Gráf tárolása

A programban egyidejűleg egy gráfot kezelünk. Futtatásához nincs szükség külső könyvtárakra, csak szabványos függvényeket használ. A gráfon műveleteket végezhetünk, melyeket egy menürendszerből választhatunk ki. A program funkciói:

- Betölthetünk a programba egy gráfot fájlból (a fájlnév nem tartalmazhat szóközt), melynek az első sora tartalmazza szóközzel elválasztva a csúcsok és az élek számát. A további sorokban egy-egy él adatai vannak, <csúcs1> <csúcs2> <súly> formátumban.
- A programban éppen tárolt gráfot fájlba menthetjük (a fájlnév nem tartalmazhat szóközt), a fent leírt formátumban.
- A gráf-építő segítségével módosíthatjuk az aktuálisan betöltött gráfot, létrehozhatunk új csúcsot, új élt, módosíthatjuk egy létező élnek a súlyát, illetve törölhetünk csúcsot, élt.
- Szélességi-/mélységi bejárást végezhetünk a gráfon. A felhasználó által megadott fájlba (a fájlnév nem tartalmazhat szóközt) kerül mentésre a bejárás sorrendje, soronként egy csúcs sorszámát tartalmazva, az első sorban található a kiindulópont.
- A program kér a felhasználótól két csúcspontot, melyek között megállapítja a legrövidebb útvonalat, majd ennek a hosszát és az útvonalat a felhasználó által megadott fájlba (a fájlnév nem tartalmazhat szóközt) menti. Az első sor tartalmazza a hosszt, a következő sorokban az útvonal található, soronként egy csúcs sorszáma, az első a kiindulópont, az utolsó a végpont. Amennyiben a két megadott csúcspont nem összefüggő, a fájl első sorában a *Nincs utvonal* kifejezés szerepel.

Adatszerkezet

A program fő adatszerkezete a *Graf*. Egy gráfot szomszédsági listával tárolunk el, tehát minden csúcsához a gráfnak eltároljuk, melyik csúccsal szomszédos, és milyen súllyal. A csúcsokat is egy listában tároljuk.

```
typedef struct Graf {
   CsucsLista *csucsok;
   int csucsok_szama;
   int elek_szama;
} Graf;
```

CsucsLista - duplán láncolt lista, csúcsokat tartalmazza. A listához rendezve lehet beszúrni elemeket, ez a tulajdonság hasznos lesz a gráf mentésénél, mivel a csúcsok növekvő sorrendben következnek.

```
typedef struct CsucsLista {
   struct CsucsListaElem *elso, *utolso;
} CsucsLista;
typedef struct CsucsListaElem {
   Csucs *csucs;
   struct CsucsListaElem *elozo, *kov;
} CsucsListaElem;
```

Csucs - egy csúcs sorszámát és szomszédsági listáját tartalmazza.

```
typedef struct Csucs {
   int id;
   SzomszedsagiLista *szomszedok;
} Csucs;
```

SzomszedsagiLista - adott csúcs szomszédjainak listája, a megfelelő élhez tartozó súllyal. Szintén rendezett lista.

```
typedef struct SzomszedsagiLista {
   struct SzomszedsagiListaElem *elso, *utolso;
} SzomszedsagiLista;
typedef struct SzomszedsagiListaElem {
   int csucs;
   int suly;
   struct SzomszedsagiListaElem *elozo, *kov;
} SzomszedsagiListaElem;
```

CsucsSor - *Csucs* pointereket tartalmazó sor, szélességi bejárást segíti.

```
typedef struct CsucsSor {
   int db;
   struct CsucsSorElem *elso, *utolso;
} CsucsSor;
typedef struct CsucsSorElem {
   Csucs *csucs;
   struct CsucsSorElem *kov;
} CsucsSorElem;
```

main.c

```
void menu_kiiras()
```

Kiírja a szabványos kimenetre a menüt.

```
Graf *betoltes_menu(Graf *g)
```

A betöltés menüpontot vezérli, a betöltött gráfra mutató pointerrel tér vissza.

```
void mentes_menu(Graf *g)
```

