ELEMI ALKALMAZÁSOK FEJLESZTÉSE II.

Dijkstra és Prim algoritmusa

Készítette: Gregorics Tibor

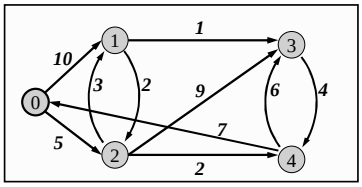
Tartalom

- Dijkstra legrövidebb utakat kereső algoritmusa
- Prim optimális feszítőfát kereső algoritmusa
- Számozott elsőbbségi sor
- Irányított illetve irányítatlan gráf szomszédsági listákkal

1. Feladat

Határozzuk meg egy nem-negatív értékekkel élsúlyozott irányított gráfban egy adott csúcsból kiinduló legolcsóbb utakat és azoknak költségét a gráf valamennyi csúcsára!

A gráf csúcsait 0-val kezdődően sorszámozzuk; a 0-dik az a kitűntetett csúcs, amelyből kivezető optimális utakat keressük. A gráfot szomszédsági listákkal adjuk meg egy szöveges fájlban. A fájl első sora a gráf csúcsainak számát, a további sorok az azon csúcsok szomszédsági listáját tartalmazzák (egy sor egy szomszédsági lista), amelyekből legalább egy él vezet ki. Minden szomszédsági lista szóközökkel elválasztott számokat tartalmaz: első szám annak a csúcsnak a sorszáma, amelyből kivezető éleket fel akarjuk itt sorolni; ezt követi a kivezető élek száma; majd minden kivezető élre páronként az él végcsúcsának sorszáma és a súlya.



input.txt

```
5
0 2 1 10 2 5
1 2 3  1 2 2
2 3 1  3 3 9 4 2
3 1 4  4
4 2 0  7 3 6
```

output

```
node cost optimal-path
  0   0   0
  1   8   0->2->1
  2   5   0->2
  3   9   0->2->1->3
  4   7   0->2->4
```

Specifikáció

$A = \boldsymbol{G} \times vekt(\boldsymbol{R}) \times vekt(\boldsymbol{N})$ $\quad \boldsymbol{G} = rec(V,E)$

$g \quad d \quad p$

$B = \boldsymbol{G}$

g'

$Q = (\, g = g' \wedge s = 0 \,)$

$R = (\, g = g' \wedge d[s] = 0 \wedge p[s] = nil \wedge$

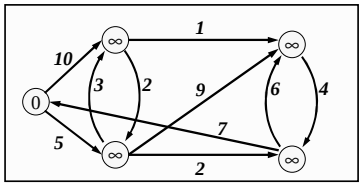
$\forall n \in g.V$:

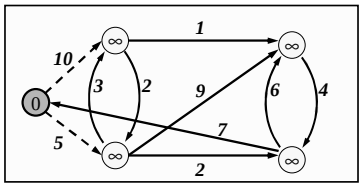
ha van g-ben $s \rightarrow n$ út akkor

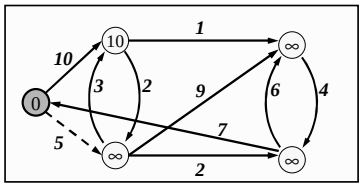
$d[n]$= a legolcsóbb g-beli $s \rightarrow n$ út költsége $\wedge$

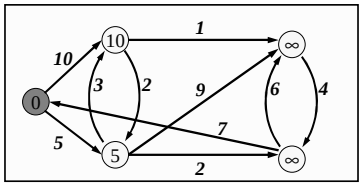
$p[n]$= n-nek egy legolcsóbb $s \rightarrow n$ úton fekvő szülője

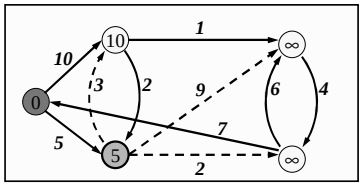
különben $d[n] = \infty \wedge p[n] = nil$)

