void MelyBejar(int p){
23 //Global: G, ido, D,Szin,L, V,H
24     D[p]=++ido;
25     L[p]=ido;
26     Szin[p]=Szurke;
27    for (int q:G[p]){
28       if (Szin[q]==Feher){//p->q faél
29          V.push(El(p,q));
30          Apa[q]=p;
31          MelyBejar(q);
32          L[p]=min(L[p],L[q]);
33          if (L[q]>=D[p]){//találtunk egy komponenst
34             if (V.top().p==p && V.top().q==q){//p-q híd
35                V.pop();
36             }else{
37                El pq;
38                vector<El> Hp;
39                do{
40                   pq=V.top(); V.pop();
41                   Hp.push_back(pq);
42                }while(pq.p!=p || pq.q!=q);
43                H.push_back(Hp);
44             }
45          }
```

```
46          }else if (q!=Apa[p] && Szin[q]==Szurke){ //p->q visszaél
47              L[p]=min(L[p],D[q]);
48              V.push(El(p,q));
49          }
50      }
51      Szin[p]=Fekete;
52  }
```

```
53 void Komponensek(){
54 //global: G, H
55    for (int p=1; p<=n; p++) Szin[p]=Feher;
56    ido=0;
57    for (int p=1; p<=n; p++)
58       if (Szin[p]==Feher) MelyBejar(p);
59    vector<El> Hp;
60    //a veremben maradt komponens kiírása
61    while(V.size()>0){
62       El pq=V.top(); V.pop();
63       Hp.push_back(pq);
64    }
65    if (Hp.size()>0) H.push_back(Hp);
66 }
```

```
67 int main (){
68    ios_base::sync_with_stdio(false);
69    cin.tie(NULL);
70    BeOlvas();
71    Komponensek();
72    cout<<H.size()<<endl;
73    for (vector<El> komp: H){
74       for(El e : komp)
75          cout<<e.p<<"-"<<e.q;
76       cout<<endl;
77    }
78 }
79 void BeOlvas(){
80 //Global:n,G
81    int m,p,q;
82    cin>>n>>m;
83    for (int i=0; i<m; i++){
84       cin>>p>>q;
85       G[p].push_back(q);
86       G[q].push_back(p);
87    }
88 }
```

3. Erősen összefüggő komponensek

3.1. Feladat: Iskolahálózat (IOI96)

Néhány iskola számítógépes hálózatba van kötve. Az iskolák a szabadon terjeszthető programok elosztására megállapodást kötöttek: minden iskola egy listát vezet azokról az iskolákról ("címzett iskolákról"), amelyek számára a programokat továbbküldi. Megjegyzés: ha B iskola szerepel az A iskola terjesztési listáján, akkor A nem feltétlenül szerepel B terjesztési listáján.

Írjon programot, amely meghatározza azt a legkisebb számot, ahány iskolához egy új programot el kell juttatni ahhoz, hogy - a megállapodás alapján, a hálózaton keresztül terjesztve - végül minden iskolába eljusson.

Bemenet

A bementi állomány első sora egy egész számot, a hálózatba kapcsolt iskolák n; számát tartalmazza $(2 \leq n \leq 1000)$. Az iskolákat az elsoő n pozitív egész számmal azonosítjuk. A következő n sor mindegyike egy-egy terjesztési listát ír le. Az $i+1$-edik sor az i-edik iskola terjesztési listáján szereplő iskolák azonosítóit tartalmazza. Minden lista végén egy 0 szám áll. Az üres lista csak egy 0 számból áll.

Kimenet

A kimenti állomány első sorába azt a legkisebb m számot kell írni, ahány iskolához egy új programot el kell juttatni ahhoz, hogy - a megállapodás alapján, a hálózaton keresztül terjesztve - végül minden iskolába eljusson. A második sor pontosan m egész számot tartalmazzon egy-egy szóközzel elválasztva, azon iskolák sorszámait, ahova kezdetben el kell juttatni az új programot.
Több megoldás esetén bármelyik megadható.

Példa bemenet és kimenet

bemenet

```
13
2 0
3 0
1 4 9 0
1 0
10 6 0
7 0
8 5 0
13 0
10 0
11 12 0
9 0
13 0
12 0
```

kimenet