A mentés menüpontot vezérli.

```
void graf_epito_menu(Graf *g)
```

A gráf-építő menüt vezérli, *g* gráfot módosítja.

```
void szelessegi_menu(Graf *g)
```

Szélességi bejárás menüpontot vezérli.

```
void melysegi_menu(Graf *g)
```

Mélységi bejárás menüpontot vezérli.

```
void utkereses_menu(Graf *g)
```

Legrövidebb útkeresés menüpontot vezérli.

```
void foprogram()
```

A programot összefoglaló függvény, lefutása után a program leáll.

listak.c

Listák kezeléséért felelős függvényeket tartalmazza.

```
SzomszedsagiLista *beszur_szomszedsagi_lista(SzomszedsagiLista *l, int csucs, int
suly)
```

l listába rendezve beszúr egy új elemet, visszatér a listára mutató pointerrel.

```
SzomszedsagiListaElem *keres_szomszedsagi_lista(SzomszedsagiLista *l, int csucs)
```
l listában keres elemet, visszatér az elemre mutató pointerrel.

```
void torol_szomszedsagi_lista(SzomszedsagiLista *l, int csucs)
```
l listában törli a megadott elemet.

```
void felszabadit_szomszedsagi_lista(SzomszedsagiLista *l)
```
Paraméterként kapott dinamikusan foglalt *l* listát felszabadítja.

```
CsucsLista *beszur_csucs_lista(CsucsLista *l, Csucs *csucs)
```
l listába rendezve beszúr egy új elemet, visszatér a listára mutató pointerrel.

```
CsucsListaElem *keres_csucs_lista(CsucsLista *l, int csucs)
```
l listában keres elemet, visszatér az elemre mutató pointerrel.

```
void torol_csucs_lista(CsucsLista *l, int csucs)
```
l listában törli a megadott elemet.

```
void felszabadit_csucs_lista(CsucsLista *l)
```
Paraméterként kapott dinamikusan foglalt *l* listát felszabadítja.

```
CsucsSor *push_csucs_sor(CsucsSor *s, Csucs *csucs)
```
s sorhoz fűzi hozzá a *csucs* pointert. Visszatér a sorra mutató pointerrel.

```
Csucs *pop_csucs_sor(CsucsSor *s)
```
s sorból kiveszi (törli) a következő elemet, visszatér a benne tárolt *Csucs* pointerrel.

```
void felszabadit_csucs_sor(CsucsSor *s)
```
Paraméterként kapott dinamikusan foglalt *s* sort felszabadítja.

```
bool ures_csucs_sor(CsucsSor *s)
```
s sor ürességét ellenőrzi, eredménnyel visszatér.