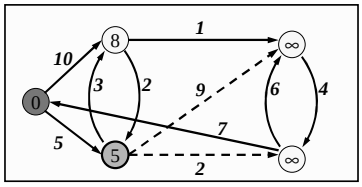


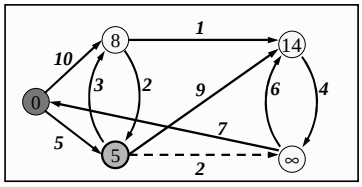

∞
∞
∞
∞
0
1
10
3
2
9
6
4
7
5
2

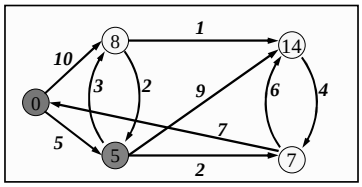

10
∞
∞
∞
0
1
10
3
2
9
6
4
7
5
2

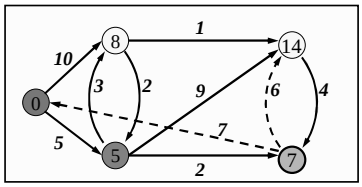

10
∞
∞
0
5
1
10
3
2
9
6
4
7
2

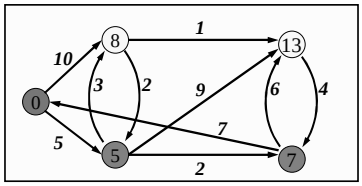

10
∞
∞
0
5
1
10
3
2
9
6
4
7
2

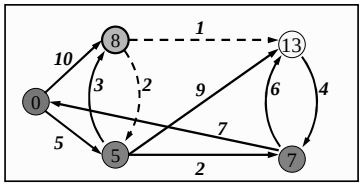

8
∞
∞
0
5
1
10
3
2
9
6
4
7
2

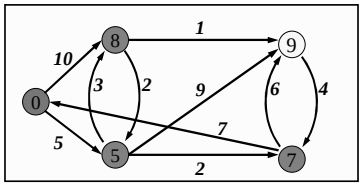

8
14
∞
0
5
1
10
3
2
9
6
4
7
2

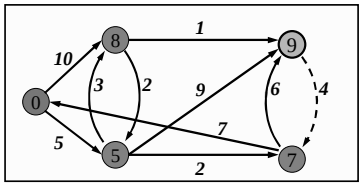

8
14
0
5
7
10
1
3
2
9
6
4
7
5
2

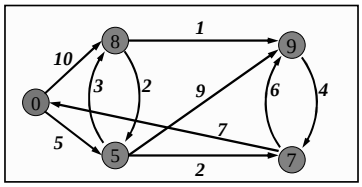

8
14
0
5
7
10
1
3
2
9
6
4
7
5
2


8
13
0
5
7
10
1
3
2
9
6
4
7
5
2


8
13
0
5
7
10
1
3
2
9
6
4
7
5
2


8
9
0
5
7
10
1
3
2
9
6
4
7
5
2


8
9
0
5
7
10
1
3
2
9
6
4
7
5
2



Absztrakt program

```
d[s]:=0; p[n]:=nil
Betesz(h,s,d[s])
for ∀n∈g.V\s-re do
  d[n]:=∞; p[n]:=nil
  Betesz(h,n,d[n])
enddo
while not ?Üres(h) do
  n:=Kivesz_Min(h)
  for ∀m∈Utodok(n)-re do
    if ?Eleme(h,m) and d[m]>d[n]+c(n,m) then
      d[m]:=d[n]+c(n,m)
      p[n]:=m
      Módosít(h,n,d[n])
    endif
  enddo
enddo
```

Irányított gráf

Számozott elsőbbségi sor

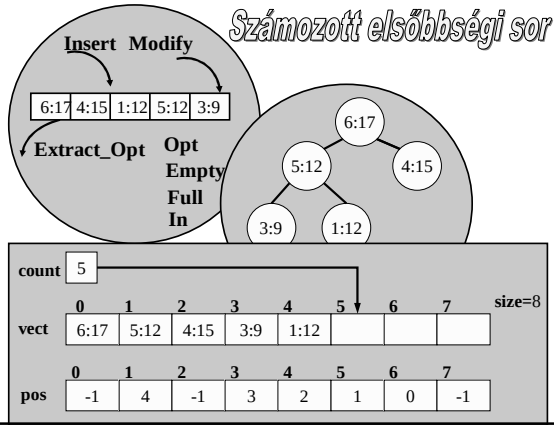
Számozott elsőbbségi sor

Készítsünk elsőbbségi sor osztálysablont olyan elemek tárolására, amelyeknek nemcsak értékük van, hanem egy azonosítójuk (kulcsuk) is. A sorban egy adott kulcsú elemből több nem lehet.

