A cikksorozat előző részeiben megismertünk néhány hasznos gráfalgoritmust, és láttuk, hogyan alkalmazhatók a Nemes Tihamér verseny vagy az informatika OKTV feladatainak megoldásánál. Próbáljunk most az eddigiek ismeretében megoldani egy KöMaL programozási feladatot.

S. 86. Egy parkban azt látjuk, hogy N majom közül az 1-es számú egy nagyon nagy fa egyik ágába kapaszkodik a farkával. A majmok a kezükkel kapcsolódhatnak más majmokhoz úgy, hogy egy majom egy másik lábát fogja. Így tarthatnak más majmokat azáltal, hogy beléjük kapaszkodnak, vagy úgy, hogy beléjük kapaszkodnak más majmok. Egy majom lábába bárhány majom kapaszkodhat. Szeretnénk végezni egy M másodpercig tartó vizsgálatot, amelynél minden másodpercben egy majom szabaddá teszi adott kezét, vagyis elengedi egy másik majom lábát, ha eddig fogta azt. A program számítsa ki, hogy mikor esnek le az egyes majmok a földre. Egy majom akkor esik le, ha az 1-es számú majom nem tartja közvetett, vagy közvetlen módon.

A standard bemenet első sorában a majmok $1 \leq N \leq 200\,000$ száma és a vizsgálat ideje $1 \leq M \leq 400\,000$ másodperc található, a következő N sor mindegyikében két szám (b_i és j_i), amely megadja, hogy az i-edik majom a bal és a jobb kezével hányadik majomba kapaszkodik. Amennyiben valamelyik szám -1, az azt jelenti, hogy azzal a kezével senkibe sem kapaszkodik, azzal a kezével senkit sem tart. A standard bemenet következő M sorában az x_j, y_j egészek megmutatják, hogy a j-edik másodpercben az x_j-edik majom az y_j-edik kezével elengedi a fogást (1. kéz a bal, 2. kéz a jobb). A program írja a standard output N sorába a majmok földet érésének idejét. Az i-edik sorba az i-edik majom földet érésének ideje kerüljön, és -1, ha nem ér földet a vizsgált időintervallum végéig.

A feladatról természetesen azonnal látható, hogy a majmok megfeleltethetők egy gráf csúcsainak, míg a kapaszkodások a csúcsok közötti irányított éleknek. Röviden összefoglalva az a feladatunk, hogy megtudjuk, a gráf valamely része mikor vált el az 1-es csúcsot tartalmazó részgráftól.

Ha nagyon egyszerűen szeretnénk megoldani a problémát, akkor azt mondhatnánk, hogy annyi feladatunk van, ahány kézelengedés történik a majmok között. Ennek megfelelően a gráfban minden egyes él elvétele után egy, az 1-es csúcstól induló bejárás megadja azokat a csúcsokat, amelyek onnan elérhetők. A bejárás során most elérhetetlenné vált csúcsok jelentik azokat a majmokat, amelyek az élnek megfelelő kéz elengedésekor leestek. Sajnos ez a megoldás nem hatékony, mivel M számú bejárásra van szükség, mert ennyi kézelengedés volt. Bár a csúcsok száma N értékéről minden kézelengedéskor csökken, de legrosszabb esetben így is előfordulhat, hogy az algoritmus lépésszáma $M \cdot N$-nel arányos, mivel az N csúcsot tartalmazó gráf bejárása N-nel arányos futásidőt igényel.

