LEMON

ELTE-TTK, Operációkutatási Tanszék

Jüttner Alpár
alpar@cs.elte.hu

ELTE-TTK, Operációkutatási tanszék

1. Letöltés
2. Fordítás - telepítés
3. Használat

Számítógépes eszközök

http://lemon.cs.elte.hu

- Nyílt forráskódú C++ library gráfokkal (hálózatokkal) kapcsolatos optimalizálási feladatok megoldására
- Cél: Olyan eszköztár létrehozása, amely egyszerre alkalmas kutatási feladatokban *és* ipari felhasználásra.
  - Nyílt forráskód, kereskedelmi felhasználást is biztosító licenc
  - Az eszközök elérhető leghatékonyabb implementációjára törekszünk (a leggyorsabb a piacon)
- Megbízható, kiterjedt implementáció
  - $\approx$ 55000 erősen optimalizált kódsor
  - Windows, Unix támogatás különféle fordítókkal
- Fejlesztő kerestetik!!!

Letöltés

Bináris:

Csomagkezelőből (openSUSE)

- “Contrib” repository-nak része
- 1-Click Install

Forráskód:

Stabil verzió (release tarball)

- `http://lemon.cs.elte.hu` ⟶ Downloads
- `wget http://lemon.cs.elte.hu/pub/sources/lemon-1.2.tar.gz`

Fejlesztői verzió

- `hg clone http://lemon.cs.elte.hu/hg/lemon-main`
- `hg clone http://lemon.cs.elte.hu/hg/lemon`
- Böngészőben: `http://lemon.cs.elte.hu/hg/lemon` vagy `http://lemon.cs.elte.hu/hg/lemon-main`

Installálás (forrásból)

- Windows vagy bármi más (Un*x - Linux, MacOs, AIX Solaris etc.)?
- Release vagy fejlesztői verzió?
- Mindenkinek vagy csak magunknak?
- Fordítóprogram (GNU C++, Intel C++, Visual Studio, MinGW, AIX xCl)?
- Fejlesztői környezet ((N)Make, Visual Studio, CodeBlocks, Eclipse)?

Installálás (Linux+CMAKE)

Kell hozzá

- CMAKE, [CMAKE GUI] (`http://cmake.org`)

Parancssor

```
tar xf lemon-1.2.tar.gz
cd lemon-1.2
mkdir build; cd build
cmake ..
make
[make check]
make install
```

CMAKE GUI

- `cmake ..` ⟶ `cmake-gui ..`
- Ki kell választani a “target”-et
  - Unix Makefile, CodeBlocks project, Eclipse project

Installálás (Linux+CMAKE)

Kell hozzá

- CMAKE, [CMAKE GUI] (`http://cmake.org`)

Parancssor

```
hg clone http://lemon.cs.elte.hu/hg/lemon-main mylemon
cd mylemon
mkdir build; cd build
cmake ..
make
[make check]
make install
```

CMAKE GUI

- `cmake ..` ⟶ `cmake-gui ..`
- Ki kell választani a “target”-et
  - Unix Makefile, CodeBlocks project, Eclipse project

Installálás (Windows+CMAKE)

Kell hozzá

- CMAKE (`http://cmake.org`)
- C++ fordító
  - Visual Studio 2008 (Express Edition is jó)
  - CodeBlocks
  - MinGW

Parancssor

- A fordító parancsértelmezőjét kell megnyitni!!!

```
hg clone http://lemon.cs.elte.hu/hg/lemon-main mylemon
cd mylemon
mkdir build; cd build
cmake-gui ..
```

- Ki kell választani a “target”-et
  - NMake Makefile, CodeBlocks project, VS2008 project
- Fordítás
  - `nmake+nmake install`
  - A “solution”/”project” fájlt megnyitni a megfelelő IDE-ből és “Build”

Első programunk

hello_lemon.cc

```
#include <iostream>
#include <lemon/list_graph.h>

using namespace lemon;
using namespace std;

int main()
{
  ListDigraph g;

  ListDigraph::Node u = g.addNode();
  ListDigraph::Node v = g.addNode();
  ListDigraph::Arc  a = g.addArc(u, v);

  cout << "Hello World! This is LEMON library here." << endl;
  cout << "We have a directed graph with " << countNodes(g) << " nodes "
  << "and " << countArcs(g) << " arc." << endl;

  return 0;
}
```

