Az előző részekben a szélességi és a mélységi keresés, illetve bejárás algoritmusaival ismerkedtünk meg. Nézzük meg példák segítségével, hogy miként használhatók föl ezek az algoritmusok gráfokra visszavezethető problémák megoldására. A továbbiakban több olyan feladatot is megvizsgálunk, amelyek a Nemes Tihamér Verseny és az Informatika OKTV Programozói kategóriájában lettek kitűzve. A feladatok teljes szövege elérhető a `http://nemes.inf.elte.hu` oldalon, ezért itt a cikkben csak a probléma lényegét írjuk le, vagy a feladat szövegéből idézünk.

1. probléma (OKTV 2014/15, második forduló, 3. feladat). Bergengócia vasúthálózata olyan, hogy bármely városból bármely másik városba csak egyféleképpen lehet eljutni. Minden vonat a fővárosból (1-es sorszámú város) indul, és valamely olyan városig megy, ahonnan már nincs tovább vasúti pálya. A városok száma N ($2 \leq N \leq 10\,000$), a vasúti pálya bármely városból legfeljebb 10-felé ágazhat. Két város között a vonatút hossza a köztük levő `vasútállomások száma + 1`. Készítsünk programot, amely megadja a leghosszabb olyan vonatút hosszát, ahol a vasút nem ágazik el!

A feladat leírásából azonnal látható, hogy a városok tekinthetők egy (irányítatlan) gráf csúcsainak, a közvetlen vasúti kapcsolatok pedig a csúcsok közötti éleknek. Mivel bármely két kiválasztott város között pontosan egy útvonal van, ezért a gráfban nincs kör. Azt is tudjuk még, hogy az 1-es számú város kitüntetett, innen indulnak a vonatok, tehát a gráf azon részét kell vizsgálnunk, amely innen elérhető. A fővárosból nyilván minden városhoz vezet vonatút, ezért egy összefüggő gráfunk van. A feladatot leíró gráf tehát körmentes és összefüggő, vagyis egy fa, amelynek az 1-es számú csúcsa a gyökere.

A feladatban akár 10 000 város is szerepelhet, ami azt jelenti, hogy egy 0 vagy 1 értékeket tartalmazó szomszédsági mátrix a legegyszerűbb esetben $10\,000 \times 10\,000 \times 1$ bájton tárolható, tehát a tárolása 100 MB memóriaterületet igényel. Most azonban ennél sokkal kevesebb hely is elegendő, mivel egy csúcshoz legföljebb 10 él csatlakozik. Tároljuk tehát egy $10\,000 \times 10$-es két dimenziós tömbben a kapcsolatokat. Az `él[i,j]` legyen az i-edik csúcs j-edik szomszédja (a szomszédok sorrendje most nem lényeges, helyezzük el őket egyszerűen a bemenetről való olvasás sorrendjében). Fölvehetünk még egy `fok[i]` táblázatot, amely megadja, hogy az i-edik csúcsnak hány szomszédja van (ez a csúcs fokszáma). Ebből tudni fogjuk, hogy az `él[i,j]` tömb első `fok[i]` számú eleme egy szomszédos csúcs sorszáma, a nagyobb indexű elemek nem jelentenek semmit.

A feladat megoldásához először készítsük el a bemenet földolgozásával az `él` és a `fok` tömböket (melyek indexelése most induljon 1-től). Adott tehát N város 1-től sorszámozva, és valahány szomszédsági kapcsolat, amelyek számpárokként olvashatók a bemeneti állományból.

```
Előkészítés
    Be: N
    fok[minden csúcsra] = 0
    Ciklus amíg nincs vége a bemenetnek
        Be: csúcs1,csúcs2
        fok[csúcs1] := fok[csúcs1]+1
        él[csúcs1][fok[csúcs1]] := csúcs2
        fok[csúcs2] := fok[csúcs2]+1
        el[csúcs2][fok[csúcs2]] := csúcs1
    Ciklus vége
Előkészítés vége
```

Folytassuk a megoldás lényegi részével. Kérdés, hogy vajon milyen módon találhatnánk meg a leghosszabb elágazás nélküli rész(ek) hosszát? A megismert gráfalgoritmusok közül a mélységi bejárás tűnik a feladathoz megfelelőnek. Legyen a fa gyökere, tehát az 1-es csúcs, a start csúcs. A mélységi bejárás a pontosan két szomszéddal rendelkező csúcsokon egyenesen halad előre, amíg egy olyan csúcshoz nem ér, amelynek csak egy szomszédja van – ezeket a fában levélnek hívjuk – vagy egy olyan csúcshoz, amelynek kettőnél több szomszédja van. Ezt a tulajdonságot kihasználva módosítsuk az algoritmust úgy, hogy számítsa ki a keresett útvonalak hosszát.

