Készítette: **Fényes Balázs**

A programom célja a Számítástudomány alapjai c. tárgyon tanult gráfalgoritmusok kipróbálása lesz. Bemenete egy fájlban tárolt gráf csúcsai (névvel ellátva), és az élei (azok súlya, miket köt össze, iránya). A futás során a felhasználónak ki kell választani, hogy milyen algoritmust akar lefuttatni (gráf bejárása, legrövidebb utak keresése, stb.). A kimenet pedig a választástól függően egy számérték, lista, fa, stb. lehet.

A programnak két bemeneti fájlja van:

csucsok.dat – a csúcsok neveit tárolja, soronként egyet

```
Budapest
Debrecen
Sopron
...
```

elek.dat – az élek adatait tárolja, soronként egyet (honnan, hova, él paramétere (pl. csúcsok távolsága))

```
0 1 100
1 2 80
0 2 50
...
```

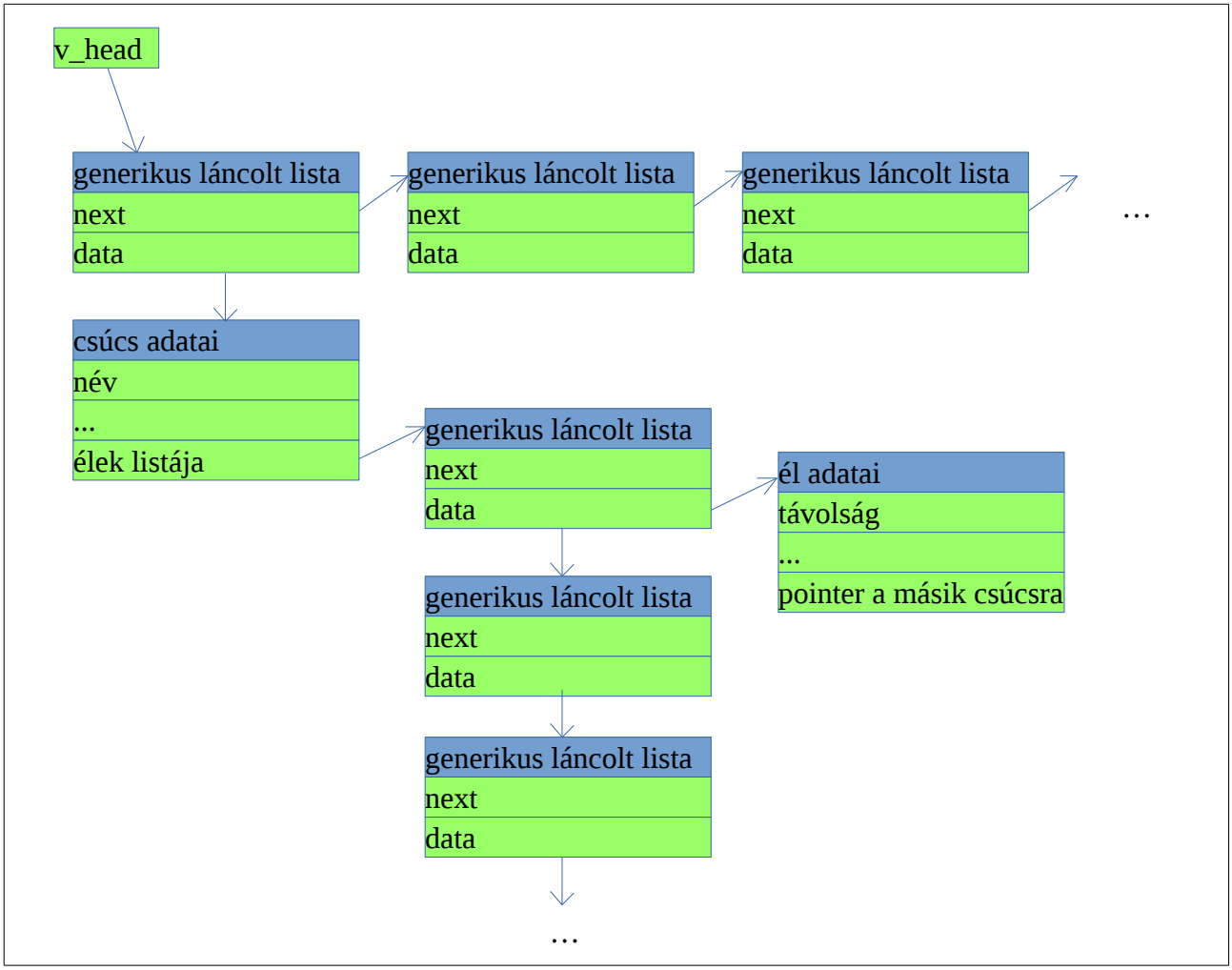
A csúcsok beolvasása során a csúcs adatait tároló struct-okból egy egyszeresen láncolt lista épül fel. Az élek beolvasásakor az él a kezdőpont saját éllistájához adódik hozzá, és egy hivatkozás tárol a végpontra. Így egy fésűs lista fog felépülni. Ha a gráf irányítatlan (ezt a felhasználó adja meg), akkor az él a kezdő- és a végpont éllistájához is hozzá lesz adva.