Mindenképp jobb megoldást kell keresnünk, mivel a feladatban százezres nagyságrendű csúcs és kézelengedés is előfordulhat. Gondolkozzunk egy keveset, járjuk körbe a problémát. Először is látnunk kell, hogy a kéz által történő fogás valójában szimmetrikus viszony a majmok között, tehát a leendő megoldásban alkalmazott bejárásnál az irányított éleken mindkét irányba kell haladnunk. Másrészt rájöhetünk, hogy egy-egy él megszüntetése csak akkor jelent igazi változást a gráfban, ha az több részre válik szét. Egy összefüggő részgráfon belüli él elvétele, ha a rész továbbra is összefüggő marad, nem jelent valódi változást: a majmok továbbra is együtt kapaszkodnak vagy esnek le. A gráf azon részeit, amelyek önmagukban összefüggőek ugyan, de nem kapcsolódnak a gráf más részeihez, komponenseknek hívjuk. Hatékonyabb megoldás volna azt megtudnunk, hogy milyen komponensekre bontják a gráfot a kézelengedések. Ezek után a leesés időpontja az, amikor egy olyan komponens keletkezik, amely nem tartalmazza az 1-es csúcsot.

Gyorsabb megoldást kapunk tehát, ha az egyedi csúcsok helyett komponensekben gondolkodunk. Mivel az összes komponens csak a végén, minden kézelengedés után alakul ki biztosan, ezért vizsgáljuk a problémát ettől az időponttól. Tegyük föl, hogy meghatároztuk a komponenseket, azaz mindegyik csúcsról tudjuk, hogy melyik komponens része. Nézzük most időben visszafelé a történteket: adjuk hozzá az éleket fordított sorrendben a gráfhoz. Ha olyan élt adunk hozzá, amely az adott időpontban azonos komponensben lévő csúcsok között van, akkor a komponens nem változik. Ha olyan élt hozunk létre, amely két különböző komponens között létesít kapcsolatot, akkor abban az időpontban a két komponens egyesül, innentől kezdve (időben ezelőtt) egy komponensként kell kezelni őket. Ha az egyesítéskor az egyik komponens az 1-es számú volt (amiben az 1-es csúcs van), akkor megtudtuk, hogy mikor kapcsolódott szét az adott komponens és az 1-es számú komponens. Vagyis megadtuk a komponensben lévő csúcsok által reprezentált összes majom leesés idejét. Bár a feladat leírása külön nem zárja ki, de feltesszük, hogy a kézelengedés a még fán lógó majmok valamelyikére vonatkozik. Ez azt jelenti, hogy időben visszafelé haladva két egyesülő komponens közül az egyik mindig az 1-est tartalmazó.

Kérdés tehát az, hogy lehet-e hatékonyan komponensekre bontani egy gráfot? Eddigi ismereteink alapján igen, hiszen a szélességi vagy a mélységi bejárás megadja az adott csúcsból elérhető részgráfot, vagyis a kiinduló csúcs komponensét. A feladatban induljunk ki az 1-es csúcstól, és jelöljük meg az elért csúcsokat, ha az 1-es komponenshez tartoznak. Ezután indítsunk a következő olyan csúcstól bejárást, amely nem az 1-es komponens része, legyen ennek a sorszáma 2-es, és így tovább (az aktuális komponensszám legyen `ksz`). A komponensekre bontás algoritmusa N-nel arányos lépésszámú, hiszen a teljes gráfot bejárja, és a bejárásnál minden csúcs pontosan egyszer kerül be az algoritmusban alkalmazott `Sor`-ba.

```
Eljárás Komponensekre bontás
 cs := 1 (kezdjük az első csúccsal)
 ksz := 0 (még nem találtunk egy komponenst sem)
 Ciklus amíg cs ≤ N
     ksz := ksz+1 (ez egy újabb komponens)
     Bejárás(cs) (pl. szélességi bejárás cs-től indulva)
```

```
        Ciklus amíg cs ≤ N és a cs-edik csúcsnak nincs komponense
            cs := cs+1
        Ciklus vége
    Ciklus vége
Komponensekre bontás eljárás vége
```

A `Bejárás(cs)` algoritmusa természetesen a `cs`-ből elérhető csúcsok komponensét az aktuális `ksz`-re állítja. Az algoritmus lefutása után `ksz` számú komponens jön létre, melyek közül az 1-est tartalmazó kapja az 1-es számot, a többiek pedig egy-egy nagyobb értéket. A felbontáshoz alkalmazott bejárás legyen például a szélességi bejárás. Fontos úgy elkészítenünk, hogy az irányított éleken fordított irányba is haladjunk, mivel a kapaszkodás szempontjából az él irányítottsága lényegtelen.