A fa önmagában is egy rekurzív struktúra, hiszen bármely csúcs tekinthető gyökérnek, amelyből szintén egy fa ágai nőnek. A bejárás során bármely csúcsnál tartunk, az onnan kiinduló, még el nem ért részgráf szintén egy fa, amelynek a gyökere az aktuális csúcs. Induljunk ki tehát a rekurzív mélységi bejárás algoritmusából, és módosítsuk azt. A neve legyen `MBR_hossz`, és tartalmazzon még egy paramétert, amely azt mutatja, hogy milyen hosszú egy megfelelő, eddig talált út, amely az aktuális csúcshoz vezet. Ha a hívó 0 értékkel hív egy csúcsot, azzal jelzi, hogy 2-nél magasabb fokszámú, tehát nem lehet egy keresett út része. Itt legföljebb indulhat egy megfelelő út. A hívó nullánál nagyobb argumentuma jelzi, hogy egy megfelelő úton járunk, egy kettes fokszámú szomszédtól érünk ide, és az adott csúcstól függ, hogy mi a folytatás. Az eredmény előállításához a rekurzív eljárást alakítsuk függvénnyé, amely adja vissza a belőle induló fa rész leghosszabb megfelelő útvonalának hosszát. Így a fa gyökerének hívása a teljes gráfban található leghosszabb megfelelő út hosszát adja, vagyis a feladat megoldását. Rekurzív függvényünk tehát bármely csúcsnál a csúcs fokszáma és a kapott hossz argumentum alapján a következőket teheti:

- ha egy levélnél vagyunk, akkor az argumentumként kapott hossznál eggyel nagyobb értéket kell visszaadnunk, hiszen egy megfelelő út végére értünk;

- ha egy olyan csúcsnál járunk, amelynek fokszáma kettő, akkor meghívjuk a rekurzív függvényt eggyel nagyobb hossz argumentummal a még el nem ért szomszédos csúcsra, és visszaadjuk a hívónak az onnan kapott értéket;
- ha a fokszám meghaladja a kettőt, akkor meghívjuk `0` hosszúsággal az összes, még be nem járt faághoz vezető szomszédos csúcsra a rekurzív függvényt. Ezen hívások eredményeit és az aktuális csúcs hívásakor kapott `hossz+1`-et összehasonlítjuk, és a legnagyobb értéket adjuk eredményként vissza. A `hossz` növelése itt is érthető, hiszen az aktuális csúccsal szintén egy megfelelő útvonal végére értünk.

Látszik, hogy a `hossz` paraméterben kapott érték növelése az elágazás mindegyik részében szerepel, de a gondolatmenetet jobban tükrözi, ha nem emeljük ki az elágazások elé. Ezek alapján a következő algoritmust készíthetjük el:

```
Leghosszabb elágazás nélküli út hossza(gráf, él, fok)
    jártunk[minden csúcsra] := nem
    MBR_hossz(0)
Leghosszabb elágazás nélküli út hossza vége

MBR_hossz(csúcs,hossz)
    jártunk[csúcs] := igaz
    Ha fok[csúcs] = 1 akkor
        MBR_hossz := hossz+1
    Különben ha fok[csúcs] = 2 akkor
        szomszéd := el[csúcs][1]
        Ha jártunk[szomszéd] akkor szomszéd := el[csúcs][2]
        MBR_hossz := MBR_hossz(szomszéd,hossz+1)
    Különben
        hossz := hossz+1
        Ciklus sz := 1-től fok[csúcs]-ig
            szomszéd := él[csúcs][sz]
            Ha nem jártunk[szomszéd] akkor
                akthossz := MBR_hossz(szomszéd,0)
                Ha akthossz > hossz akkor hossz := akthossz
            Elágazás vége
        Ciklus vége
        MBR_hossz := hossz
    Elágazás vége
MBR_hossz vége
```

Az algoritmus helyes eredményt ad minden olyan gráfra, amelyben legalább egy él van az induló csúcsból, de a feladat leírása alapján ez föltételezhető. A feladat megoldását tehát megkapjuk a mélységi bejárás rekurzív algoritmusának módosításával.

