Ugrólisták

Ugrólisták

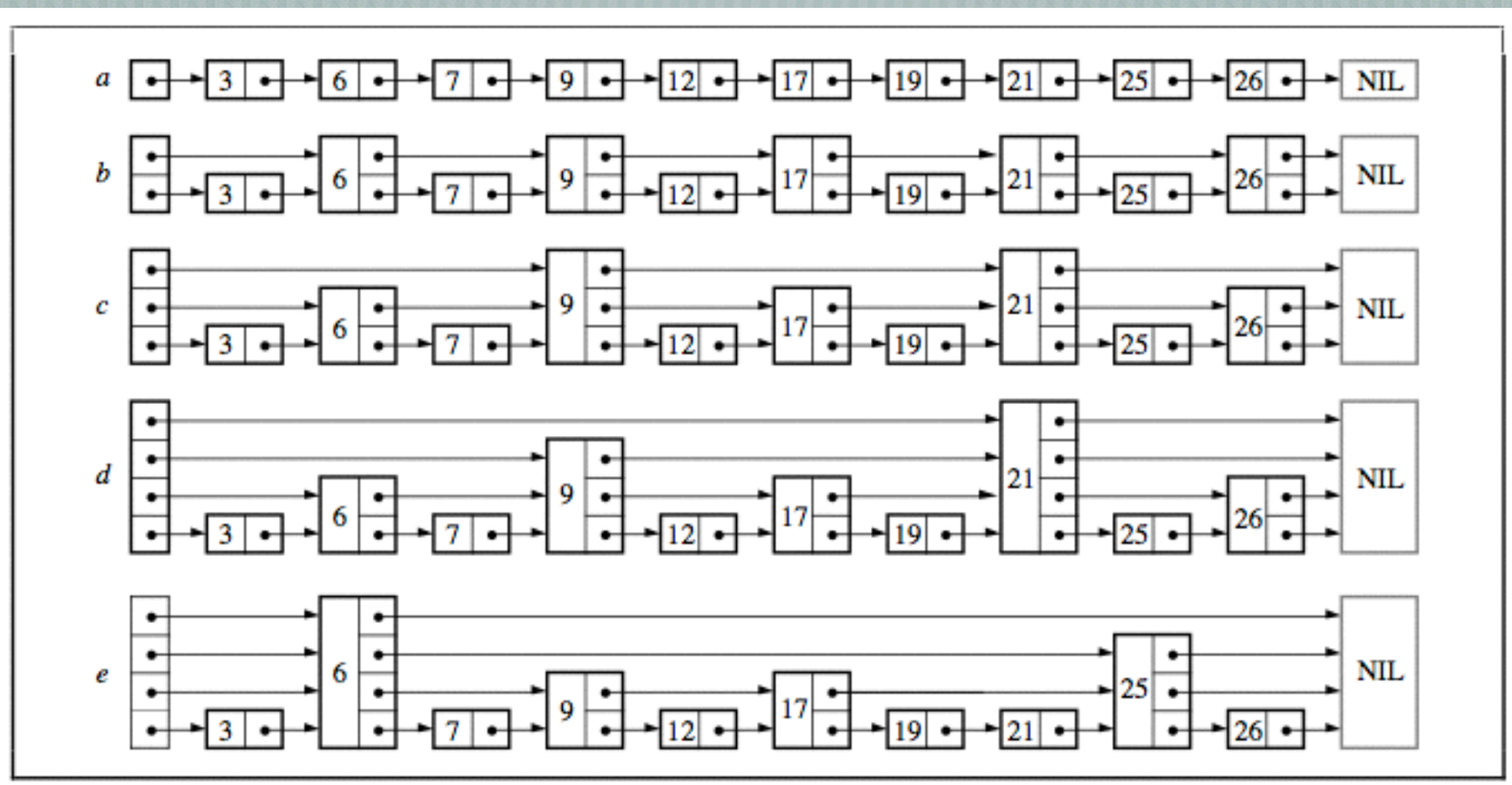