Módosítsuk az elsőbbségi sort arra az esetre, ha a kulcsokról feltehető, hogy azok a [*0..size-1*] intervallumba esnek.

Vezessünk be egy új műveletet: adott kulcsú elem értékének megváltoztatása

Számozott elsőbbségi sor



Kulcsos elemek típusa

```
template <class KeyType, class ValueType>
class Keyed_Element{
public:
  Keyed_Element(){};
  Keyed_Element(const KeyType& a,
                const ValueType & b)
  :key(a),value(b){};
  bool operator==(const Keyed_Element& r) const
                          {return key==r.key;}
  bool operator>(const Keyed_Element& r) const
                        {return value>r.value;}
  bool operator<(const Keyed_Element& r) const
                        {return value<r.value;}
  ValueType Value() const { return value;}
  KeyType Key()   const { return key;}
private:
  KeyType   key;
  ValueType value;
```

numbered_priority_queue.h

Az elsőbbségi sor elemi típusa

```
typedef
  Keyed_Element<Node,TypeReversed<float> >  Item;

...

Numbered_Priority_Queue<Item> h(nr);
```

shortest_path.cpp

Számozott elsőbbségi sor

```
template <class Element>
Numbered_Priority_Queue
          : public Unique_Priority_Queue<Element>{
public:
   enum Exceptions{EMPTY,FULL,EXISTING,NOTFOUND};
   Numbered_Priority_Queue(const int s);
   void Modify(const Element& e);
   ~Numbered_Priority_Queue()
             {delete[] pos; }
protected:
   int* pos;
   virtual int Search(const Element& e) const
                {return pos[e.Key()] ;}
   virtual Element Get(int i){
      pos[vect[i].Key()] = -1;
      return vect[i];}
   }
   virtual void Put(const Element& e, int i)
                {vect[i] = e; pos[e.Key()]=i; }
```

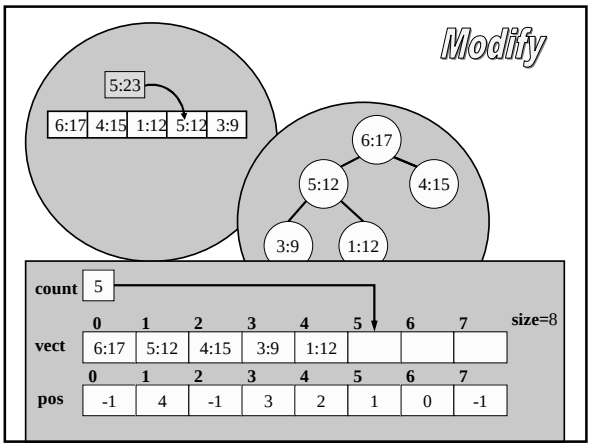
numbered_priority_queue.h

Konstruktor

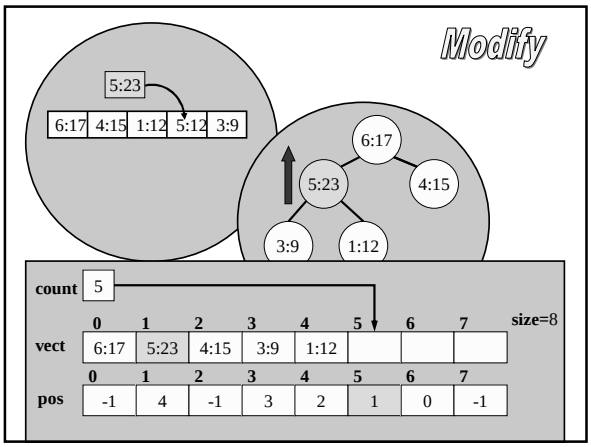
```
template <class Element>
Numbered_Priority_Queue<Element>::
Numbered_Priority_Queue(const int s)
       :Unique_Priority_Queue<Element>(s)
{
    pos = new int[size];
    for(int i = 0; i<size; i++) pos[i] = -1;
}
```

