Műholdas telefon

A Föld körül keringő, műholdakból álló Iridium hálózat az egész Földet lefedi, és interkontinentális hívást tesz lehetővé helyi mobilhálózat nélkül. A műholdak 1-től N-ig számozottak. A hívás a Földről először egy onnan látható műholdon, majd más műholdakon keresztül végül visszairányul a Földre, a másik hívó félhez. Fel akarjuk hívni új-zélandi nagybátyánkat, hogy megkérdezzük, milyen az idő odalent.

Adott két adatállomány. Az első szöveges: RENDSZER.TXT, ez tartalmazza az első sorban a műholdak számát (N) és a többi sorban az egész Iridium rendszer hibátlan működése esetén műholdpárok között használható közvetlen csatornák percdíját (egy műhold kb 4 másikkal tud kommunikálni közvetlenül). Minden sorban 3 adat áll,és a következő a szerkezetük:

az egyik műhold száma -a kisebb számú elöl
szóköz
a másik műhold száma -a nagyobb számú hátul
szóköz
a kapcsolatukon keresztüli telefonálás percdíja -egész forint

Egy ilyen kapcsolat nem szerepel kétszer, két irányból, és csak azok a kapcsolatok szerepelnek, amelyek a hibátlan rendszerben használhatók.
A második állomány a HIVAS.TXT, ez tartalmazza az első sorban a hívó felek felett található egy-egy műhold számát szóközzel elválaszva,a többi sorban pedig az aktuálisan nem működő vagy drágábban működő kapcsolatok adatait, az előző formátumban (ha egy kapcsolat pillanatnyilag üzemen kívül van, akkor annak a percdíja 0).
Feladat: Írjunk programot, amely megtervezi az adott hívás legolcsóbb útvonalát, és kiírja az egyik hívótól a másikig érintett műholdak számát sorrendben, valamint a hívás percdíját(a használt szakaszok percdíjai összeadódnak). Azt is írja ki, ha a rendszerhibák miatt nem köthető össze a két fél.

Adatszerkezet

A hívás résztvevői: hívó, hívást fogadó, műhold, kapcsolat (2 műhold között).

A műholdak számát nem ismerjük fordítási időben, ezért a tárolásuk láncolt listával oldható meg hatékonyan. A műholdakat tartalmazó láncolt lista egy irányba láncolt és nem strázsás. Minden műhold listaelemről az abból a műholdból látható másikak adatai függnek egy újabb láncolt listán. Ezek a műhold kapcsolatai.

Ha az adatszerkezet felépült a fájlok beolvasásával, le kell rajta futtatni egy Dijkstra algoritmust. A Dijkstra futása kezdetekor (a hívott fölöttit kivéve) minden műhold eléretlen és befejezetlen. Az egyes műholdak hívottól való elérésének felső becslését nyilvántartjuk, és élmenti (vagyis kapcsolatmenti) javításokat végzünk. Minden műholdhoz nyilván kell tartani az a közvetlen szomszédját, ahonnan befejezetté válik, ez a beállító-ja (egyszerűbb a nyilvántartás, ha sikeres élmenti javításkor a “beállító” az a műhold lesz, amelyikből az élmenti javítást végeztük és külön adatban tároljuk a befejezettséget).

A Dijkstra futása közben egy lépésként egy műhold összes szomszédja felé végzünk élmenti javítást. Ehhez ismernünk kell egy adott műhold összes kapcsolatát, ez teszi indokolttá a kapcsolatok két oldali tárolását (vagyis az 1-es és 2-es műhold kapcsolatát nyilvántartjuk az 1-es és a 2-es műhold felől is).

Nyilván kell tartani a következőket egy adott műholdról: befejezett-e már (vagyis megtaláltuk- e már a legolcsóbb útvonalat ide a hívottól), mennyi az eddig meghatározott legkisebb

percdíjú összeköttetése a hívottal, és melyik műholddal közös közvetlen kapcsolata állította be (“Beállító”) a végleges (befejezése utáni) legrövidebb percdíjat a hívó és az adott műhold között. A beállító műholdat egy az arra a műhold típusra mutató pointer tartalmazza, ami nullpointer, ha még nincs elérve a műhold.

A KAPCS típus adatai:
a kapcsolódó műhold száma (int szam),
a kapcsolat percdíja (int dij),
a következő kapcsolatra mutató pointer (kapcs* kov).