Mielőtt azonban elkészítjük a bejárást, be kell olvasnunk és tárolnunk kell a feladatban szereplő gráfot. Adott N mellett a gráf szomszédsági mátrixa N^2 számú elemet tartalmaz, ami N legnagyobb értéke esetén $4 \cdot 10^{10}$ logikai érték. Ha minden érték 1 bitet foglal el, akkor is 5 GB szükséges a tárolásához. Ez túl sok, főleg ahhoz képest, hogy valójában csak $4N$ élt kell tárolnunk. Válasszunk kisebb memóriafoglalást igénylő adatszerkezetet. Mivel a bejárásnál az adott csúcstól kiinduló éleket sorban feldolgozzuk, ezért logikus pl. egy listában elhelyezni őket. Minden majomnál tároljuk a bal és jobb kéz által fogott másik majom sorszámát, illetve egy listába fűzzük az őket fogó majmok sorszámát. Az N majom adatának beolvasásakor a kapcsolatok tárolása és a listák fölépítése N-nel arányos lépésszámú művelettel elvégezhető, hiszen lényegtelen a listában a kapcsolódó majmok sorrendje, tehát egyszerűen mindig a lista egyik végére fűzünk. A listákat a program futása során nem kell megváltoztatnunk, csupán azt kell jelölnünk a két kéz esetén, hogy melyik mikor engedett el egy másik majmot.

A listákról ebben a cikkben nem kívánunk külön szólni, mivel nagyon jó leírás található róluk pl. a http://tamop412.elte.hu/tanar online tananyagban, és természetesen más forrásokban is. A legtöbb magas szintű programozási nyelvhez vannak kiegészítő programkönyvtárak, amelyekben készen megtalálható a listakezelés (pl. RTL, java.util, STL). Hasonló módon megtaláljuk a programkönyvtárakban a bejárás során használt `Sor`-t.

Foglaljuk össze, hogy milyen adatokat kell tehát tárolnunk egy-egy majomról, vagyis mit tartalmaz egy csúcs: ismerjük a bal és jobb kezével tartott másik majom sorszámát, azt, hogy a két kéz mikor engedi el a másik majmot, hogy hányadik komponensben szerepel majd a csúcs, valamint azt, hogy mely más majmok fogják őt. Ezeket az adatokat a következő rekord/struktúra tagjaiként adjuk meg:

```
Rekord Majom
 kéz[] (kételemű tömb - az 1. és 2. kéz által fogott másik majom sorszáma, vagy −1,
        ha nem fog senkit)
 eng[] (szintén kételemű tömb - a két kéz mikor engedi el a másik majmot, −1, ha soha)
 komp (a csúcs komponensének száma, vagy −1, ha még ismeretlen)
 láb (az adott majmot fogó majmok sorszámának listája)
Rekord vége
```

A bemenet tárolásához tehát N ilyen rekordra van szükségünk, amelyek egy `majmok` tömbben tárolhatók, pl. a beolvasás sorrendjében. A bemenetben megtalálható további M számpár egy `enged` tömbben tárolható, amelynek elemei a számpárokat tartalmazó rekordok. A bemenet az elengedés idejének sorrendjében tartalmazza a számokat, tehát a tömb indexe jelenti az elengedés időpontját. A bemenet feldolgozása így a következőképp alakul:

```
Eljárás Adatok beolvasása
 Beolvas: N, M
 Ciklus cs := 1-től N-ig
     majmok[cs].láb := üres lista
     majmok[cs].komp := −1
     majmok[cs].eng[1] := −1
     majmok[cs].eng[2] := −1
 Ciklus vége
 Ciklus cs := 1-től N-ig
     Beolvas: k1, k2
     majmok[cs].kéz[1] := k1
     Ha k1 ≠ −1 akkor majmok[k1].láb.hozzáfűz(cs)
     majmok[cs].kéz[2] := k2
     Ha k2 ≠ −1 akkor majmok[k2].láb.hozzáfűz(cs)
 Ciklus vége
 Ciklus idő := 1-től M-ig
      Beolvas: mi, ki
      enged[idő].maj := mi
      enged[idő].kéz := ki
      majmok[mi].eng[ki] := idő
 Ciklus vége
Adatok beolvasása eljárás vége
```