```
2
1 5
```

Elvi megoldás

Legyen $G = (V,E)$ az iskolahálózatot leíró irányított gráf, tehát $V = \{1,\ldots,n\}$ és $(p,q) \in E$ akkor és csak akkor, ha a p iskola terjesztési listájában szerepel q.
A megoldás egy olyan $D \subseteq V$, amelyre teljesül:

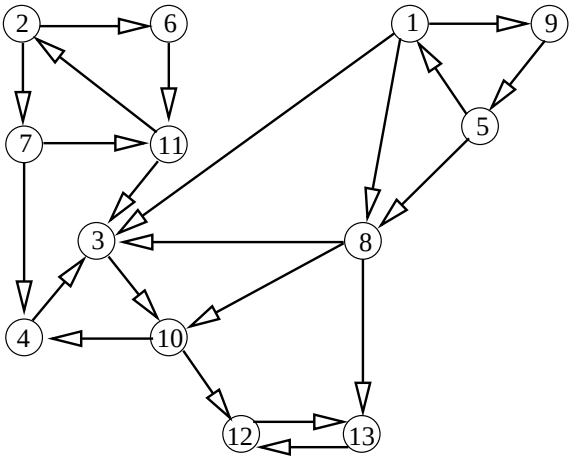
$$(\forall q \in V)(\exists p \in D)(p \rightsquigarrow q)$$

A legkisebb elemszámú ilyen D halmazt keressük.
A megoldást lépésenként építhetjük:
Adott p pontra jelölje $Eler(p)$ a p pontból elérhető pontok halmazát:

$$Eler(p) = \{q \in V : p \rightsquigarrow q\}$$

$D := \emptyset$;
$DbolElert := \emptyset$;
$p \in V$
  $p \notin DbolElert$ {
    $D := D - Eler(p) \bigcup \{p\}$
    $DbolElert := DbolElert \bigcup Eler(p)$
  }

Az algoritmus futási ideje legrosszabb esetben $O(n^3)$. Megmutatjuk, hogy lényegesen gyorsabban is lehet számítani.

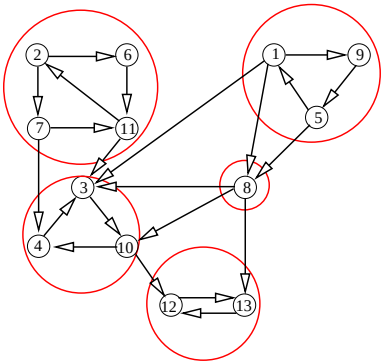


4. ábra. A hálózat gráfja

Soroljuk egy osztályba azokat a pontokat, amelyek egymásból elérhetők, jelölje ezt az eqvivalencia relációt $\equiv$. Tehát $p \equiv q$ akkor és csak akkor, ha $p \rightsquigarrow q$ és $q \rightsquigarrow p$. A $p \equiv q$ reláció nyilvánvalóan eqvivalencia-reláció. Jelölje $[p]$ a p pontot tartalmazó eqvivalencia-osztályt, tehát

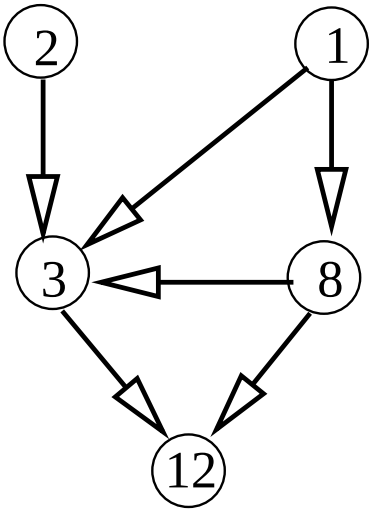
$$[p] = \{q : p \rightsquigarrow q \text{ és } q \rightsquigarrow p\}$$

A $[p]$ halmazokat a gráf erősen összefüggő komponenseinek nevezzük.
Tekintsük azt a gráfot, amelynek pontjai a G gráf erősen összefüggő komponensei, és $[p] \to [q]$ akkor és csak akkor él, ha van olyan $u \in [p]$ és van olyan $v \in [q]$ hogy $u \to v$ él az eredeti G gráfban. Ezt a gráfot G komponens-gráfjának nevezzük.
Nyilvánvaló, hogy a komponens-gráf körmentes. A megoldás kiolvasható a komponens-gráfból:



5. ábra. Erősen összefüggő komponensek

minden olyan komponensből kell egy-egy pontot választani, amely komponensbe nem vezet él a



6. ábra. Komponens-gráf

komponens-gráfban.
A feladat megoldásához nem kell feltétlenül előállítani a komponens-gráfot.
Tegyük fel, hogy a G gráfnak k komponense van (ha k=1, akkor a gráfot erősen össefüggőnek mondjuk). Végezzünk egy mélységi bejárást, és legyenek $p_1, \ldots, p_k$ az egyes komponensek azon elemei, amelyeket elsőnek érjük el a mélységi bejárás során. Ezeket a pontokat a komponens ősapjának nevezzük.
Rakjuk sorba a gráf pontjait úgy, hogy amikor a mélységi bejárás során a pontot feketére színezzük, akkor a pontot a sor elejére rakjuk. Ebben a sorrendben az ősapák olyan sorrendben lesznek, hogy ha van $p_i \to p_j$ út, akkor p_i előbb szerepel a sorban, mint p_j. Tehát a sorban az első pont olyan ősapa, amely komponense 0-befokú. Vegyük be ezt a megoldáshalmazbe, majd fessük be a belőle elérhető pontokat. Így az első be nem festett pont ismet egy olyan ősapa, amelynek komponense 0-befokú. Folytatva a beválasztás, befestés eljárást, amíg minden pontot be nem festettünk, megkapjuk a megoldást.
Ez az algoritmus az élek számával arányos futási idejű lesz.