Az élek és a csúsok listájának használhatjuk ugyanazt az adatszerkezetet:

```
typedef struct generic_linkedlist{
	struct generic_linkedlist* next;
	void* data;
} generic_linkedlist;
```

A lista vége null pointerrel van jelölve.

A beolvasott adatokat tároló szerkezet részlete:



A fontosabb algoritmusok leírása:

A BFS algoritmus pszeudokódja a Wikipédiáról, a programban ezt fogom implementálni:

Input: A graph *Graph* and a *starting vertex root* of *Graph*

Output: All vertices reachable from *root* labeled as explored.

A non-recursive implementation of breadth-first search:

```
Breadth-First-Search(Graph, root):

    for each node n in Graph:
        n.distance = INFINITY
        n.parent = NIL

    create empty queue Q

    root.distance = 0
    Q.enqueue(root)

    while Q is not empty:
        current = Q.dequeue()
        for each node n that is adjacent to current:
            if n.distance == INFINITY:
                n.distance = current.distance + 1
```

```
                n.parent = current
                Q.enqueue(n)
```

Tehát szükséges még egy queue (FIFO) adatszerkezet is. Ez valójában egy láncolt lista, aminek csak a végére tudunk beszúrni (enqueue()) és csak az elejéről tudunk törölni (dequeue()).

```
typedef struct {
        generic_linkedlist* head;
        generic_linkedlist* tail;
} generic_queue;

void generic_queue_enqueue(generic_queue* gq, void* d);
void* generic_queue_dequeue(generic_queue* gq);
```

A bejárás során egy fát is tudunk építeni. Mivel nem tudjuk előre, hogy a fa egy csúcsából hány él fog indulni, ezért ezt is láncolt listában kell tárolni.

```
typedef struct generic_tree{
        void* data;
        struct generic_tree* children;
        struct generic_tree* siblings;
} generic_tree;

void generic_tree_insert(generic_tree* gt, generic_tree* leaf);
```

A DFS-algoritmus megvalósítása egyszerűbb (Wikipédiáról):

Input: A graph *G* and a vertex *v* of G

Output: All vertices reachable from *v* labeled as discovered

A recursive implementation of DFS:

```
1  procedure DFS(G,v):
2      label v as discovered
3      for all edges from v to w in G.adjacentEdges(v) do
4          if vertex w is not labeled as discovered then
5              recursively call DFS(G,w)
```

A BFS/DFS használható arra is, hogy ellenőrizzük, hogy a gráf összefüggő-e (minden csúcsból minden másikba el lehet jutni). A kapott fát rekurzívan bejárjuk, és ha a fa csúcsszáma megegyezik a gráf csúcsszámával, akkor a gráf összefüggő.