Alapok

Header file-ok

`#include<lemon/list_graph.h>`	*A* gráf adatstruktúra
`#include<lemon/bfs.h>`	BFS algoritmus
`#include<lemon/dijkstra.h>`	Dijkstra algoritmus

Namespace

```
using namespace lemon;
```

Gráf felépítése

Csúcsok, élek hozzádása

```
ListDigraph::Node x = g.addNode();
ListDigraph::Node y = g.addNode();
ListDigraph::Node z = g.addNode();

g.addArc(x,y); g.addArc(y,z); g.addArc(z,x);
```

Beolvasás fájlból

```
#include <lemon/list_graph.h>
#include <lemon/lgf_reader.h>

using namespace lemon;

int main()
{
  ListDigraph g;
  digraphReader(g, "digraph.lgf").run();
}
```

LGF formátum

digraph.h

```
@nodes
label
0
1
...
41

@arcs
@
0     1
0     2
2     12
...
36    41
```

LGF formátum

digraph.h

```
@nodes
label
0
1
...
41

@arcs
            label
0     1     0
0     2     1
2     12    2
...
36    41    123
```

LGF formátum

digraph.h

```
@nodes
label size
0     12
1     3
...
41    12

@arcs
          label length
0    1    0     16
0    2    1     12
2    12   2     20
...
36   41   123   21
```

ListGraph

Csúcsok, élek hozzáadása és törlése

```
ListDigraph g;
Node n; n = g.addNode();
Arc e; e = g.addArc(n1,n2);
g.erase(n); g.erase(e);
```

Párhuzamos és hurokélek

```
ListDigraph::Arc parallel = g.addArc(x,y);
ListDigraph::Arc loop = g.addArc(x,x);
```

Élek eleje és vége

```
if (g.source(loop) == g.target(loop))
  std::cout << "This is a loop arc" << std::endl;
```

Él és csúcs ID

- `g.id(n)` és `g.id(a)`
- Egyedi, permanens azonosító
- Az ID nem túl nagy nemnegatív egész szám
  - Nem okvetlen alkotnak folytonos intervallumot

Iterátorok

- Gráf elemeinek felsorolására való
- Különbözik az STL iterátoroktól!
- Az iterátor konstruktora a kezdőértékre állít
- Prefix `++`-szal lépkedünk az elemeken
- Ha túlmentünk az utolsó elemen, akkor egyenlő lesz az `INVALID` konstanssal

Példa: Gráf pontjainak megszámolása

```
int cnt = 0;
for (ListDigraph::NodeIt n(g); n != INVALID; ++n)
  cnt++;
std::cout << "Number of nodes: " << cnt << std::endl;
```

Példa II.: Teljes gráf hozzáadása

```
for (ListDigraph::NodeIt u(g); u != INVALID; ++u)
  for (ListDigraph::NodeIt v(g); v != INVALID; ++v)
    if (u != v) g.addArc(u, v);
```

Iterátorok II.

`ArcIt`: Gráf éleinek felsorolása

```
int cnt = 0;
for (ListDigraph::ArcIt a(g); a != INVALID; ++a)
  cnt++;
std::cout << "Number of arcs: " << cnt << std::endl;
```

`OutArcIt`: Egy csúcsból kimenő élek felsorolása

```
int cnt = 0;
for (ListDigraph::OutArcIt a(g, n); a != INVALID; ++a)
  cnt++;
std::cout << "Num of arcs leaving the node 'n': "
          << cnt << std::endl;
```

`InArcIt`: Egy csúcsba bejövő élek felsorolása

```
int cnt = 0;
for (ListDigraph::NodeIt n(g); n != INVALID; ++n)
  for (ListDigraph::InArcIt a(g, n); a != INVALID; ++a)
    cnt++;
std::cout << "Number of arcs: " << cnt << std::endl;
```

Iterátorok III.: STL sequences

STL kompatibilis iterátorok

```
int cnt = 0;
for (ListDigraph::Arc a : g.arcs())
  cnt++;
std::cout << "Number of arcs: " << cnt << std::endl;
```

```
int cnt = 0;
for (auto a : g.outArcs(n))
  cnt++;
std::cout << "Num of arcs leaving the node 'n': "
          << cnt << std::endl;
```

- `g.nodes(), g.arcs()`
- `g.inArcs(n), g.outArcs(n)`

Iterátorok (egyebek)

A felsorolás sorrendje

- A csúcsok és élek felsorolásának sorrendje bármi lehet
- Nem okvetlen egyezik meg a hozzáadásuk sorrendjével
- Ami garantált: Ha a gráfot nem változtatjuk, akkor az egymás után következő felsorolások ugyanazt a sorrendet adják.