FIGURE 1 - Linked lists with additional pointers

Ugrólisták

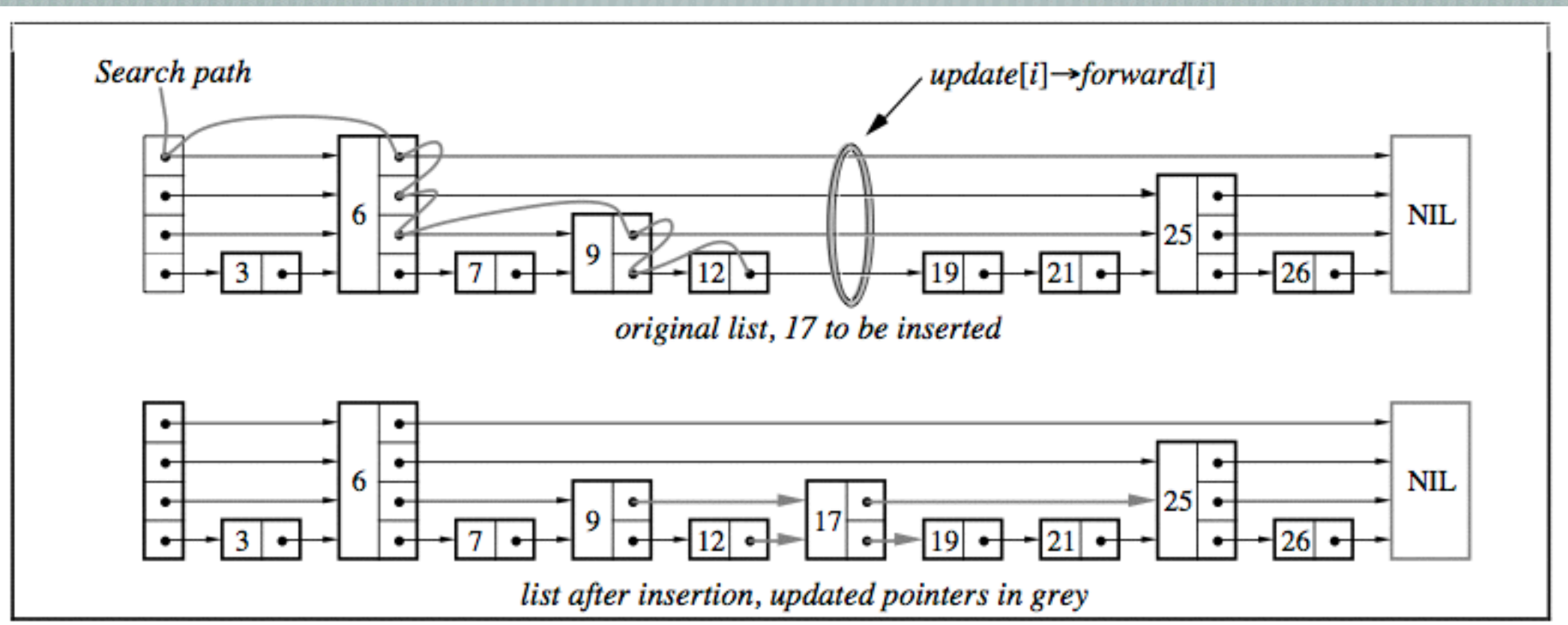
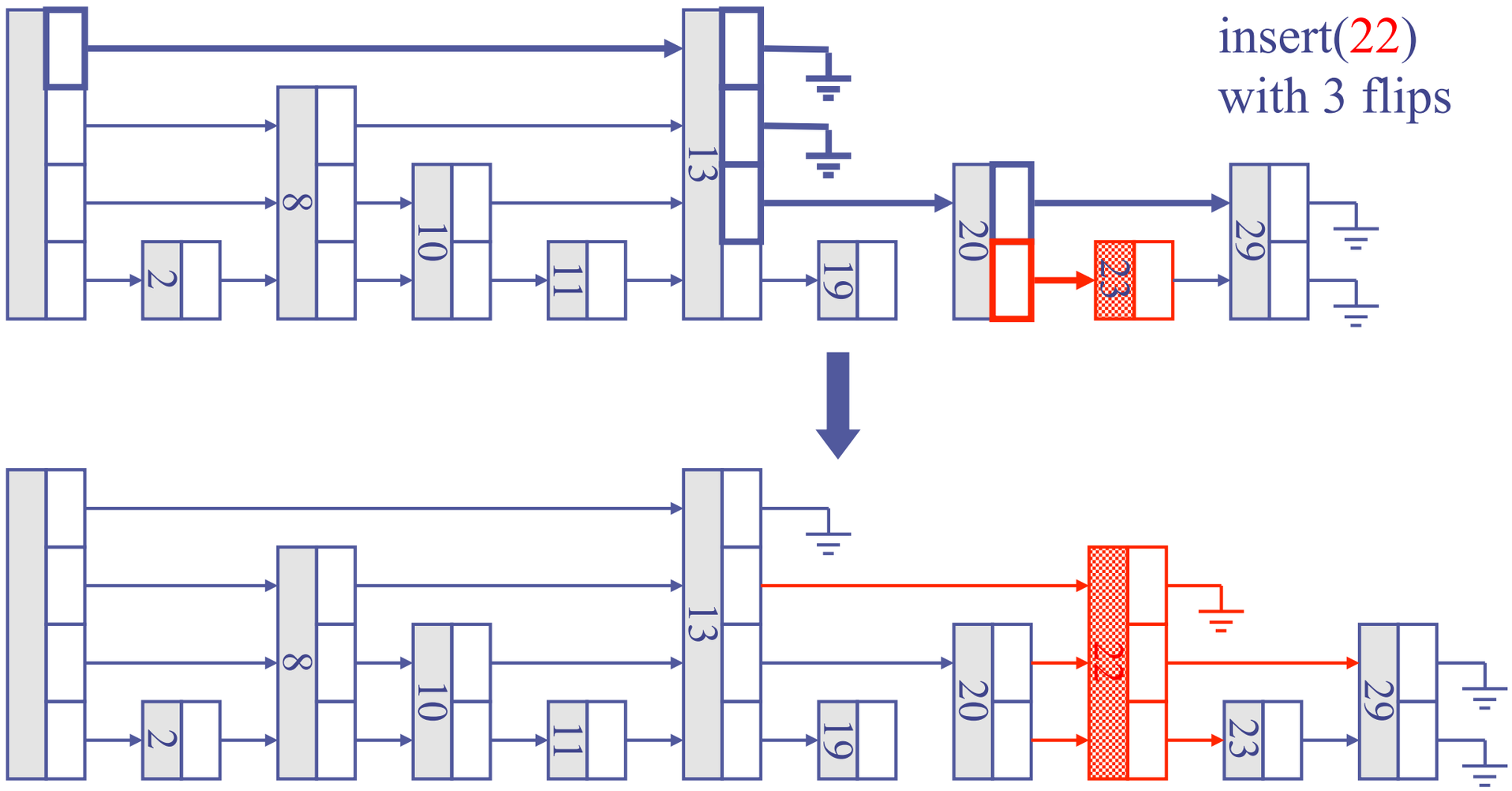