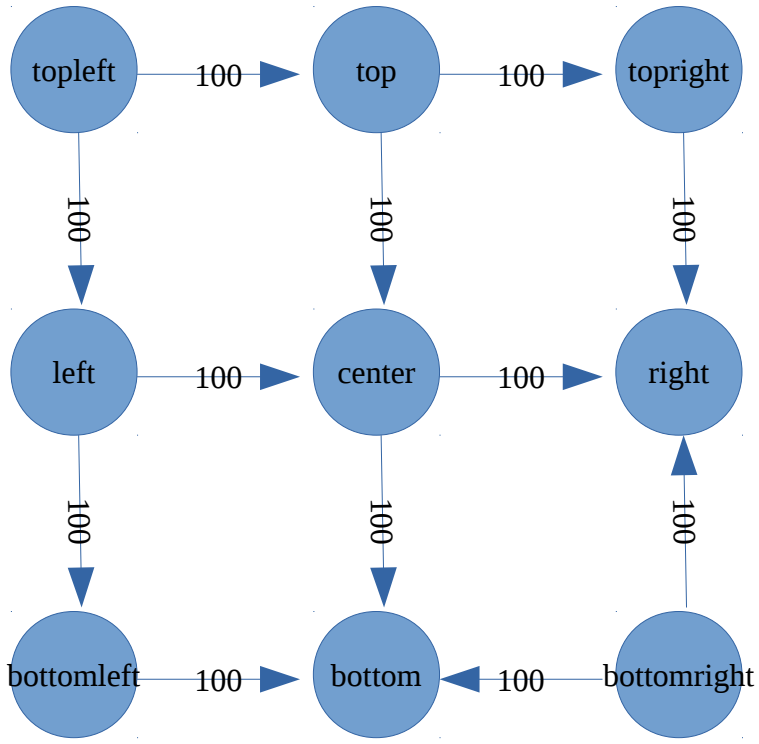
A program dokumentációját és a megvalósított gráfalgoritmusok leírását doxygen segítségével generáltam és az megtalálható a doc/html/index.html fájlban.

A program indításához a bemeneteket az parancssori argumenetumokként kell megadni.

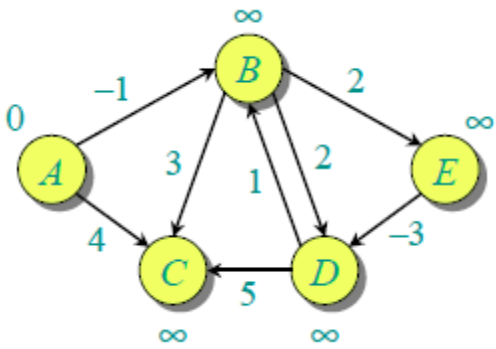
graf.out -v csucsok.dat -e elek.dat -d 0

ahol:
-v csúcsokat tartalmazó fájl
-e éleket tartalmazó fájl
-d [0|1] irányított-e a gráf

A dat mappában két teszteset is rendelkezésre áll.



bin/graf.out -v dat/csucsok.dat -e elek.dat -d 1



(forrás: http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/)

bin/graf.out -v dat/bf_csucsok.dat -e dat/bf_elek.dat -d 1

Futtatáshoz használható a run.sh szkript, ami elkészíti a dokumentációt, lefordítja a programot és a valgrind segítségével memóriaszivárgást keres futás közben. Windows-on a szkript pl. cygwin-nel futtatható.

```
#!/bin/bash
cd src
doxygen ../doc/Doxyfile > /dev/null
gcc -g graf.c generic_tree.c generic_queue.c generic_linkedlist.c io.c algorithms.c -o ../bin/graf.out
cd ..
valgrind --leak-check=full bin/graf.out -v dat/csucsok.dat -e dat/elek.dat -d 1
```