Segédeszközök

- `countNodes(g)`
- `countArcs(g)`
- `countInArcs(g,n)`
- `countOutArcs(g,n)`

A Node és Arc típusok a '<' operátorral rendezhetők

```
for (ListDigraph::NodeIt u(g); u != INVALID; ++u)
  for (ListDigraph::NodeIt v(g); v != INVALID; ++v)
    if (u < v) g.addArc(u, v);
```

Adatok hozzárendelése csúcsokhoz és élekhez

Mapek jellemzői

- Gyors/hatékony
  - mint a tömbök
- Dinamikus
  - bármikor allokálhatunk új mapeket
- Automatikus
  - Új élek/csúcsok hozzáadásakor a megfelelő mapekben is foglalódnak tárhelyek és inicializálódnak
  - Elemek törlésekor a hozzárendelt attributumok destruktorai meghívódnak

Használat (mint egy vektor)

```
ListDigraph::NodeMap<int> map(g);

map[x] = 2;
map[y] = 3;
map[z] = map[x] + map[y];
```

Mapek II.

Mindenféle adattípus használható

```
ListDigraph::NodeMap<std::string> name(g);
name[x] = "Node A";
name[y] = "Node B";
```

Példa

```
ListDigraph::NodeMap<char> label(g);
char ch = 'A';
for (ListDigraph::NodeIt n(g); n != INVALID; ++n)
  label[n] = ch++;
```

Példa: Kezdőérték

```
ListDigraph::NodeMap<int> out_deg(g, 0);
for (ListDigraph::ArcIt a(g); a != INVALID; ++a)
  out_deg[g.source(a)]++;
```

Vigyázat

A később foglalt csúcsokhoz tartozó értékek már nem fognak a megadott értékre inicializálódni!

Példa

Szélességi keresés

```
#include<lemon/list_graph.h>
#include<queue>

bfs(ListDigraph &g, Node s)
{
  ListDigraph::NodeMap<ListDigraph::Arc> pred(g,INVALID);
  ListDigraph::NodeMap<bool> visited(g,false);
  std::queue<Node> queue;
  queue.push(s);
  visited[s]=true;
  while(!queue.empty()) {
    for(ListDigraph::OutArcIt a(g,queue.front()); a!=INVALID; ++a) {
      n=g.target(a);
      if(!visited[n]) {
        queue.push(n);
        visited[n]=true;
        pred[n]=a;
      }
    }
    queue.pop();
  }
}
```

A “gyári” BFS algoritmus

A `Bfs` osztály (`#include <lemon/bfs.h>`)

- Inicializálás

```
ListDigraph g;
Bfs<ListDigraph> bfs(g);
```

- Futtatás

```
bfs.run(s);
bfs.run(s,t);
```

- Eredmény kiolvasása

```
bfs.dist(n)
bfs.predNode(n)
bfs.predArc(n)
bfs.path(n)
...
```

A `bfs()` függvény (`#include <lemon/bfs.h>`)

```
ListDigraph::NodeMap<int> dist_map(g);
bfs(g).distMap(dist_map).run(s);
```

BFS példa

max_dist-nél közelebbi pontokba vezető utak

```
Bfs<ListDigraph> bfs(g);
bfs.run(s);

for (ListGraph::NodeIt n(g); n != INVALID; ++n) {
  if (bfs.reached(n) && bfs.dist(n) <= max_dist) {
    std::cout << gr.id(n);
    Node prev = bfs.prevNode(n);
    while (prev != INVALID) {
      std::cout << "<-" << gr.id(prev);
      prev = bfs.prevNode(n);
    }
    std::cout << std::endl;
  }
}
```