2. probléma (Nemes Tihamér Verseny 2014/15, 9–10. osztályosok, 2. forduló, 3. feladat). Egy kémszervezetben minden tagnak legfeljebb két beosztottja lehet. Az üzenetek a tagoktól 1 nap alatt jutnak el a közvetlen beosztottjaikhoz. A főnök az 1-es sorszámú tag. Készíts programot, amely megadja, hogy a főnöktől induló üzenetet az üzenetküldéstől számítva hányadik napon kapja meg a legtöbb tag!

A standard bemenet első sora a tagok számát tartalmazza ($2 \leq N \leq 10\,000$), majd $N-1$ sorban a kapcsolatok (A_i, B_i) leírása következik, ami azt jelenti, hogy az A_i sorszámú tag közvetlen beosztottja a B_i sorszámú tag ($1 \leq A_i \neq B_i \leq N$).

A feladat leírásából látszik, hogy most is egy fával van dolgunk, amelynek a gyökere az 1-es számú csúcs. A bemeneti számpárok rendezettek, ezért tekintsük a gráfot irányítottnak. Mivel minden csúcs legföljebb két kimenő élt tartalmaz, ezért az előző feladathoz hasonlóan érdemes egy $2 \cdot N$ méretű `él` táblázatot fölvenni a kapcsolatok tárolására. A `fok` tömbre most nincs is szükségünk: jelöljük az `él` táblázat második oszlopában negatív értékkel, ha csak egy kivezető él van az adott csúcsból, vagy írjunk mindkét oszlopba negatív számot, ha a csúcs egy levél.

A feladat megoldása egy maximális érték kiválasztását jelenti. Ezt akkor tudjuk megtenni, ha bejárjuk a fát és minden csúcsnál tudjuk, hogy milyen távol van a kezdő csúcstól. Úgy tűnik, hogy a szélességi bejárás segíthet a megoldásban, mivel az algoritmus a `start` csúcstól vett távolságuk sorrendjében éri el a csúcsokat. Most az útvonalakat nem akarjuk megismerni, ezért a `honnan` táblázatra nincs szükségünk, sőt a `jártunk` tömb sem kell, mert egy fában nem érhetünk el egy csúcsot kétféle úton. De szükségünk lesz egy N méretű táblázatra, amelybe a bejárás közben bejegyezzük a csúcsok távolságát az 1-es csúcstól.

A bemeneti adatok feldolgozása és az él táblázat kitöltése után következik a bejárás az irányított gráfban, ami csak annyiban bővül az eredeti szélességi bejáráshoz képest, hogy minden elért csúcs esetén megadja, hogy a szomszédjai távolsága eggyel nagyobb a csúcs távolságánál.

```
Szélességi bejárás - távolságok(él)
    Sor_legyen_üres
    Sorba(1)
    távolság[1] = 0
    Ciklus amíg nem üres a Sor
        Sorból(csúcs)
        szomszéd := él[csúcs][1]
        Ha szomszéd > 0 akkor
            Sorba(szomszéd)
            távolság[szomszéd] = távolság[csúcs]+1
            szomszéd := él[csúcs][2]
            Ha szomszéd > 0 akkor
                Sorba(szomszéd)
                távolság[szomszéd] = távolság[csúcs]+1
            Elágazás vége
        Elágazás vége
    Ciklus vége
Szélességi bejárás - távolságok vége
```

A bejárás lefutása után minden, az 1-es csúcsból elérhető csúcsra megkapjuk a tőle való távolság értékét, és nincs más dolgunk, mint a maximumot kiválasztani a `távolság` tömbből.

Feladatok:

1. Módosítsuk az 1. probléma rekurzív megoldását úgy, hogy ne a fa gyökétől induljon a bejárás. Vajon helyes eredményt kapunk ebben az esetben is? Milyen csúcsoktól indulva kapunk helyes eredményt?

2. Módosítsuk a mélységi bejárás nem rekurzív algoritmusát úgy, hogy az 1. probléma megoldását adja!

3. Olvassuk el a Nemes Tihamér Verseny 2014/15, 9–10. osztályosok, 3. forduló, 2. feladat feladatát, és gondoljuk végig, hogy pontosan miben különbözik az eddig megoldott problémáktól! Oldjuk meg az eddig megismertek alapján (kisebb bemeneti értékekre) a feladatot! Gondoljuk végig, mire volna szükség a megoldáshoz nagyobb bemeneti értékek esetén!