FIGURE 3 - Pictorial description of steps involved in performing an insertion

Ugrólisták

RSL Insert Example



Runtime?

Ugrólisták

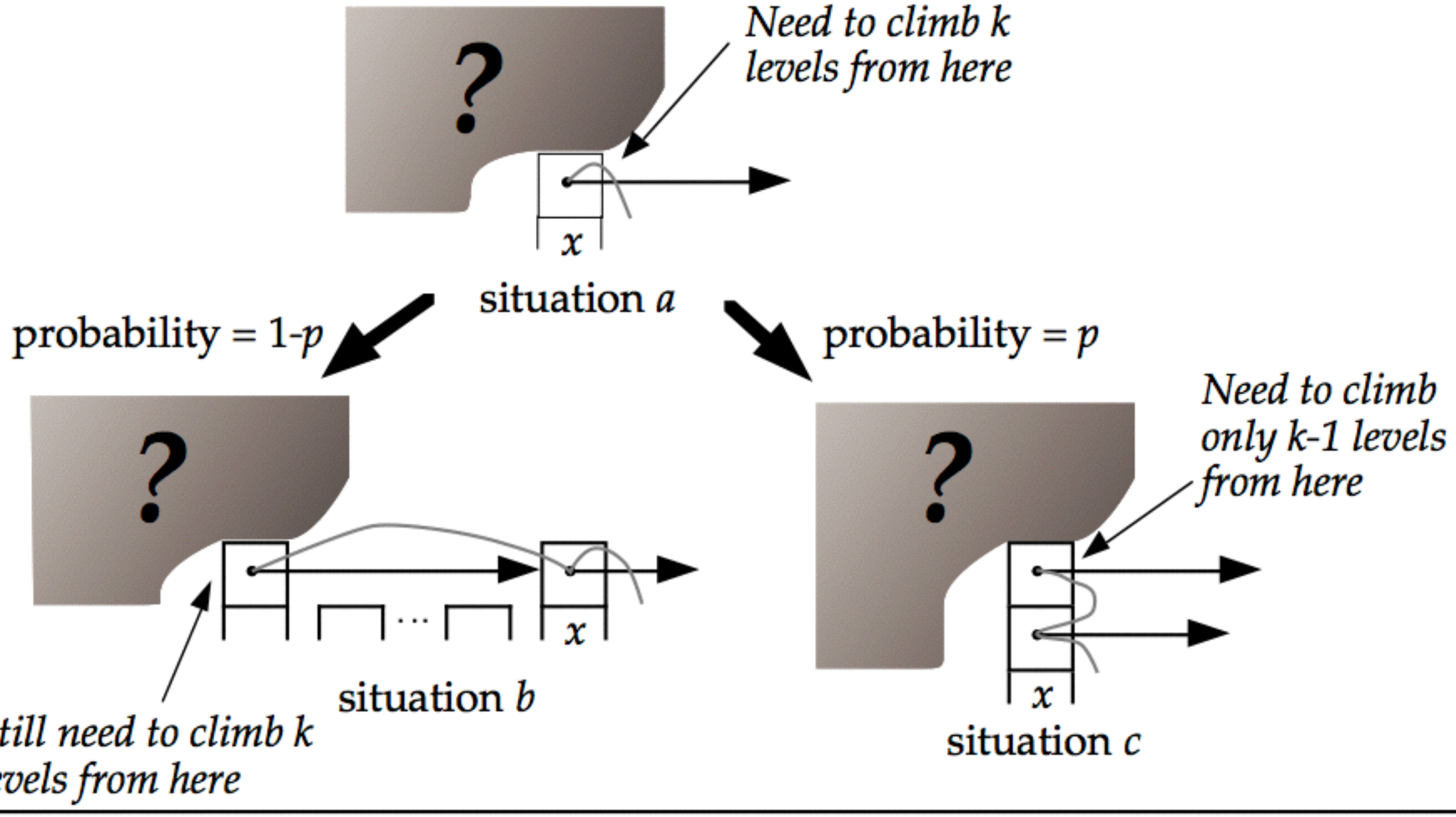


FIGURE 6 - Possible situations in backwards traversal of the search path

Empirical analysis

Implementation	Search Time	Insertion Time	Deletion Time
Skip lists	0.051 msec (1.0)	0.065 msec (1.0)	0.059 msec (1.0)
non-recursive AVL trees	0.046 msec (0.91)	0.10 msec (1.55)	0.085 msec (1.46)
recursive 2–3 trees	0.054 msec (1.05)	0.21 msec (3.2)	0.21 msec (3.65)
Self–adjusting trees:			
top-down splaying	0.15 msec (3.0)	0.16 msec (2.5)	0.18 msec (3.1)
bottom-up splaying	0.49 msec (9.6)	0.51 msec (7.8)	0.53 msec (9.0)

