Kedves Versenyző! A megoldások értékelésénél csak **a programok futási eredményeit** vesszük tekintetbe. Ezért igen fontos a **specifikáció pontos betartása**. Ha a feladat szövege adatok valamilyen állományból történő beolvasását írja elő, és a program ezt nem teljesíti, akkor a feladatra nem adunk pontot (akkor sem, ha egyébként tökéletes lenne a megoldás); az objektív értékelés érdekében ugyanis a programszövegekben egyetlen karaktert sem javítunk, s az előre megadott javítási útmutatótól semmiben nem térünk el. A programokat csak a feladatkiírásban leírt szabályoknak megfelelő adatokkal próbáljuk ki, emiatt nem kell ellenőrizni, hogy a bemenő adatok helyesek-e, illetve a szükséges állományok léteznek-e (sőt ezért plusz pont sem jár). Ha a programnak valamilyen állományra van szüksége, akkor azt mindig az aktuális könyvtárba kell rakni. Az állomány neve minden esetben rögzített.

1. feladat: Lovak (50 pont)

Lótenyésztők sok generációra visszamenőleg tartják nyilván lovaik leszármazását. A lovakat sorszámukkal azonosítjuk, vagy mindkét szülőjüket ismerjük, vagy csak az egyiket, vagy pedig egyiket sem. Így ismerhetjük a lovak nagyon régi őseit is. Előfordulhat, hogy egy ló egyes ősei többféle leszármazási ágon is ősök.

Készíts programot **`lovak.cbp (main.cpp)`**, amely adott ló esetén megadja, hogy

- hány olyan őse van, amelyik több leszármazási ágon is ős;
- melyik az a ló, ami a legtöbb leszármazási úton szerepel! (Leszármazási út mindig olyan lótól indul, aminek nem ismerjük a szüleit és olyan lónál ér véget, aminek nem ismerjük az utódait.)

A **`lovak.be`** állomány első sorában a nyilvántartott lovak N és leszármazási kapcsolatok M száma van ($1 \le N,M \le 100000$). A következő M sor mindegyike 2 egész számot tartalmaz, egy szóközzel elválasztva, az első szám egy ló sorszáma, a második pedig az egyik szülőjének sorszáma. Az utolsó sorban egy ló L sorszáma van ($1 \le L \le N$).

A **`lovak.ki`** állomány első sorába azon lovak számát kell írni, ahányan többszörös ősei az L lónak, a második sorba pedig azon ló sorszámát, amely a legtöbb leszármazási úton szerepel.

Példa:

lovak.be	lovak.ki
10 11 10 3 1 2 1 3 2 4 2 5 3 5 3 6 5 7 4 8 6 8 6 9 1	3 3

2. feladat: Darabolás (50 pont)

Adott egy fémrúd, amelyet megadott számú és hosszúságú darabokra kell felvágni. A darabok hosszát milliméterben kifejezett értékek adják meg. Olyan vágógéppel kell a feladatot megoldani, amely egyszerre csak egy vágást tud végezni. A vágások tetszőleges sorrendben elvégezhetőek. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk. A célunk optimalizálni a műveletsor teljes költséget.

Készíts programot **darabol.cbp (main.cpp)**, amely

- Kiszámítja a vágási műveletsor optimális összköltségét.
- Megad egy olyan vágási sorrendet, amely optimális költséget eredményez.

A **darabol.be** szöveges állomány első sora egy egész számot tartalmaz, a darabok ***N*** számát ***(0<N≤1000)***. A második sor ***N*** darab pozitív egész számot tartalmaz egy-egy szóközzel elválasztva, a darabok hosszát. A második sorban szereplő számok nem nagyobbak, mint ***1000***.

A **darabol.ki** szöveges állomány első sorába egyetlen számot, a vágási műveletsor optimális összköltségét kell írni! A további ***N-1*** sor mindegyikébe két egész számot kell írni, egy szóközzel elválasztva. Az első szám legyen az adott lépésben kettévágott léc hossza, a második szám pedig az egyik keletkező darab hossza. Minden sor csak olyan hosszúságú darab kettévágását tartalmazhatja, amelyből a korábbi lépések során több keletkezett, mint az azóta elvégzett lépések által felhasználtak száma. Ha több vágássorozattal is el lehet érni az optimális költséget, akkor ezek közül bármelyiket meg lehet adni.

Példa:

darabol.be

```
5
2 5 2 7 10
```

darabol.ki

```
55
26 10
16 7
9 4
4 2
```

3. feladat: Háromszög (50 pont)

Adott egy N oldalhosszúságú háromszög. A háromszög - az elemek elrendezését tekintve - hasonló a Pascal háromszögre, tehát minden elem fölött balra es jobbra (ha az nem a háromszög bal vagy jobb szélén helyezkedik el) található egy-egy elem.