A MUHOLD típus adatai:
a műhold száma (int szam),
a kapcsolatai láncolt listájának feje (kapcs* khead),
az elérése költségének felső becslése (int fb), (-1, ha nincs elérve)
a következő műholdra mutató pointer (muhold* kov),
a beállító műholdra mutató pointer (muhold* beallito), (nullpointer, ha nincs elérve)
befejezettség (char bef, lehet: ’I’/’N’).

A műholdak láncolt listájának szerkezete:
Láncolt lista feje: muhold* MHEAD
Következő műhold pointere: nullpointer, ha ez az utolsó elem
Rendezettség: a műholdak láncolt listájának rendezettsége lényegtelen, mivel futás közben a meghatározó tulajdonság a műholdak kapcsolata, ami nem feltétlenül sorba rendezhető.
Strázsák: itt nincs szükség strázsákra, mert a műholdlista az üresen legenerálástól és a felszabadítástól eltekintve nem változik, ellentétben a kapcsolatokkal.

Egy műholdon lógó kapcsolatok láncolt listájánal szerkezete:
Láncolt lista feje: kapcs* khead (ez a muhold típus egyik paramétere)
Következő kapcsolat pointere: nullpointer, ha nincs több kapcsolata az adott műholdnak
Rendezettség: a kapcsolódó műholdak száma szerint rendezett, ez azért indokolt, mert a program futása közben egyes műholdak kapcsolatainak módosítása, keresése egyszerűbb rendezett listában.
Strázsák: kétstrázsás lista. A strázsák használata a kapcsolatok többszöri módosítása miatt egyszerűbb pl. beszúráskor.

Alap függvények:

Létrehoz egy kapcsolatok nélküli, eléretlen műholdakból álló, n-hosszú láncolt listát:
```
muhold* M_lista_Generator(int n)
```

Felszabadít egy kapcsolatok nélküli műholdlistát:
```
void M_free(muhold* MHEAD)
```

Felszabadítja az összes műhold összes kapcsolatát:
```
void K_free(muhold* MHEAD)
```

2 megadott műhold (n és k) között megadott percdíjú kapcsolatot hoz létre:
```
void osszekapcs(muhold* MHEAD, int n, int k, int ujdij)
```

A már meglevő kapcsolat percdíját megváltoztatja, vagy kitörli a kapcsolatot, ha az új percdíj 0:
`void kapcs_frissit(muhold* MHEAD, int m1, int m2, int ujdij)`

Egy műhold (n) kapcsolatai mentén élmenti javítást végez:
`void javito(muhold* MHEAD, int n)`

Végrehajtja a Dijkstra algoritmus egy iterációját (megkeresi a legolcsóbban elérhető, befejezetlen műholdat, azt befejezi és a kapcsolatai mentén javítja a szomszédai elérési költségének felső becslését):
`muhold* Dijkstra(muhold* MHEAD)`

Megmutatja, hogy vége van-e a keresésnek (int-et ad vissza, 0-t ha még nincs vége a keresésnek, 1-et ha megvan a legolcsóbb út a kezdő- és végpont között, 2-t ha minden műhold befejezett vagy eléretlen és a hívott fél műholdja eléretlen, vagyis nem lehet a hívó feleket összekötni):
`int vege(muhold* MHEAD, int hivo, int hivott)`

További függvények:

A rendszer kiépítéséhez szükséges függvények:

```
void K_rend_beszur(muhold *mp, kapcs* uj)
muhold* M_Lista_generator(int n)
kapcs* K_hely(muhold* mp, int n)
void szetkapcs(muhold* MHEAD, int m1, int m2)
void osszekapcs(muhold* MHEAD, int m1, int m2, int ujdij)
void kapcs_frissit(muhold* MHEAD, int m1, int m2, int ujdij)
muhold* rendszer_epit(int* hivo, int* hivott)
```

Az algoritmus futása közben szükséges függvények:

```
muhold* muholdkereso(muhold* MHEAD,int n)
void javito(muhold* MHEAD, int n)
int vege(muhold* MHEAD, int hivott)
muhold* LKFBEBM(muhold* MHEAD)
muhold* Dijkstra(muhold* MHEAD)
```

Az algo. Lefutása után szükséges függvények:

```
void utvonal_kiir(muhold* mp)
void K1_free(muhold* mp)
void M1_free(muhold** MHEAD)
void K_free(muhold* MHEAD)
void M_free(muhold** MHEAD)
```