```
vector<int> G[maxN];

enum Paletta {Feher, Szurke, Fekete};
int T[maxN];
Paletta Szin[maxN];
int n,nn;

void MelyBejar1(int p){
   Szin[p]=Szurke;
   for (int q:G[p])
      if (Szin[q]==Feher)
         MelyBejar1(q);
   T[nn--]=p;
}
void MelyBejar2(int p){
   Szin[p]=Szurke;
   for (int q:G[p])
      if (Szin[q]==Feher)
         MelyBejar2(q);
}
void Beolvas(){}
```

```
22 int main(){
23    Beolvas();
24    for(int i=1;i<=n;i++) Szin[i]=Feher;
25    nn=n;
26    for(int x=1;x<=n;x++)
27       if(Szin[x]==Feher)
28          MelyBejar1(x);
29    for(int i=1;i<=n;i++) Szin[i]=Feher;
30    vector<int> mego;
31    for(int i=1;i<=n;i++){
32       int x=T[i];
33       if(Szin[x]==Feher){
34          mego.push_back(x);
35          MelyBejar2(x);
36       }
37    }
38    cout<<mego.size()<<endl;
39    for(int x:mego)
40       cout<<x<<"␣";
41    cout<<endl;
42   return 0;
43 }
```

Ha elő kell állítani az erősen összefüggő pomponenseket, akkor az előző algoritmust annyiban kell módosítani, hogy a festés helyett a transzponált gráfban az eléhető pontok kerülnek egy-egy komponensbe.

```
1  //Irányított gráf erősen összefüggő komponensei
2  #include <bits/stdc++.h>
3  #define maxN 10001
4  using namespace std;
5  struct El{
6     int p,q;
7     El(){};
8     El(int x, int y){p=x; q=y;}
9  };
10 int n;//pontok száma
11 int ido;
12 vector<int> G[maxN];
13 vector<int> Gt[maxN];//a transzponált gráf
```

```
14 void BeOlvas(){
15 //Global:n,G
16    int m,p,q;
17    cin>>n>>m;
18    for (int i=0; i<m; i++){
19       cin>>p>>q;
20       G[p].push_back(q);
21       Gt[q].push_back(p);
22    }
23 }
24 enum Paletta {Feher, Szurke, Fekete};
25 Paletta Szin[maxN];
26 int T[maxN];
27 vector<vector<int>> H;
```

```
28 void MelyBejar(int p){
29 //Global: G, ido, D,Szin
30    Szin[p]=Szurke;
31    for (int q:G[p])
32       if (Szin[q]==Feher)
33          MelyBejar(q);
34    T[ido++]=p;
35 }
36 vector<int> Hp;
37 void MelyBejarT(int p){
38 //Global: Gt, Szin, Hp
39    Hp.push_back(p);
40    Szin[p]=Fekete;
41    for (int q:Gt[p])
42       if (Szin[q]==Szurke)
43          MelyBejarT(q);
44 }
```