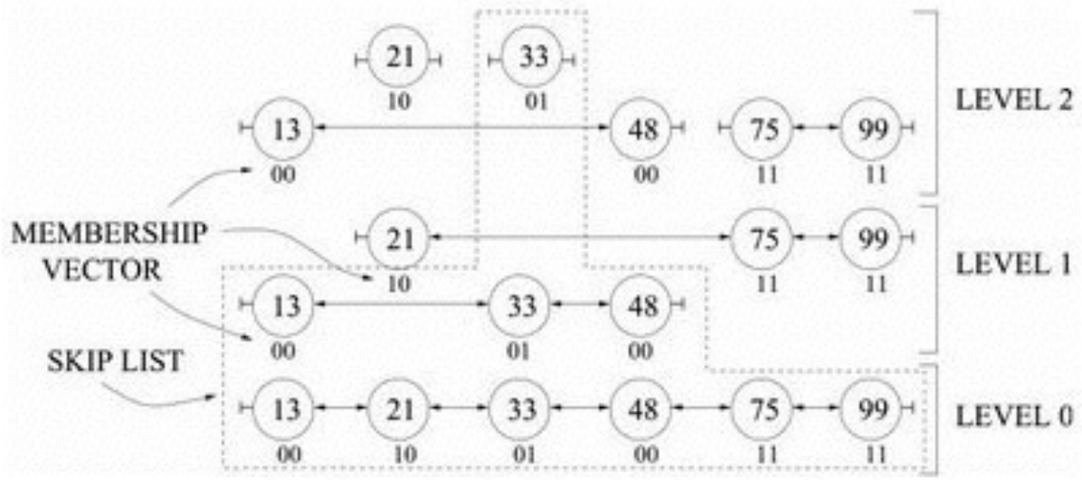
Table 2 - Timings of implementations of different algorithms

CONCLUSIONS

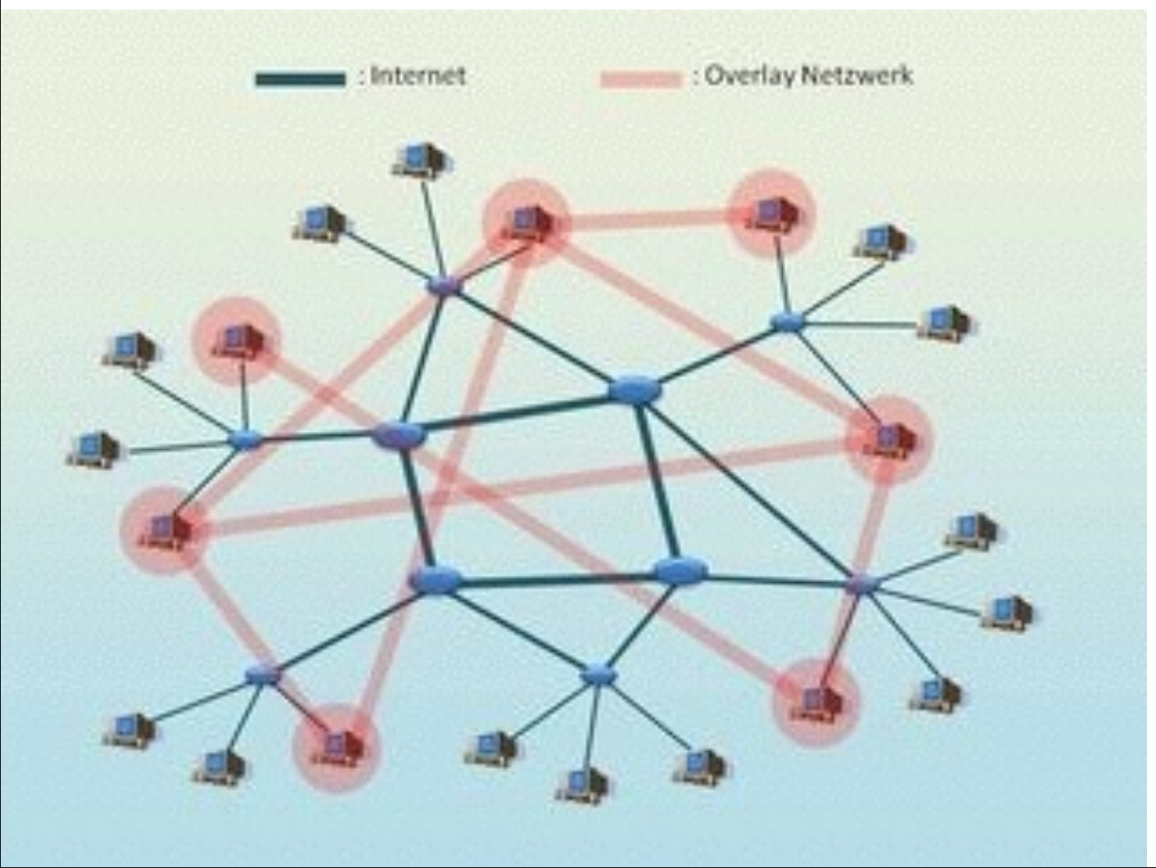
From a theoretical point of view, there is no need for skip lists. Balanced trees can do everything that can be done with skip lists and have good worst-case time bounds (unlike skip lists). However, implementing balanced trees is an exacting task and as a result balanced tree algorithms are rarely implemented except as part of a programming assignment in a data structures class.