Fájlokat kezelő függvény:

```
muhold* rendszer_epit(int* hivo, int* hivott)
```

A függvény pszeudokódja:

```
rendszer.txt megnyitása
Ha nem sikerül megnyitni a rendszer.txt-t
     hibajelzés
     NULL visszatérés
muholdak számának beolvasása
Ha nem sikerül beolvasni a muholdak számát
     hibajelzés
     rendszer.txt bezárása
     NULL visszatérés
Muholdlista legenerálása
Amíg új kapcsolatot sikerült beolvasni loop
     beolvasott kapcsolat beillesztése a muholdak közé
fájl bezárása
(eddig az rendszer.txt fájllal foglalkozik a program)
Ha a hivas.txt-t nem lehet megnyitni
     hibajelzés
     eddigi muholdlistával visszatérés
Hívó felek muholdjainak beolvasása és eltárolása
Ha nem sikerült beolvasni a hívó feleket
     hibajelzés
     fájl bezárása
     eddigi muholdlistával visszatérés
loop míg sikerül beolvasni kapcsolatváltozást
     beolvasás
     kapcsolat frissítése
fájl bezárása

hívó muhold beállítása
visszatérés a kész muholdlista fejével
```

Így elkerülhető a memóriaszivárgás, mivel minden esetben visszaadja a hiba miatt félkész műholdlista fejét ami alapján az később felszabadítható.

A végeredményt meghatározó és kiíró függvény:

```
void utvonal_kiir(muhold* mp)
{
    printf("\n\nTeljes Díj: %d\nÚtvonal: ", mp->fb);
    if(mp==NULL)return;
    while(mp!=NULL)
    {
        printf(" %d ",mp->szam);
        mp=mp->beallito;
    }
}
```

Ez a függvény választ ad a program fő feladataira:
Mennyi a legolcsóbb telefonkapcsolat percdíja?
Mi az egyik legolcsóbb útvonal a hívó és hívott műholdja között?

Az első kérdést a Dijkstra algoritmusban szereplő felső becslések útján könnyen meg lehet válaszolni, mivel a tanultak alapján egy befejezett gráfcsúcs felső becslése a gyökérponttól oda vezető legrövidebb irányított út hossza. A mostani esetben a gráfcsúcs műholdnak, a gyökérpont pedig a hívó műholdjának felel meg.

A második kérdés megválaszolásához minden elért műholdhoz eltároljuk, hogy melyik szomszédja az, amelyik beállította a legkisebb felső becslését. Ha a hívott műholdtól kezdve mindig átlépünk az aktuális műhold felső becslését beállító szomszédjába, a keresett legrövidebb úton végiglépegetve végül elérünk a másik hívó fél műholdjába. Eközben kiiratjuk a legrövidebb úton lévő műholdak számát sorrendben, ez pedig meghatározza a keresett legrövidebb utat.

Main:

```
int main()
{
    int hivo, hivott, i=0;
    muhold* head=rendszer_epit(&hivo, &hivott);
    while(vege(head, hivott)==0)
    {
        Dijkstra(head);
        i++;
    }
    if(vege(head, hivott)==1) utvonal_kiir(muholdkereso(head,
hivott));
    else printf("\nA hívó felek nem tudnak beszélni");
    printf("\nDijkstra iterációk száma: %d\n", i);
    K_free(head);
    M_free(&head);
    return 0;
}
```

Először megépül a műholdrendszer, aztán elkezd futni a Dijkstra algoritmus.
Az algoritmus minden iterációja után a void vege(...) függvény ellenőrzi, hogy le kell-e állítani a keresést, és mi az aktuális helyzete a keresésnek.

Ha a vege fv. Leállítja a keresést azt két okból teheti: az algoritmus megtalálta a keresett legrövidebb utat, vagy belátható, hogy nem létezik a keresett út. Ettől a visszatérési értéktől függően ad üzenetet a program.

Ha sikerült megtalálni a legrövidebb utat, akkor annak az összköltségét és magát az útvonalat az utvonal_kiir függvény kiírja a standard outputra. Mint extra információ, a program kiírja, hogy a Dijkstra algoritmus hány iterációja után állt le a keresés.

Végül minden esetben felszabadul először az összes kapcsolat, majd a műholdak.