BFS/DFS futtatása

BFS futtatása

```
bfs.run(s);
```

BFS/DFS futtatása

BFS futtatása

```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
bfs.start();
```

BFS/DFS futtatása

BFS futtatása

```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
while (!bfs.emptyQueue()) {
  bfs.processNextNode();
  ...
}
```

BFS/DFS futtatása

BFS futtatása

```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
while (!bfs.emptyQueue()) {
  bfs.processNextNode();
  ...
}
```

DFS futtatása

```
bfs.run(s);
```

BFS/DFS futtatása

BFS futtatása

```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
while (!bfs.emptyQueue()) {
  bfs.processNextNode();
  ...
}
```

DFS futtatása

```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
bfs.start();
```

BFS/DFS futtatása

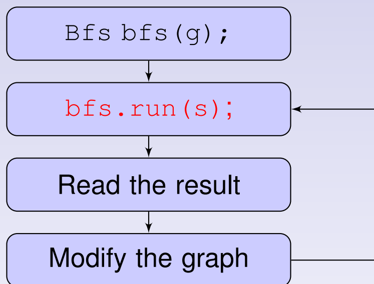
BFS futtatása

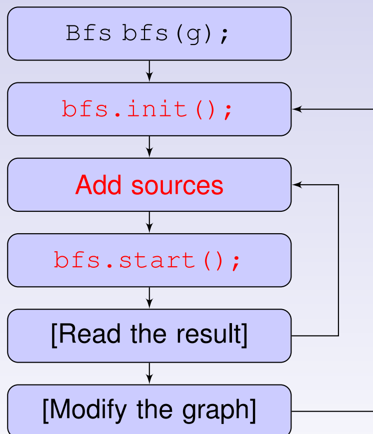
```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
while (!bfs.emptyQueue()) {
  bfs.processNextNode();
  ...
}
```

DFS futtatása

```
bfs.init();
bfs.addSource(s);
[bfs.addSource(s2);]
while (!bfs.emptyQueue()) {
  bfs.processNextArc();
  ...
}
```

BFS/DFS ismételt futtatása





Segédeszközök

Miből mennyi van?

- `countNodes()`, `countArcs()`, `countOutArcs()`, `countInArcs()`
- Ha a gráf nyilvántartja, hogy mennyi van az adott elemből, akkor ezek a függvények $O(1)$-ben futnak.

Gráfok másolása (nincs copy constructor!)

```
ListDigraph sg;
SmartDigraph tg;
digraphcopy(sg, tg).run();
```

Segédeszközök

Miből mennyi van?

- `countNodes()`, `countArcs()`, `countOutArcs()`, `countInArcs()`
- Ha a gráf nyilvántartja, hogy mennyi van az adott elemből, akkor ezek a függvények $O(1)$-ben futnak.

Gráfok másolása (nincs copy constructor!)

```
ListDigraph sg;
SmartDigraph tg;
ListDigraph::NodeMap<SmartDigraph::Node> nr(sg);
digraphcopy(sg, tg).nodeRef(nr).run();
```

Segédeszközök

Miből mennyi van?

- `countNodes()`, `countArcs()`, `countOutArcs()`, `countInArcs()`
- Ha a gráf nyilvántartja, hogy mennyi van az adott elemből, akkor ezek a függvények $O(1)$-ben futnak.

Gráfok másolása (nincs copy constructor!)

```
ListDigraph sg;
SmartDigraph tg;
ListDigraph::NodeMap<SmartDigraph::Node> nr(sg);
SmartDigraph::ArcMap<ListDigraph::Arc> acr(tg);
digraphcopy(sg, tg).nodeRef(nr).arcCrossRef(acr).run();
```

Élek két pont közöt

Sima keresés (végiglépdel a kimenő éleken)

```
for(ListDigraph::Arc e=findArc(g,u,v); e!=INVALID; e=findArc(g,u,v,e))
{...  }
for(ConArcIt<ListDigraph> a(g,u,v); a!=INVALID; ++a) { ...  }
```

Static Arc Lookup Table ($O(\log d)$ időben)

```
ArcLookUp<ListDigraph arcs(g);
ListDigraph::Arc a=arcs(u,v);
...  //Modify the graph
arcs.refresh();
```