numbered_priority_queue.h

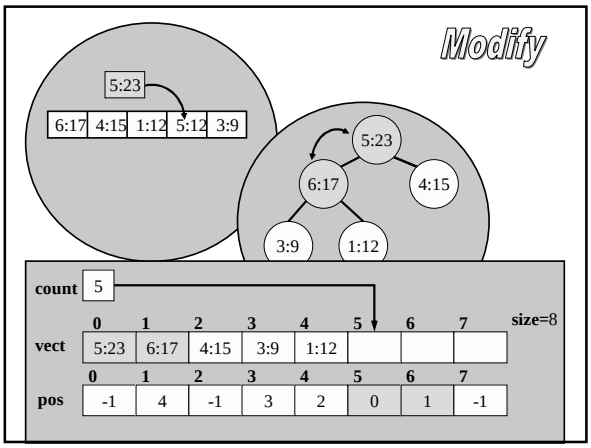
Modify



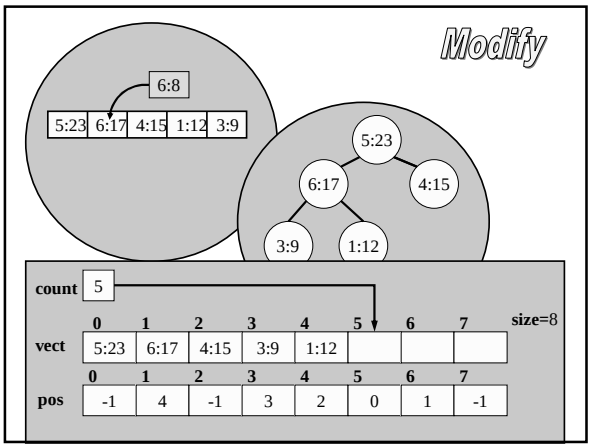
Modify



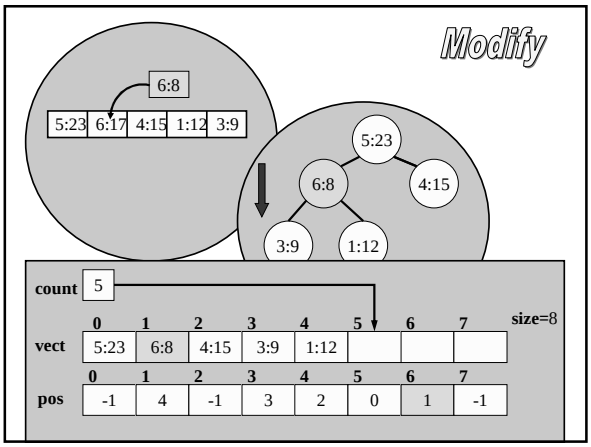
Modify



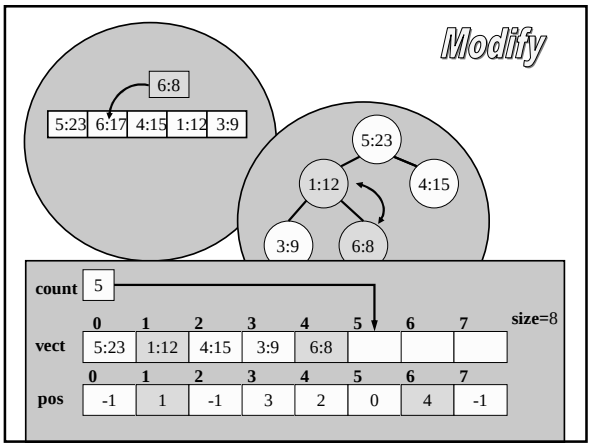
Modify



Modify



Modify



Modify

```
template <class Element>
void Numbered_Priority_Queue<Element>::
Modify(const Element& e)
{
   int i = Search(e);
   if(i<0) throw NOTFOUND;

   if( e>vect[i] ){
      Put(e,i);
      Up(i);
   }else{
      Put(e,i);
      Down(i);
   }
}
```

numbered_priority_queue.h

C++ program

```
Dir_Graph g("input.txt");
int nr = g.Nr();
Numbered_Priority_Queue<Item> h(nr);

float d[nr];
Node  p[nr];
float c;
Node  n,m;

d[0] = 0; p[n] = nil;
h.Insert(Item(0,d[0]));
for(n = 1; n<nr; n++){
    d[n] = infinite; p[n] = nil;
    h.Insert(Item(n,d[n]));
}
```