Adott a háromszög elemeivel együtt. Ez egy játék, amit két játékos játszik. A háromszög oldalának hossza páros. Minden lépésben a soron következő játékos a háromszög három oldala közül választhat, és amelyik oldalt választja, azon oldal mentén levő elemek az ő birtokába kerülnek. Ezeket az elemeket eltávolítja a háromszögből, ezáltal egy (N-1) méretű háromszöget kapunk. Most a második játékos lép, hasonló szabály alapján.

Készíts programot **harom.cbp (main.cpp)**, amely kiszámítja, hogy maximálisan hány pontot tud összegyűjteni az első játékos, feltéve, hogy a második játékos is úgy játszik, hogy a legtöbb pontot szerezze meg. Pontszám alatt a birtokában levő elemek összegét értjük

A **harom.be** szöveges állomány első sora egyetlen egész számot tartalmaz, a háromszög ***N*** oldalhosszát ***(0<N≤24)*** (***N*** páros szám)**.** A következő ***N*** sor tartalmazza a háromszög leírását. Az állomány i-edik sora i-1 darab pozitív egész számot tartalmaz egy-egy szóközzel elválasztva, ami a háromszög i-1-edig sora lesz. Minden háromszögbeli szám értéke legfeljebb 100.

A **harom.ki** szöveges állomány első sorába egyetlen számot kell írni, ami az első játékos által megszerezhető maximális pontszám!

Példa:

harom.be

```
4
1
2 3
```

harom.ki

```
15
```

```
3 4 2
4 2 3 1
```

4. feladat: ProtoNet (50 pont)

A ProtoNet számítógépes hálózat úgy alakult ki, hogy eredetileg különálló hálózatokat összekapcsoltak. Mindegyik hálózat saját, egyedi protokollt használt. Az egyes részhálózatok az összekapcsolás után is a régi módon, azaz saját protokollt használva működnek. A hálózati hardvert azonban felszerelték olyan szoftverrel, amely képes bármelyik két protokoll közötti váltásra. A teljes hálózat most úgy működik, hogy ha egy X csomópontból közvetlenül egy Y csomópontba kell csomagot küldeni, és X valamint Y nem azonos részhálózathoz tartozik, akkor előbb protokollváltást kell végrehajtani. A hálózati működést optimalizálni szeretnék. Ez azt jelenti, hogy olyan szoftvert kell készíteni, amely meghatározza, hogy ha egy adott **A** csomópontból egy másik **B** csomópontba kell küldeni a csomagot, akkor milyen útvonalat kell választani ahhoz, hogy a protokollváltások száma minimális legyen.

Készíts programot **`proto.cbp (main.cpp)`**, amely

- Kiszámítja, hogy az **A** csomópontból a **B** csomópontba küldendő csomag esetén legkevesebb hány protokollváltás szükséges!
- Megad egy olyan **A**-ból **B**-be vezető útvonalat, amelyen a protokollváltás a lehető legkevesebb!

A **`proto.be`** szöveges állomány első sora két egész számot tartalmaz, a csomópontok ***N*** számát ***(0<N≤1000)***, és a részhálózatok ***M*** számát ***(0<M≤N)***. A következő ***M*** sorban csomópontok sorszámai szerepelnek, minden sort a ***0*** szám zárja. Az egy sorban felsorolt csomópontok alkotnak egy részhálózatot. Minden csomópont pontosan egy részhálózathoz tartozik. A következő sor a csomópontok közötti közvetlen átviteli kapcsolatok ***K*** számát tartalmazza ***(0<K≤20000)***. A következő ***K*** sor mindegyike egy ***X Y*** számpárt tartalmaz, ***(1≤ X, Y≤N, X≠Y)***, ami azt jelenti, hogy az ***X*** és az ***Y*** csomópont közvetlenül össze van kötve átviteli csatornával. Az utolsó sor egy ***A B*** számpárt tartalmaz, ezek azon csomópont sorszámai ***(A≠B)***, amelyek közötti átvitelt vizsgáljuk. Az egyes részhálózatok nem feltétlenül összefüggőek!

A **`proto.ki`** szöveges állomány első sorába egyetlen számot kell írni, ami az ***A.*** részfeladat megoldása. A második sorba egy az **A** csomópontból a **B** csomópontba vezető olyan útvonalat kell megadni, amely esetén a protokollváltások száma a lehető legkevesebb! Ha nincs útvonal **A** és **B** között, akkor az első sorba a **`-1`** értéket kell kiírni!

Példa:

`proto.be`

```
7 3
1 3 5 0
4 2 6 0
7 0
10
1 2
1 3
1 4
2 3
2 5
3 5
3 6
4 7
5 6
5 7
1 7
```

`proto.ki`

```
1
1 3 5 7
```