Az adatok feldolgozása és a kezdeti értékek beállítása után kezdődhet a `Komponensekre bontás`, amelyhez a `Bejárás` algoritmust még nem készítettük el. Csak annyiban kell bővítenünk a szélességi bejárást, hogy az aktuális komponens értéket bejegyezzük minden csúcshoz. A komponens -1 kezdeti értéke egyben azt is jelzi, hogy még egyik bejárásnál sem jártunk az adott csúcsban. Mivel a komponenseket az összes elengedés utáni állapotban keressük, ezért figyelnünk kell, hogy csak olyan éleken haladjunk át, amelyek elengedési ideje -1 maradt, hiszen a többieknél ez az érték a beolvasásnál az elengedés időpontjára módosult, ami biztosan pozitív.

Ciklus elem := majmok[cs].láb első elemétől az utolsó eleméig
sz := elem által meghatározott majom sorszáma
Ha majmok[sz].kéz[1] = cs **és** majmok[cs].eng[1] = -1 **vagy**
majmok[sz].kéz[2] = cs **és** majmok[cs].eng[2] = -1
akkor
Ha majmok[sz].komp = -1 **akkor**
majmok[sz].komp := ksz
Sorba(sz)
Elágazás vége
Elágazás vége
Ciklus vége
Ciklus vége
Bejárás eljárás vége

```
Eljárás Bejárás(start)
    Sor legyen üres
    majom[start].komp = ksz
    Sorba(start)
    Ciklus amíg Sor nem üres
        Sorból(cs)
        Ciklus k := 1-től 2-ig (mindkét kezet megvizsgáljuk)
            Ha majmok[cs].kéz[k] ≠ −1 és majmok[cs].eng[k] = -1 akkor
                sz := majmok[cs].kez[k] (az sz sorszámú majmot fogjuk)
                Ha majmok[sz].komp = −1 akkor (sz még nincs egyik komponensben sem)
                   majmok[sz].komp := ksz
                    Sorba(sz)
                Elágazás vége
            Elágazás vége
        Ciklus vége
       Ciklus elem := majmok[cs].láb első elemétől az utolsó eleméig
            sz := elem által meghatározott majom sorszáma
            Ha majmok[sz].kéz[1] = cs és majmok[cs].eng[1] = −1 vagy
                    majmok[sz].kéz[2] = cs és majmok[cs].eng[2] = −1
            akkor
                Ha majmok[sz].komp = −1 akkor
                    majmok[sz].komp := ksz
                    Sorba(sz)
                Elágazás vége
            Elágazás vége
        Ciklus vége
    Ciklus vége
Bejárás eljárás vége
```

Mivel a listákat nem akartuk bejárni akkor, amikor a kézfogásokat elengedtük, ezért a listákból nem derül ki, hogy a kapcsolat megszakad, vagy sem. A bejárás időigényes művelet, mivel csak a lista végéről indulva lehet egyesével végiglépkedni az elemeken, amíg meg nem találjuk és ki nem töröljük a megfelelő elemet. Ezért amikor a bejárás a listából kiolvasható sz szomszédot mutatja, akkor először meg kell keresnünk, hogy az melyik kezét jelenti az sz-edik majomnak, és meg kell tudnunk, hogy tart-e az a kéz még a vizsgálat végén.