Példa a program futására:

```
bal@bal-SATELLITE-L50-B:~/Desktop/progc$ sh run.sh
/home/bal/Desktop/progc/src/generic_linkedlist.c:51: warning: Member _listsize(void *dataptr)
(function) of file generic_linkedlist.c is not documented.
/home/bal/Desktop/progc/src/generic_tree.c:44: warning: Member _gtprint(generic_tree *gt,
void(*printfunc)(void *), int depth) (function) of file generic_tree.c is not documented.
/home/bal/Desktop/progc/src/graf.c:18: warning: Member vertexnameprint(void *dataptr)
(function) of file graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:23: warning: Member vertexprint(void *dataptr) (function) of
file graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:33: warning: Member elistfree(void *dataptr) (function) of file
graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:39: warning: Member vlistfree(void *dataptr) (function) of file
graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:47: warning: Member countedges(void *dataptr) (function) of
file graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:52: warning: Member pathprint(void *dataptr) (function) of
file graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:57: warning: Member pathlistfree(void *dataptr) (function) of
file graf.c is not documented.
/home/bal/Desktop/progc/src/graf.c:63: warning: Member main(int argc, char **argv) (function) of
file graf.c is not documented.
==5545== Memcheck, a memory error detector
==5545== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5545== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==5545== Command: bin/graf.out -v dat/csucsok.dat -e dat/elek.dat -d 1
==5545==

Válasszon a lehetőségek közül:
 1. Statisztikák a gráfról
 2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
 3. BFS
 4. DFS
 5. Dijkstra
 6. Ford
1

csúcsok száma: 9
élek száma: 12

Válasszon a lehetőségek közül:
```

1. Statisztikák a gráfról
2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
3. BFS
4. DFS
5. Dijkstra
6. Ford
2

Összefüggőség csak irányítatlan gráf esetén vizsgálható!

Válasszon a lehetőségek közül:
1. Statisztikák a gráfról
2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
3. BFS
4. DFS
5. Dijkstra
6. Ford
3

Válasszon gyökércsúcsot: topleft, top, topright, left, center, right, bottomleft, bottom, bottomright
top

top
 center
  bottom
  right
 topright

Válasszon a lehetőségek közül:
1. Statisztikák a gráfról
2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
3. BFS
4. DFS
5. Dijkstra
6. Ford
4

Válasszon gyökércsúcsot: topleft, top, topright, left, center, right, bottomleft, bottom, bottomright
top

top
 center
  bottom
  right
 topright

Válasszon a lehetőségek közül:
1. Statisztikák a gráfról
2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
3. BFS
4. DFS
5. Dijkstra

```
 6. Ford
5

Válasszon gyökércsúcsot: topleft, top, topright, left, center, right, bottomleft, bottom, bottomright
top

Válasszon célcsúcsot: topleft, top, topright, left, center, right, bottomleft, bottom, bottomright
bottom

top(0) --> center(100) --> bottom(200)

Válasszon a lehetőségek közül:
 1. Statisztikák a gráfról
 2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
 3. BFS
 4. DFS
 5. Dijkstra
 6. Ford
6

Válasszon gyökércsúcsot: topleft, top, topright, left, center, right, bottomleft, bottom, bottomright
top

topleft(INFINITY)

top(0)

topright(100) <-- top(0)

left(INFINITY)

center(100) <-- top(0)

right(200) <-- topright(100) <-- top(0)

bottomleft(INFINITY)

bottom(200) <-- center(100) <-- top(0)

bottomright(INFINITY)


Válasszon a lehetőségek közül:
 1. Statisztikák a gráfról
 2. Összefüggőség vizsgálata (irányítatlan gráf esetén)
 3. BFS
 4. DFS
 5. Dijkstra
 6. Ford

==5545==
==5545== HEAP SUMMARY:
```

```
==5545==    in use at exit: 552 bytes in 1 blocks
==5545==  total heap usage: 108 allocs, 107 frees, 22,564 bytes allocated
==5545==
==5545== LEAK SUMMARY:
==5545==    definitely lost: 0 bytes in 0 blocks
==5545==    indirectly lost: 0 bytes in 0 blocks
==5545==      possibly lost: 0 bytes in 0 blocks
==5545==    still reachable: 552 bytes in 1 blocks
==5545==         suppressed: 0 bytes in 0 blocks
==5545== Reachable blocks (those to which a pointer was found) are not shown.
==5545== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5545==
==5545== For counts of detected and suppressed errors, rerun with: -v
==5545== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```