Static Arc Lookup Table (minden él, $O(\log d)$ időben)

```
AllArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
...  //Modify the graph
arcs.refresh();
```

Dynamic Arc Lookup Table (automatikusan frissítődik, $O(\log d)$ időben)

```
DynArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
```

Élek két pont közöt

Sima keresés (végiglépdel a kimenő éleken)

```
for(ListDigraph::Arc e=findArc(g,u,v); e!=INVALID; e=findArc(g,u,v,e))
{...  }
for(ConArcIt<ListDigraph> a(g,u,v); a!=INVALID; ++a) { ...  }
```

Static Arc Lookup Table ($O(\log d)$ időben)

```
ArcLookUp<ListDigraph arcs(g);
ListDigraph::Arc a=arcs(u,v);
...  //Modify the graph
arcs.refresh();
```

Static Arc Lookup Table (minden él, $O(\log d)$ időben)

```
AllArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
...  //Modify the graph
arcs.refresh();
```

Dynamic Arc Lookup Table (automatikusan frissítődik, $O(\log d)$ időben)

```
DynArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
```

Élek két pont közöt

Sima keresés (végiglépdel a kimenő éleken)

```
for(ListDigraph::Arc e=findArc(g,u,v); e!=INVALID; e=findArc(g,u,v,e))
{...  }
for(ConArcIt<ListDigraph> a(g,u,v); a!=INVALID; ++a) { ...  }
```

Static Arc Lookup Table ($O(\log d)$ időben)

```
ArcLookUp<ListDigraph arcs(g);
ListDigraph::Arc a=arcs(u,v);
...  //Modify the graph
arcs.refresh();
```

Static Arc Lookup Table (minden él, $O(\log d)$ időben)

```
AllArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
...  //Modify the graph
arcs.refresh();
```

Dynamic Arc Lookup Table (automatikusan frissítődik, $O(\log d)$ időben)

```
DynArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
```

Élek két pont közöt

Sima keresés (végiglépdel a kimenő éleken)

```
for(ListDigraph::Arc e=findArc(g,u,v); e!=INVALID; e=findArc(g,u,v,e))
{...  }
for(ConArcIt<ListDigraph> a(g,u,v); a!=INVALID; ++a) { ...  }
```

Static Arc Lookup Table ($O(\log d)$ időben)

```
ArcLookUp<ListDigraph arcs(g);
ListDigraph::Arc a=arcs(u,v);
...  //Modify the graph
arcs.refresh();
```

Static Arc Lookup Table (minden él, $O(\log d)$ időben)

```
AllArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
...  //Modify the graph
arcs.refresh();
```

Dynamic Arc Lookup Table (automatikusan frissítődik, $O(\log d)$ időben)

```
DynArcLookUp<ListDigraph arcs(g);
for(ListDigraph::Arc a=arcs(u,v); a!=INVALID; a=arcs(u,v,a)) { ...  }
```

Irányítatlan gráfok

```
ListGraph g;
ListGraph::Edge e = g.addEdge(u,v);
```

Egyrészt: úgy működik, mint egy irányított gráf

Másrészt: Irányítatlan gráf-műveletek

Irányítatlan gráfok

```
ListGraph g;
ListGraph::Edge e = g.addEdge(u,v);
```

Egyrészt: úgy működik, mint egy irányított gráf

- Minden hozzáadott irányítatlan él oda-vissza behúzott irányított élnek látszik $\Longrightarrow$ Minden irányított gráfos algoritmus használható irányítatlan gráffal is
  - Más kérdés, hogy amit ad, az értelmes-e.

