Buszjáratok programozói dokumentáció

Útvonalkeresés

A megállók közötti útvonalkeresés a programban dijkstra algoritmusával van megvalósitva. Legelőször meghivjuk a függvényt a felhasználó által megadott két megállóra. Ezután végigmegyünk az induló megálló összes gyalog megtehető átszállásán, illetve ahova busszal lehet eljutni innen, és ezekre is meghivjuk újra a függvényt. Ilyenkor viszont már egyel közelebb vagyunk a célmegállónkhoz, igy egy idő után eljutunk a célmegállóhoz. Ezzel egy rekurziv programhoz jutunk. Ezzel viszont még nem oldottuk meg a teljes problémát hiszen nekünk a legrövidebb út kell. Ehhez minden alkalommal mikor újra szeretnénk hivni a függvényt az összes elérhető megállóra, ezt aszerint kell sorban megtenni, hogy innen melyiket érjük el a leggyorsabban. Ezt a megállók listájában tárolom. Igy mindig meg kell nézni és elmenteni, hogy melyik megállóba mennyi idő alatt érünk oda, illetve mivel elmentjük ezt az adatot csak akkor kell innen az adott megállóba újra meghivni a függvényt, hogyha gyorsabban értük el mint eddig. Kizárólag a gyorsabb jó nekünk, hogy ne akadjon el egy végtelen ciklusban, hiszen a gyalogos átszállásoknál nem számolunk plusz időt. Ezáltal megtaláltuk a leggyorsabb utat a kezdőból a célmegállóba. Ahhoz, hogy esetlegesen a legkevesebb átszállásos vagy más kritériumos útvonalakat keressünk, a sorbarendezés kritériumát kellene megváltoztatunk.

Adatszerkezetek

A programnak két fő adatszerkezete van, a **Jarat** és a **Megallo** ezenkivül van egy harmadik különálló, az **Ido**, illetve ezekhez segéd adatszerkezetek.

```
typedef struct Jarat {
    char nev[51];
    Ido elso_indulas, utolso_indulas, tovabbi_indulasok;
    Megallo_list *megallok;
} Jarat;

typedef struct Jarat_list {
    Jarat jarat;
    struct Jarat_list *kov;
} Jarat_list;
```

A **Jarat** adatszerkezethez tartozik egy **Jarat_list** is, ami egy **Jaratot** és a kovetkező elemre mutató elemet tartalmaz. Ezekkel van felépitve egy láncolt lista, ami a járatok listáját tartalmazza.

```
typedef struct Megallo {
    char nev[51];
    struct Megallo_list *atszallasok;
    Ido tav;
    char jarat_atszallas[51];
} Megallo;

typedef struct Megallo_list {
    Megallo *megallo;
    Ido erkezes;
    struct Megallo_list *kov;
} Megallo_list;
```

A **Megallo** adatszerkezethez is szintén tartozik egy **Megallo_list** a láncolt lista felépitéséhez. Itt viszont a **Megallo**-t is mutatóként tároljuk, mivel minden listában ugyanazokra a megállókra mutatnak, igy mindenhonnan elérjük az összes adatukat. Itt található az **Ido tav** változó is, amiben az útvonalkeresés érkezéseit mentjük el, ennek a megállón belül kell lennie, mivel minden megállóva egyszer érkezhetünk. A **Megallo_list**-en belül viszont van egy **Ido erkezes**, ez az egyes járatok szerinti érkezést tárolja, ami viszont minden járatnál más-más az egyes megállóknak.

```
typedef struct Ido {
    int ora, perc;
} Ido;
```

Az **Ido** adatszerkezet pedig azért lett létrehozva, mivel a programban nagyon sok helyen dolgozunk időpontokkal. Ezt hivattot megkönnyiteni, egyszerűsiteni.

A működését vezérlő fő függvények

```
Jarat_list *jarat_beolvas(Jarat_list *elso_jarat, FILE *fajl, Megallo_list *elso_megallo);
```

Beolvassa a megadott járatokat tartalmazó fájlt, lefoglalja a memóriát és létrehozza a járatok listáját. Első paramétere az eddigi járatok lista első eleme, illetve NULL ha még nincs listánk. Második paramétere a beolvasandó fájl. Harmadik paramétere a már beolvasott megállók listája. Fontos, hogy már be legyenek olvasnva azok a megállók, amiket a járatokban használni szeretnénk.

```
Megallo_list *megallo_beolvas(Megallo_list *elso_megallo, FILE *fajl);
```

Beolvassa a megadott megállókat tartalmazó fájlt, lefoglalja a memóriát és létrehozza a megállók listáját. Első paramétere az eddigi megállók lista első eleme, illetve NULL ha még nincs listánk. Második paramétere a beolvasandó fájl. A járatok beolvasása előtt kell használni.

A Jarat tipus lényeges függvényei és szerepük

A járatok listájáért felelős, illetve ennek a listának a későbbi szerkezstéséért, mint például további elemek hozzáadása, vagy elem keresése a listában.

```
void jarat_mentes(Jarat_list *elso_jarat, FILE *fajl);
```

Fájlba menti az eddig beolvasott járatokat.

```
void jarat_felszabadit(Jarat_list *elso_jarat);
```

Felszabaditja az eddigi járatoknak lefoglalt memóriát.

```
Jarat *jarat_keres(Jarat_list *elso_jarat, char *nev);
```

Megkeres a neve (string) alapján megalapján megadott járatot és visszaadja a mutatóját, illetve NULL-t, ha nincs ilyen járat.

A Megallo tipus lényeges függvényei és szerepük

A megállók listájáért felelős, illetve ennek a listának a későbbi szerkezstéséért, mint például további elemek hozzáadása, vagy elem keresése a listában. Ezenfelül itt vannak elmentve az útvonaltervezéshez szükséges adatok is.

```
void megallo_mentes(Megallo_list *elso_megallo, FILE *fajl);
```

Fájlba menti az eddig beolvasott megállókat.

```
void megallo_felszabadit(Megallo_list *elso_megallo);
```

Felszabaditja az eddigi megállóknak lefoglalt memóriát.

```
Megallo *megallo_keres(Megallo_list *elso_megallo, char *nev);
```

Megkeres a neve (string) alapján megalapján megadott megállót és visszaadja a mutatóját, illetve NULL-t, ha nincs ilyen megálló.

Az Ido tipus lényeges függvényei és szerepük

Ez egy segéd tipus, amivel az időpontok tárolását és szerkesztését tesszük egyszerűbbé, jobban kezelhetővé.

```
Ido str_to_ido(char *str);
```

Egy megadott **string**-ből **Ido** tipust csinál. Ez a funkció többféleképpen is létre van hozva, **Ido**-ből **int**-et készit, illetve **int**-ből **Ido**-t.

```
Ido ido_osszead(Ido i1, Ido i2);
```

Két **Ido** tipust összead és visszaadja **Ido** tipusban az összegüket.

```
int ido_cmp(Ido i1, Ido i2);
```

Két **Ido** tipusó változót összehasonlit és időpont szerint, ha az első nagyon 1-et, ha egyenlők 0-t, illetve -1-et ad vissza.