grafkezeles.c

Gráfot manipuláló, gráfokon számítást végző függvényeket tartalmaz.

```
void csucs_hozzaad(Graf *g, int id)
```
g gráfban létrehoz egy új csúcsot *id* számmal.

```
void csucs_torol(Graf *g, int id)
```
g gráfban törli az *id* számú csúcsot és az összes hozzátartozó élt.

```
void el_torol(Graf *g, int c1, int c2)
```
g gráfban törli a *c1* és *c2* számú csúcsok közötti élt (mindkét csúcs szomszédsági listájából kiveszi az adott csúcs sorszámát).

```
void el_modosit(Graf *g, int c1, int c2, int s)
```
g gráfban módosítja a *c1* és *c2* számú csúcsok közötti él súlyát *s*-re.

```
void el_letrehoz(Graf *g, int c1, int c2, int s)
```
g gráfban létrehoz a *c1* és *c2* számú csúcsok között egy élt, *s* súllyal.

```
Graf *letrehoz_ures_graf()
```
Létrehoz egy dinamikusan foglalt üres gráfot (nincs benne csúcs és él), visszatér a rámutató pointerrel.

```
void felszabadit_graf(Graf *g)
```
Felszabadítja *g* dinamikusan foglalt gráfot.

```
static int *init_volt(Graf *g)
```
Segéd függvény, bejárásoknál a már felfedezett csúcsok tárolására használt dinamikus tömböt inicializálja, visszatér a rámutató pointerrel.

```
static void megjelol_elem_volt(int *volt, int hossz, int e)
```
Segéd függvény, megjelöli az *e* számú csúcsot felfedezettnek a *volt* dinamikus tömbben.

```
static bool elem_volt(int *volt, int hossz, int e)
```
Segéd függvény, megállapítja, hogy *e* számú csúcs fel van-e fedezve a *volt* dinamikus tömbben, visszatér az eredménnyel.

```
static void melysegi_rekurziv(Graf *g, int k, FILE *fp, int *volt)
```
Mélységi bejárást hajt végre rekurzívan a *g* gráfban a *k* számú csúcsból kiindulva. A volt dinamikus tömbben követi, melyik csúcsokat járta be eddig. Az *fp* pointer segítségével írja ki féjlba futás közben a sorrendet.

```
bool melysegi_bejaras(Graf *g, int k, char *fajlnev)
```
Előkészíti, majd végrehajtja a mélységi bejárást a *g* gráfon a *k* számú csúcsból kiindulva. Az eredményt *fajlnev* útvonalú fájlba menti (nem tartalmazhat szóközt), amennyiben a mentés nem sikerül, *false* értékkel tér vissza. Minden más esetben *true* értéket ad.

```
bool szelessegi_bejaras(Graf *g, int k, char *fajlnev)
```
Előkészíti, majd végrehajtja a szélességi bejárást a *g* gráfon a *k* számú csúcsból kiindulva. Az eredményt *fajlnev* útvonalú fájlba menti (nem tartalmazhat szóközt), amennyiben a mentés nem sikerül, *false* értékkel tér vissza. Minden más esetben *true* értéket ad.

```
static int *init_tav(Graf *g, int a)
```
Segéd függvény, inicializálja a dinamikus tömböt, amely legrövidebb útkeresésénél majd a kiinduló csúcstól (*a*) számítva a legrövidebb út hosszát tartalmazza minden csúcsra.

```
static int leker_tav_i(int *tav, int hossz, int e)
```
Segéd függvény, visszatér a *tav* dinamikus tömb azon indexével, amely *e* számú csúcsnak tartalmazza az adatait.

```
static int leker_min_tav_i(int *tav, int *volt, int hossz)
```
Segéd függvény, visszatér a *tav* dinamikus tömb azon csúcsára mutató indexével, mely csúcs még nem volt felfedezve a *volt* dinamikus tömbben és értéke a *tav* tömbben a legkisebb. (*tav* es *volt* tömbök ugyanabban a sorrendben vannak indexelve a gráf csúcslistájának rendezettsége miatt)

```
static int *init_elozo(Graf *g)
```
Segéd függvény, inicializálja a dinamikus tömböt, amely tartalmazza minden csúcsra, hogy melyik csúcsból a legrövidebb eljutni hozzá.

```
static void legrovidebb_ut_fajlba(int *tav, int *elozo, int hossz, int cel, FILE *fp)
```
Segéd függvény, az *fp* pointer segítségével fájlba írja a *tav* és *elozo* tömbök alapján a *cel* számú csúcshoz vezető legrövidebb utat.

```
bool dijkstra_legrovidebb_ut(Graf *g, int a, int b, char *fajlnev)
```
Megvalósítja Dijkstra legrövidebb útkereső algoritmusát *g* gráfon *a* számú csúcsból *b* számú csúcsba. Az eredményt *fajlnev* útvonalú fájlba menti (nem tartalmazhat szóközt), amennyiben a mentés nem sikerül, *false* értékkel tér vissza. Minden más esetben *true* értéket ad.

fajlkezeles.c

Gráfok mentését és betöltését kezeli.

```
Graf *betolt_graf(Graf *g, char* fajlnev)
```
Beolvassa *g* gráfba a *fajlnev* útvonalú fájl (nem tartalmazhat szóközt) által meghatározott gráfot.

```
bool ment_graf(Graf *g, char* fajlnev)
```
g gráfot menti a *fajlnev* útvonalú fájlba (nem tartalmazhat szóközt). Amennyiben a mentés nem sikerül, *false* értékkel tér vissza. Minden más esetben *true* értéket ad.