A gráfbeli csúcsok száma

shortest_path.cpp

C++ program

Egy csúcs gráfbeli utódainak bejárása

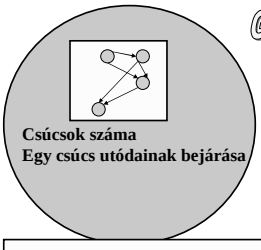
```
while( !h.Empty() ) {
    n = h.Extract_Opt().Key();
    Dir_Graph::Iterator it(g,n);
    for(it.Begin(); !it.End(); it.Next()) {
        m = it.Current_Node();
        c = it.Current_Cost();
        if(h.In(Item(m,c) && d[m]>d[n]+ c){
            d[m] = d[n]+c;
            p[m] = n;
            h.Modify(Item(d[m],m));
        }
    }
}
```

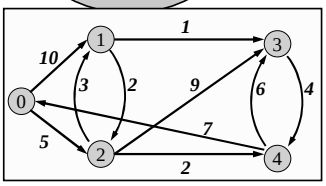
shortest_path.cpp

Tartalom

- Dijkstra legrövidebb utakat kereső algoritmusa
- Prim optimális feszítőfát kereső algoritmusa
- Számozott elsőbbségi sorral
- Irányított illetve irányítatlan gráf szomszédsági listákkal

Gráftípus specifikációja és reprezentációja





size 5

	a			
n_0	2,-	1,10	2,5	
n_1	2,-	3,1	2,2	
n_2	3,-	1,3	3,9	4,2
n_3	1,-	4,4		
n_4	2,-	0,7	3,6	

Gráftípus absztrakt osztálya

```
typedef int Node;

class Graph {
public:
   virtual ~Graph();
   int Nr() const {return size;}
protected:
   struct Pair{
      Node  n;
      float c;
   };
   int size;
   Pair** a;

   Graph(){};
   Graph(const Graph& g);
   Graph& operator=(const Graph& g);
}
```

graph.h

Destruktor

```
virtual Graph::~Graph()
{
   for(Node n = 0; n<size; n++){
     delete[] a[n];
   }
   delete[] a;
}
```

graph.h

Gráftípus absztrakt osztálya

```
friend class Iterator;
class Iterator {
public:
   Iterator(Graph& g, Node& n):  gr(&g), nd(n){};
   void Begin(){current = 1;}
   void Next() {current++;}
   bool End() const
      {return current>(gr->a[nd][0].n);}
   Node Current_Node()const
      {return gr->a[nd][current].n;}
   double Current_Cost()const
      {return gr->a[nd][current].c;}
private:
   Node  nd;
   Graph *gr;
   int   current;
};
```

graph.h

Irányított gráftípus osztálya

```
class Dir_Graph : public Graph {
public:
   Dir_Graph(const string name);
};
```

A származtatott típushoz olyan konstruktort tervezünk, amely egy fájlból olvassa be a dinamikus mátrixba a szomszédsági listákat.

dirgraph.h

Konstruktor

```
Dir_Graph::Dir_Graph(const string name)
{
   ifstream f(name.c_str());
   if(f.fail()){
      cout<<"Az inputfájl nem létezik!"<<endl;
      char ch; cin>>ch; exit(1);
   }
   Node n;
   f>>size;
   a = new Pair* [size];
   for(n = 0; n<size; n++) {
      a[n] = new Pair [1];
      a[n][0].n=0;
   }
```

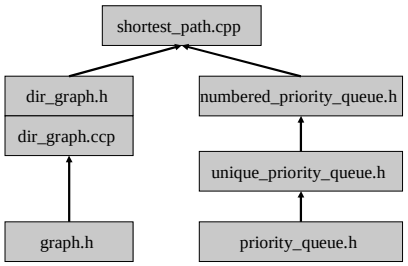
dirgraph.cpp

Konstruktor

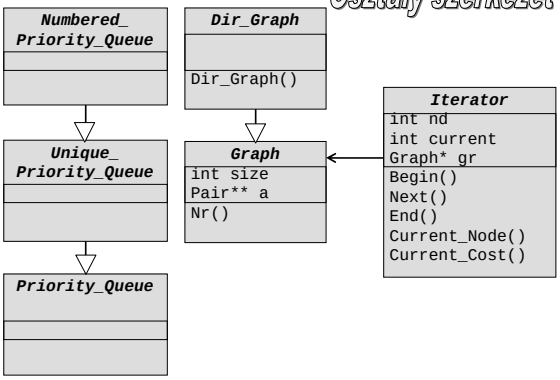
```
   int nr_succ;
   while (f>>n) {
      f>>nr_succ;
      delete[] a[n];
      a[n] = new Pair [nr_succ+1];
      a[n][0].n = nr_succ;
      for(int k = 1; k<=nr_succ; k++){
         f>>a[n][k].n>>a[n][k].c;
      }
   }
}
```