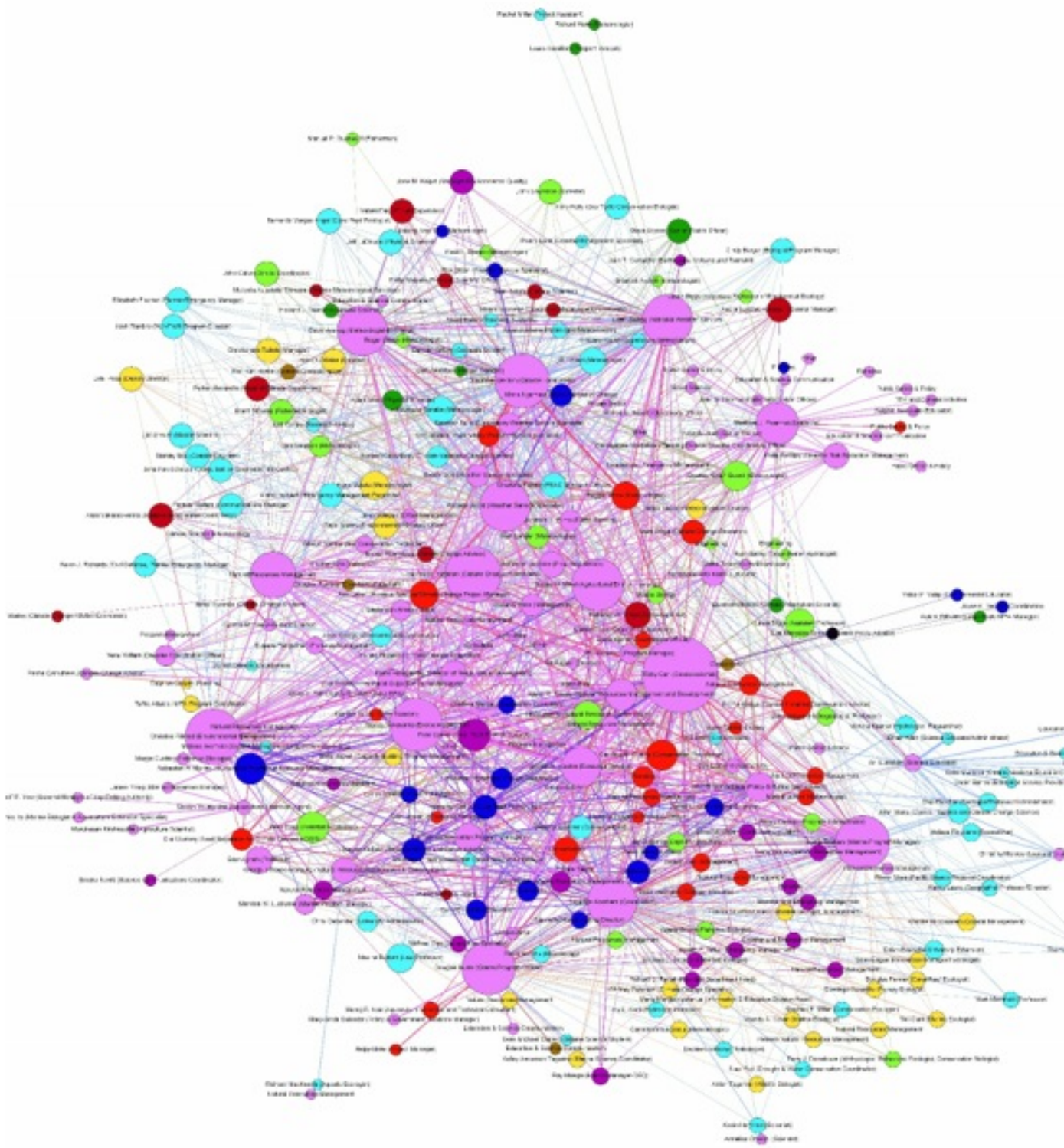
Skip lists are a simple data structure that can be used in place of balanced trees for most applications. Skip lists algorithms are very easy to implement, extend and modify. Skip lists are about as fast as highly optimized balanced tree algorithms and are substantially faster than casually implemented balanced tree algorithms.