```
45 int main (){
46    BeOlvas();
47    for (int p=1; p<=n; p++) Szin[p]=Feher;
48    ido=1;
49    for (int p=1; p<=n; p++)
50       if (Szin[p]==Feher) MelyBejar(p);
51    for(int i=n;i>0;i--){
52       if(Szin[T[i]]==Szurke){
53          Hp.clear();
54          MelyBejarT(T[i]);
55          H.push_back(Hp);
56       }
57    }
58    cout<<H.size()<<endl;
59    for(vector<int> h: H){
60       for(int x:h)
61          cout<<x<< "␣";
62       cout<<endl;
63    }
64    return 0;
65 }
```

4. Párosítás páros gráfban

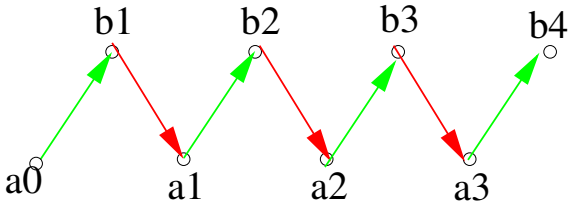
4.1. definíció. *A $G = (V,E)$ irányítatlan gráfot páros gráfnak nevezzük, ha a V ponthalmaz felbontható két diszjunkt A és B részhalmazra úgy, hogy bármely $(u,v) \in E$ él esetén vagy $u \in A$ és $v \in B$, vagy $u \in B$ és $v \in A$.*

4.2. definíció. *A $G = (V,E)$ irányítatlan páros gráf esetén egy $P \subseteq E$ élhalmazt párosításnak nevezünk, ha bármely két P-beli élnek nincs közös pontja.*

4.1. Maximális párosítás: magyar módszer

Legyen P egy párosítása a $G=(V,E)$ páros gráfnak, ahol $A+B=V$. Azt mondjuk, hogy egy $p \in V$ pont párosított, ha valamely q pontra $(p,q) \in P$. Az $a_0, b_1, a_1, \ldots, b_k, a_k, b_{k+1}$ pontsorozatot növelő útnak nevezzük, ha az alábbiak teljesülnek.

1. a_0 és b_{k+1} nem párosított és $a_0 \in A$ és $b_{k+1} \in B$ vagy $a_0 \in B$ és $b_{k+1} \in A$
2. $(a_i, b_i) \in P$; $i = 1, \ldots, k$
3. $(a_i, b_{i+1}) \in E$; $i = 0, \ldots, k$



7. ábra. Növelő út

Nyilvánvaló, hogy ha a fenti pontsorozat növelőút, akkor a $P - \{(a_i, b_i) : i = 1, \ldots, k\} \bigcup \{(a_i, b_{i+1}) : i = 1, \ldots, k+1\}$ is párosítás és eggyel több párt tartalmaz, mint P. A megyar módszer (Kőnig Dénes és Egervári Károly munkája alapján) olyan algoritmus maximális párosítás előállításáa, amely növelő utak keresésén alapszik. Az algoritmus helyességét az alábbi állítás bizonyítja.

4.3. Állítás. *Legyen P egy olyan párosítása a $G=(V,E)$ páros gráfnak, hogy nincs növelő út a gráfban. Ekkor P maximális párosítás.*

```cpp
#include <bits/stdc++.h>
#define maxN 10001
using namespace std;

vector<int> G[maxN];
bool Nemvolt[maxN];
int Par[maxN];
int n;
```

```
 9  bool NoveloUtKeres(int a){
10  //Globalis: G,Par,Nemvolt
11     Nemvolt[a]=false;
12     for (int b:G[a]){
13        if(Nemvolt[b]){
14           if(Par[b]==b){
15              Par[a]=b; Par[b]=a;
16              return true;
17           }else{
18              Nemvolt[b]=false;
19              if(NoveloUtKeres(Par[b])){
20                 Par[a]=b; Par[b]=a;
21                 return true;
22              }
23           }
24        }
25     }
26     return false;
27  }
```