dirgraph.cpp

Modul szerkezet



Osztály szerkezet



Főprogram

```
#include "dir_graph.h"
#include "priority_queue.h"
#include <iostream>
#include <iomanip>
using namespace std;

typedef
   Keyed_Element<Node,TypeReversed<float> >  Item;

const int   nil      = -1;
const float infinite = 1000000.0;

int main()
{
   // Dijkstra's shortest paths algorithm
     ...
   // Results
     ...
   char ch; cin>>ch; return 0;
```

shortest_path.cpp

Főprogram

```
   // Results

   cout<< "node" << " cost" << " optimal path"
   << endl;
   for (n = 0; n<nr; n++) {
      cout << setw(3) << n << setw(5) << d[n];
      Node s[nr];
      int i = 0;
      m = n;
      while ( m!=nil ) {
         s[i++] = m;
         m = p[m];
      }
      cout << setw(4) <<
      for(i = i-2; i>=0; i--) {
         cout << "->" << s[i];
      }
      cout << endl;
```

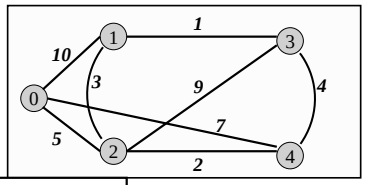
node	cost	optimal-path
0	0	0
1	8	0->2->1
2	5	0->2
3	9	0->2->1->3
4	7	0->2->4

shortest_path.cpp

2. Feladat

Határozzunk meg egy nem-negatív értékekkel élsúlyozott irányítatlan gráfban egy minimális élsúlyú feszítőfát!

A gráf csúcsait 0-val kezdődően sorszámozzuk.
A gráfot éleinek felsorolásával adjuk meg egy szöveges fájlban.
A fájl első sora a gráf csúcsainak számát, majd soronként az élek következnek. Minden sor három számot tartalmaz szóközökkel elválasztva: első két szám az él végcsúcsainak sorszáma, a harmadik szám az él súlya.



input.txt

```
5
0 1 10
0 2 5
0 4 7
1 3 1
1 2 3
2 3 9
2 4 2
3 4 4
```

output

edge		cost
2	0	5
4	2	2
1	2	3
3	1	1
	total:	11

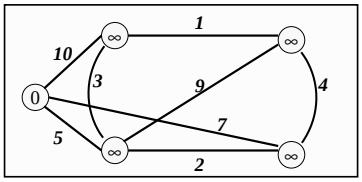
Specifikáció

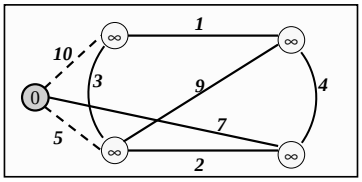
$A = \boldsymbol{G} \times E^{*}_{g} \qquad \boldsymbol{G} = rec(V,E)$