http://www.inf.u-szeged.hu/~tnemeth/alga2/eloadasok/skiplists.pdf



A skip graph with $n = 6$ nodes and $\lceil \log n \rceil = 3$ levels.





Adatszerkezetek tervezése

Halmazok egyesítése

motiváció

Kruskal algoritmus

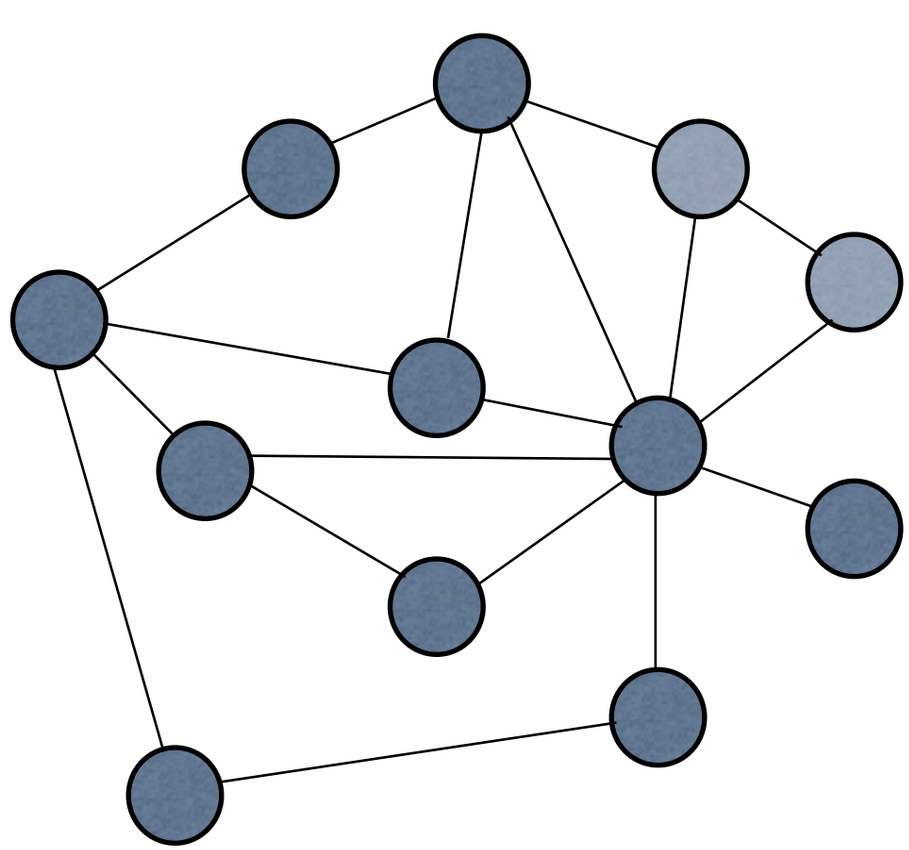
felelevenítés

```
kezdetben a gráf minden pontja külön halmazban van ;
vegyük a gráfból az éleket növekvő sorrendben {
   ha az él 2 végpontja külön halmazban van {
      az él legyen része a minimális feszítőfának ;
      a 2 végpontot tartalmazó halmazt egyesítsük ;
   }
}
```