```
28 int MagyarModszer(){
29 //A+B prarició meghatározása
30    bool A[maxN];
31    queue<int> S;
32    for(int x=1;x<=n;x++){
33       A[x]=false;
34       Nemvolt[x]=true;
35    }
36    A[1]=true; Nemvolt[1]=false;
37    S.push(1);
38    while(!S.empty()){
39       int x=S.front(); S.pop();
40       for(int y:G[x])
41       if(Nemvolt[y]){
42          Nemvolt[y]=false;
43          A[y]=!A[x];
44       }
45    }
46    int parokszama=0;
47    for(int i=1;i<=n;i++) Par[i]=i;
```

```
48  //Növelo utak keresése
49     for(int a=1;a<=n;a++)
50     if(A[a] && Par[a]==a){
51        for(int x=1;x<=n;x++) Nemvolt[x]=true;
52        if(NoveloUtKeres(a))
53           parokszama++;
54     }
55     return parokszama;
56  }
```

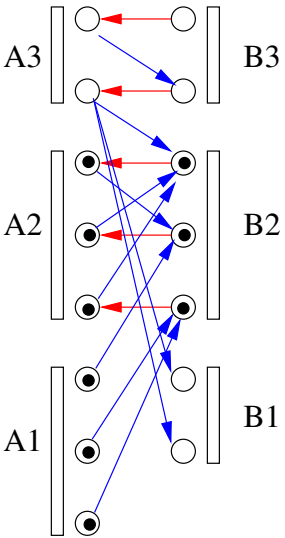
```
57 void Beolvas(){
58   int m,x,y;
59   cin>>n>>m;
60   for (int i=1;i<=m;i++){
61     cin>>x>>y;
62     G[x].push_back(y);
63     G[y].push_back(x);
64   }
65 }
66 int main(){
67    Beolvas();
68    int m=MagyarModszer();
69    cout<<m<<endl;
70    for(int a=1;a<=n;a++)
71       if(Par[a]!=a){
72          cout<<a<<" "<<Par[a]<<endl;
73          Par[Par[a]]=Par[a];
74       }
75   return 0;
76 }
```

4.2. Minimális lefedő halmazok

Tekintsünk egy $G = (V, E)$ páros gráfot, ahol $V = A + B$ és legyen P egy maximális párosítása G-nek.

4.4. definíció. *Egy $p_1, \ldots, p_k$ utat alternáló útnak vezünk, ha az egymást követő élek közül az egyik P-beli, a másik nem P-beli.*

A gráf éleit irányítsuk úgy, hogy $u \to v$ ha $u \in A$ és (u, v) nem párosított, vagy $u \in B$ és (u, v) párosított. Jelölje $A(x)$ azt a logikai függvényt, amleyre $A(x)$ akkor és csak akkor igaz, ha $x \in A$.
Ha az x pont párosított, akkor $Par(x)$ jelölje x párját, egyébként legyen $Par(x) = x$
Definiáljuk az L logikai függvényt az V ponthalmazon úgy, hogy $L(x)$ akkor és csak akkor igaz, ha van olyan $a \in A$ pont, hogy x elérhető a-ból alternáló úton. Ha $L(x)$ igaz, akkor azt mondjuk, hohy x L-cimkézett.
Soroljuk a gráf pontjait az $A1, A2, A3, B1, B2, B3$ diszjunk részhalmazokba az alábbiak szerint.
$A1(x) = A(x) \wedge Par(x) = x$: nem párosított A-beli pontok halmaza
$A2(x) = A(x) \wedge Par(x) \neq x \wedge L(x)$: párosított A-beli L-címkézett pontok halmaza
$A3(x) = A(x) \wedge Par(x) \neq x \wedge !L(x)$: párosított A-beli nem L-címkézett pontok halmaza
$B1(x) = !A(x) \wedge Par(x) = x$: nem párosított B-beli pontok halmaza
$B2(x) = !A(x) \wedge Par(x) \neq x \wedge L(x)$: párosított B-beli L-címkézett pontok halmaza
$B3(x) = !A(x) \wedge Par(x) \neq x \wedge !L(x)$: párosított B-beli nem L-címkézett pontok halmaza



8. ábra. Pontok osztályozása (maximális) párosítás alapján.

4.3. Minimális lefedő ponthalmaz

4.5. definíció. *Egy $G = (V,E)$ irányítatlan gráf $F \subseteq V$ ponthalmazát lefedő ponthalmaznak nevezzük, ha bármely $(u,v) \in E$ élre $u \in F$ vagy $v \in F$.*

4.6. Állítás. *$A3 \bigcup B2$ halmaz minimális feledő ponthalmaz.*

4.7. Állítás. *A minimális feledő ponthalmaz elemszáma megegyezik a maximális párosítás elemszámával.*

4.4. Maximális független ponthalmaz

4.8. definíció. *Egy $G = (V,E)$ irányítatlan gráf $F \subseteq V$ ponthalmazát független ponthalmaznak nevezzük, ha bármely $(u,v) \in E$ élre $u \notin F$ vagy $v \notin F$.*

4.9. Állítás. *$A1 \bigcup A2 \bigcup B1 \bigcup B3$ maximális független ponthalmaz.*

4.5. Minimális lefedő élhalmaz

4.10. definíció. *Egy $G = (V,E)$ irányítatlan gráf $F \subseteq E$ élhalmazát lefedő élhalmaznak nevezzük, ha bármely $u \in V$ pontra van olyan $v \in V$ hogy $(u,v) \in F$.*

4.11. Állítás. *Párosító élek + egy-egy $A1 \to B2$ és $B1 \to A3$ él minimális lefedő élhalmazt alkot.*