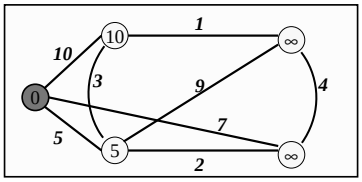
$B = \boldsymbol{G}_{g'}$

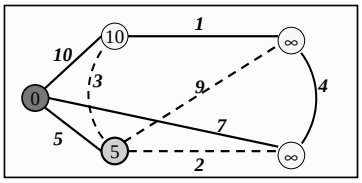
$Q = (g = g')$

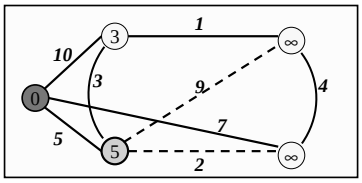
$R = (g = g' \wedge e = MFF(g))$

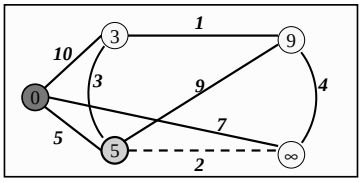


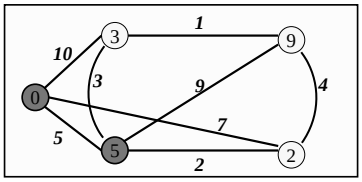


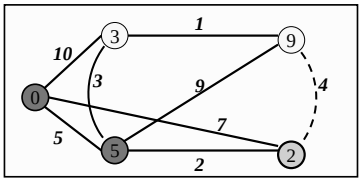


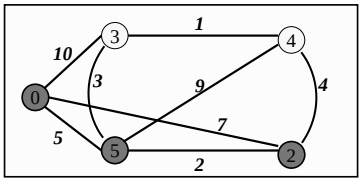

10
∞
∞
0
5
1
10
3
9
4
7
5
2

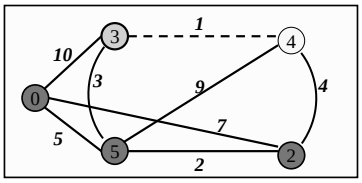

3
∞
∞
0
5
1
10
3
9
4
7
5
2

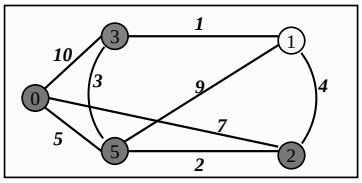

3
9
∞
0
5
1
10
3
9
4
7
5
2

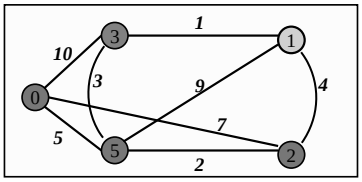

3
9
2
0
5
1
10
3
9
4
7
5
2

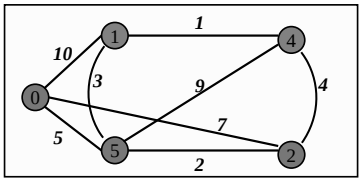

3
9
2
0
5
1
10
3
9
4
7
5
2


3
4
2
0
5
1
10
3
9
4
7
5
2









Absztrakt program

```
d[s]:=0;p[s]:=nil
Betesz(h,n,d[n])
for ∀n∈g.V\s-re do
  d[n]:=∞; p[n]:=nil
  Betesz(h,n,d[n])
enddo
while not ?Üres(h) do
  n:=Kivesz_Min(h)
  for ∀m∈Utódok(n)-re do
    if ?Eleme(h,m) and d[m]>c(n,m) then
      d[m]:=c(n,m)
      p[n]:=nil
      Módosít(h,n,d[n])
    endif
  enddo
enddo
```

Irányított gráf

Számozott elsőbbségi sor

C++ program

```
Undir_Graph g("input.txt");
int nr = g.Nr();
Numbered_Priority_Queue<Item> h(nr);

float d[nr];
Node  p[nr];
Node  n,m;
float c;

d[0] = 0; p[0] = nil;
h.Insert(Item(0,d[0]));
for(n = 1; n<nr; n++){
  d[n] = infinite; p[n] = nil;
  h.Insert(Item(n,d[n]));
}
```

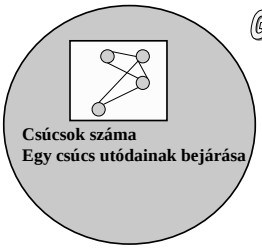
spanning_tree.cpp

C++ program

```
while ( !h.Empty() ) {
   n = h.Extract_Opt().Key();
   Undir_Graph::Iterator it(g,n);
   for(it.Begin(); !it.End(); it.Next()) {
      m = it.Current_Node();
      c = it.Current_Cost();
      if ( h.In(Item(m,c)) && c<d[m] ) {
         d[m] = c; p[m] = n;
         h.Modify(Item(m,d[m]));
      }
   }
}
```

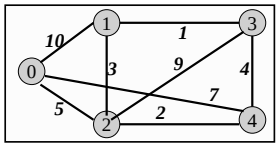
spanning_tree.cpp

Gráftípus specifikációja és reprezentációja



Csúcsok száma
Egy csúcs utódainak bejárása

size 5



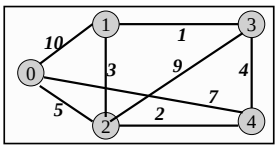
n_0	3,-	1,10	2,5	4,7	
n_1	3,-	0,10	2,3	3,1	
n_2	4,-	0,5	1,3	3,9	4,2
n_3	3,-	1,1	2,9	4,2	
n_4	3,-	0,7	2,2	3,4	