Bár a gráf éleit részben egy listában tároltuk, ez a helytakarékosság javítása mellett szerencsére nem növelte számottevően a komponensekre bontás futásidejét. Először is mind az N él csak egyszer kerül a bejárásnál alkalmazott Sor-ba, és minden él listáján csak egyszer haladunk végig. A listákban legföljebb kétszer szerepelhet az N csúcs bármelyike, ezért a végigjárt listák teljes hossza legföljebb $2 \cdot N$. Így a komponensekre bontás összességében N-nel arányos lépésszámú művelet.

Ennyi komoly előkészítő munka után nézzük a végeredmény előállításának módját. Vegyünk föl egy komple nevű táblázatot, amelynek megfelelő sorszámú helyére beleírjuk az adott komponens leesésének idejét. Legyen ebben a táblázatban az 1-es komponensnél -1, és kezdetben legyen 0 a többi helyen, a még ismeretlen

leesési idejű komponenseknél.

Vizsgáljuk időben visszafelé a majmok szétkapcsolódását, ami most a gráf csúcsai között élek létrehozását jelenti. Ha egy kapcsolat (kézfogás) abban az időpontban egy komponensen belül jön létre, akkor azzal a komponenst nézve semmi nem változik. Igazi változás akor történik, ha két olyan csúcsot kötünk épp össze, amelyek az adott időpontban különböző komponensben vannak. Ez most csak úgy lehetséges, ha a két komponens leesési ideje közül az egyik zérus (még ismeretlen), a másik pedig nem zérus (-1 vagy pozitív). Ekkor a zérus idő helyére beírjuk az aktuális időpontot, vagyis az adott él sorszámát.

Eljárás S86
komple[] = 0
komple[1] = -1
    Ciklus idő := M-től 1-ig -1-esével
        m1 := enged[idő].maj (a kezével elengedő majom sorszáma)
        k1 := majmok[m1].komp (a kezével elengedő majom komponense)
        k := enged[idő].kéz (az elengedő kéz sorszáma)
        m2 := majmok[m1].kéz[k] (az elengedett majom sorszáma)
        k2 := majmok[m2].komp (az elengedett majom komponense)
        Ha komple[k1] $\neq$ 0 és komple[k2] = 0 akkor
            komple[k2] := idő
        különben ha komple[k2] $\neq$ 0 és komple[k1] = 0 akkor
            komple[k1] := idő
        Elágazás vége
    Ciklus vége
S86 eljárás vége

Időben visszafelé mindegyik kézelengedéskor a következőket tesszük: először megállapítjuk a két csúcs komponensének számát a k1 és k2 változókban; majd ha az egyik leesési ideje 0, akkor, a másiké pedig ismert (vagyis az ebben az időpontban még kapcsolódik az 1-es csúcshoz) a 0 leesési idő helyére az aktuális idő értéket írjuk. Amennyiben az elsővel is végeztünk, akkor minden komponens megfelelő idővel rendelkezik, és mivel minden csúcsról tudjuk, hogy melyik komponens része, így a csúcsok leesési idejét is ismerjük.

Feladatok

1. Készítsük el az algoritmus egy olyan változatát, ahol a csúcsokba bejövő élek (vagyis a lábak) listájába csak azokat az éleket vesszük föl, amelyek mindvégig megmaradnak. Nézzük meg, hogy mely adatok tárolása fölösleges, és ne kezeljük őket.

2. Engedjük meg, hogy a már leesett majmok is összekapcsolódjanak. Ez ugyan nincs hatással a leesés idejére, de az algoritmusunk működését befolyásolja. Készítsük el a helyes, lehetőleg hatékony megoldást adó változatot, amelyben megengedett a majmok kézelengedése a földön is. Gondoljunk arra, hogy akár sok-sok szétkapcsolódás is lehet a leesés után.