```cpp
#include <bits/stdc++.h>
#define maxN 10001
using namespace std;

vector<int> G[maxN];
bool Nemvolt[maxN];
int Par[maxN];
bool A[maxN];
bool L[maxN];
bool C[maxN];
int n;
```

```
12 bool NoveloUtKeres(int a){
13 //Globalis: G,Par,Nemvolt
14    Nemvolt[a]=false;
15    for (int b:G[a]){
16       if(Nemvolt[b]){
17          if(Par[b]==b){
18             Par[a]=b; Par[b]=a;
19             return true;
20          }else{
21             Nemvolt[b]=false;
22             if(NoveloUtKeres(Par[b])){
23                Par[a]=b; Par[b]=a;
24                return true;
25             }
26          }
27       }
28    }
29    return false;
30 }
```

```
31 int MagyarModszer(){
32 //A+B prarició meghatározása
33    queue<int> S;
34    for(int x=1;x<=n;x++){
35       A[x]=false;
36       Nemvolt[x]=true;
37    }
38    A[1]=true; Nemvolt[1]=false;
39    S.push(1);
40    while(!S.empty()){
41       int x=S.front(); S.pop();
42       for(int y:G[x])
43       if(Nemvolt[y]){
44          Nemvolt[y]=false;
45          A[y]=!A[x];
46       }
47    }
48    int parokszama=0;
49    for(int i=1;i<=n;i++) Par[i]=i;
```

```
50 //Növelo utak keresése
51    for(int a=1;a<=n;a++)
52    if(A[a] && Par[a]==a){
53       for(int x=1;x<=n;x++) Nemvolt[x]=true;
54       if(NoveloUtKeres(a))
55          parokszama++;
56    }
57    return parokszama;
58 }
59
60 void AltBejar(int a){
61    L[a]=true;
62    Nemvolt[a]=false;
63    for (int b:G[a]) if (Nemvolt[b] && Par[b]!=b){
64       L[b]=true; Nemvolt[b]=false;
65       AltBejar(Par[b]);
66    }
67 }
```

```
68 void Beolvas(){
69   int m,x,y;
70   cin>>n>>m;
71   for (int i=1;i<=m;i++){
72     cin>>x>>y;
73     G[x].push_back(y);
74     G[y].push_back(x);
75   }
76 }
```

```
77 int main(){
78    Beolvas();
79    int m=MagyarModszer();
80    cout<<m<<endl;
81 //minimális lefedő élhalmaz
82    for(int a=1;a<=n;a++)
83       if(Par[a]!=a){
84          cout<<a<<" "<<Par[a]<<endl;
85          Par[Par[a]]=Par[a];
86       }
87 //minimális lefedő ponthalmaz
88    for(int x=1;x<=n;x++){
89       Nemvolt[x]=true;
90       L[x]=!A[x];
91       C[x]=false;
92    }
93    for(int x=1;x<=n;x++)
94       if(!A[x]) AltBejar(x);
95    for(int x=1;x<=n;x++)
96       C[x]= A[x] && !L[x] || !A[x] && L[x];
97
98  return 0;
99 }
```