Irányítatlan gráf osztály

```
class Undir_Graph : public Graph {
public:
   Undir_Graph(const string name);
};
```

A származtatott típushoz olyan konstruktort tervezünk, amely egy fájlból olvassa be a dinamikus mátrixba a szomszédsági listákat.

undir_graph.h



size 5

b

	n_0	n_1	n_2	n_3	n_4
n_0	∞	10	5	∞	7
n_1	10	∞	3	1	∞
n_2	5	3	∞	9	2
n_3	∞	1	9	∞	4
n_4	7	∞	2	4	∞

a

n_0	3,-	1,10	2,5	4,7	
n_1	3,-	0,10	2,3	3,1	
n_2	4,-	0,5	1,3	3,9	4,2
n_3	3,-	1,1	2,9	4,2	
n_4	3,-	0,7	2,2	3,4	

Konstruktor

```
const float infinite=1000000.0;
Undir_Graph::Undir_Graph(const string name)
{
   ifstream f(name.c_str());
   if (f.fail()) {
      cout << "Az inputfájl nem létezik!" << endl;
      char ch; cin>>ch; exit(1);
   }
   float **b;
   Node n,m;
   f>>size;
   b = new float* [size];
   for(n = 0; n<size; n++){
      b[n] = new float[size];
      for(m = 0; m<size; m++)b[n][m] = infinite;
   }
   float c;
   while(f>>n>>m>>c) b[n][m]=b[m][n]=c;
```

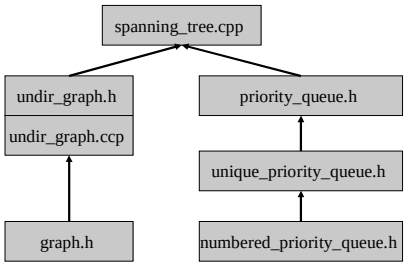
undir_graph.cpp

Konstruktor

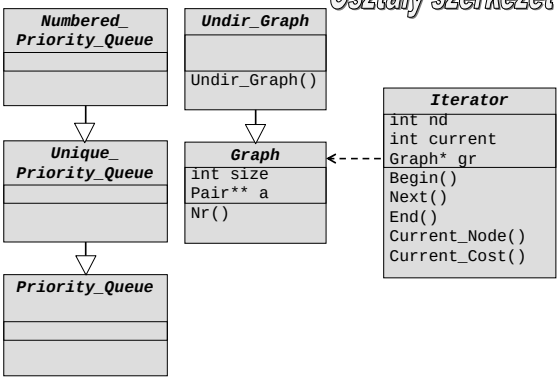
```
a = new Pair* [size];
for(n = 0; n<size; n++) {
   int s;
   for(m = 0, s = 0; m<size; m++){
      if(b[n][m]!=infinite) s++;
   }
   a[n] = new Pair [s+1];
   a[n][0].n = s;
   int k = 1;
   for(m = 0; m<=size; m++) {
      while(b[n][m]==infinite) m++;
      a[n][k].n = m;
      a[n][k].c = b[n][m];
      k++;
   }
}
for(n = 0; n<size; n++) delete[] b[n];
delete[] b;
```

undir_graph.cpp

Modul szerkezet



Osztály szerkezet



Főprogram

```
#include "dir_graph.h"
#include "priority_queue.h"
#include <iostream>
#include <iomanip>
using namespace std;

typedef
   Keyed_Element<Node,TypeReversed<float> >  Item;
const Node   nil      = -1;
const float infinite = 1000000.0;

int main()
{
   // Prim's minimal spanning tree algorithm
     ...
   // Results
     ...
   char ch; cin>>ch; return 0;
}
```

spanning_tree.cpp

Főprogram

```
// Results

float s = 0;
cout << "node" << " node" << " cost" << endl;
for (n = 1; n<nr; n++) {
   s+=d[n];
   cout << setw(3) << n << setw(5) << p[n]
        << setw(5) << d[n] <<endl;
}
cout << setw(12) << "total: " << s << endl;
```

```
  edges     cost
  2   0       5
  4   2       2
  1   2       3
  3   1       1
      total: 11
```

spanning_tree.cpp