Másrészt: Irányítatlan gráf-műveletek

Irányítatlan gráfok

```
ListGraph g;
ListGraph::Edge e = g.addEdge(u,v);
```

Egyrészt: úgy működik, mint egy irányított gráf

Másrészt: Irányítatlan gráf-műveletek

- `ListGraph::Edge e = g.addEdge(u,v);`
  - A `ListGraph::Arc` konvertálódik erre
  - Visszafele konverzió: `a = direct(e, true)` és `a = direct(e, u)`
  - `oppositeArc(a)`: Az ellentétes irányítású él
- `ListGraph::EdgeMap<int> emap(g);`
- `ListGraph::IncEgdeIt it(g,u);`
- A két vége
  - Egyik-másik: `g.u(e)` és `g.v(e)`
  - A másik végpont: `oppositeNode(u, e)`
  - Iterátoroknál (`*ArcIt`-re és még irányított gráfokban is!)
    - Ami körül forog: `g.baseNode(e);` a másik: `g.runningNode(e)`

Map-ek III.: a ReadMap concept

ReadMap

```
struct MyMap {
  typedef ListDigraph::Arc Key;
  typedef double Value;
  Value operator[](Key &k) const { return PI; }
};
```

Vagy:

```
struct MyMap:  MapBase<ListDigraph::Arc,double> {
  Value operator[](Key &k) const { return PI; }
};
```

- Mindenhol, ahol csak olvasható map-et vár egy algoritmus, ott használhatunk saját map-típust
- Fordítási időben kapcsolódik össze a map és az algoritmus $\Longrightarrow$ hatékony
- Van sok “map adaptor” (l. később)

Readmap példa

Dijkstra futtatás redukált költségekkel

```
class ReducedLengthMap  : public MapBase<Digraph::Arc,double>
{
  const Digraph &g;
  const Digraph::ArcMap<double> &orig_len;
  const Digraph::NodeMap<double> &pot;

public:
  Value operator[](Key e) const {
    return orig_len[e]-(pot[g.target(e)]-pot[g.source(e)]);
  }

  ReducedLengthMap(const Digraph &_g,
                   const Digraph::ArcMap &_o,
                   const Digraph::NodeMap &_p)
    : g(_g), orig_len(_o), pot(_p) {};
};

...
ReducedLengthMap rm(g,len,pot);
Dijkstra<Digraph,ReducedLengthMap> dij(g,rm);
dij.run(s);
...
```

Map-ek IV.: a WriteMap és ReferenceMap conceptek

WriteMap

```
struct MyMap:  MapBase<ListDigraph::Arc,double> {
  Value operator[](Key &k) const { return PI; }
  void set(Key &k, Value v) { ...  }
};
```

RefenenceMap

```
struct MyMap:  MapBase<ListDigraph::Arc,double> {
  const Value &operator[](Key &k) const { ...  }
  Value &operator[](Key &k) { ...  }
  void set(Key &k, Value v) { ...  }
};
```

Map adaptorok

Példa

```
ListDigraph graph;

typedef ListDigraph::ArcMap<double> DoubleArcMap;
DoubleArcMap length(graph);
DoubleArcMap speed(graph);

typedef DivMap<DoubleArcMap, DoubleArcMap> TimeMap;
TimeMap time(length, speed);

Dijkstra<ListDigraph, TimeMap> dijkstra(graph, time);
dijkstra.run(source, target);
```

Vagy:

```
dijkstra(graph, divMap(length, speed).run(source,target));
```

- `AddMap<M1,M2>`, `SubMap<M1,M2>`, `MulMap<M1,M2>` stb.
- `ComposeMap<M1,M2>`, `CombineMap<M1,M2,F,V>`, `FunctorToMap<F,K,V>`, `CombineMap<M1,M2,F,V>`, `ConvertMap<M,V>` stb.
- `ForkMap<M1,M2>`, `NotWriteMap<M>`, `NegWriteMap<M>`

A Kruskal algoritmus (minimális feszítőfa)

```
kruskal(g,in,out);
```

`g`

Lehet irányított vagy irányítatlan gráf

`in`

- `EdgeMap<>` vagy `ArcMap<>`, ami megadja az élek súlyát
- Egy STL “Forward container” (pl. `std::vector<>` vagy `std::list<>`, aminek minden eleme egy `std::pair<GR::Arc/Edge,C>` és az elemek a költségek szerint növekvő sorrendben vannak.

`out`

- `EdgeMap<bool>` vagy `ArcMap<bool>`
- Egy Arc/Edge STL “container” iteratora.
  ```
  std::vector<Arc> tree(53);
  kruskal(g,cost,tree.begin());
  ```
  vagy, ha nem tudjuk előre az élek számát:
  ```
  std::vector<Arc> tree;
  kruskal(g,cost,std::back_inserter(tree));
  ```