Halmazok egyesítése

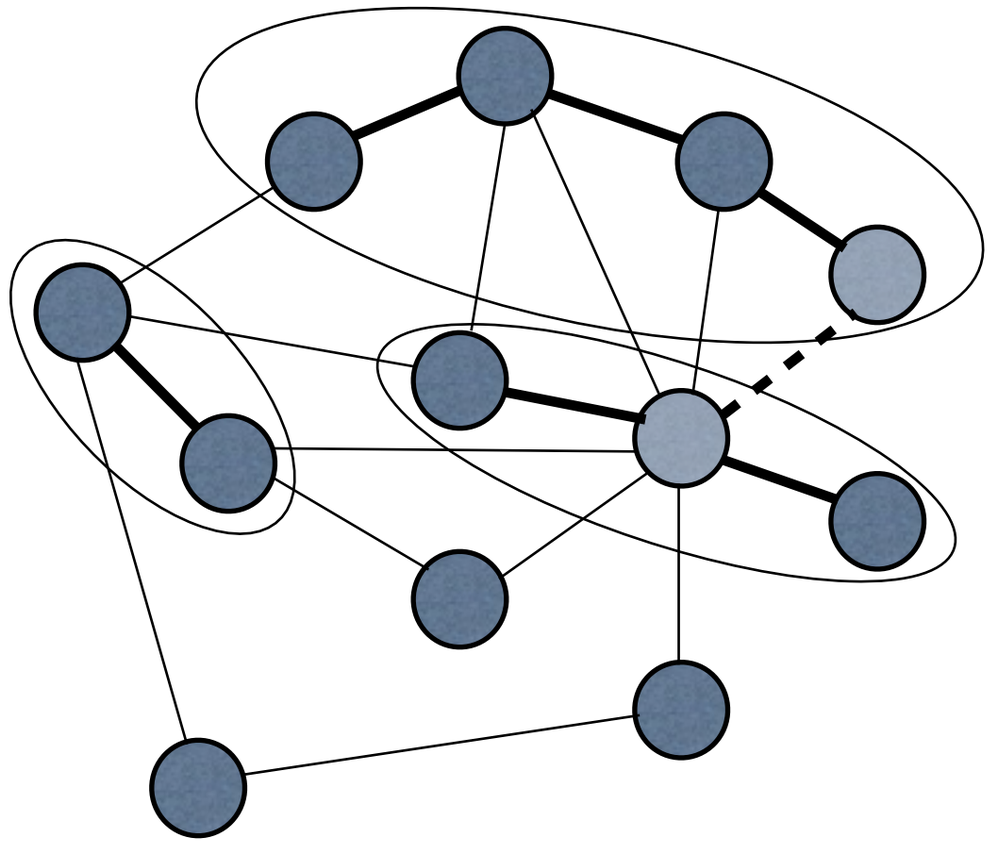
Kruskal algoritmus

felelevenítés



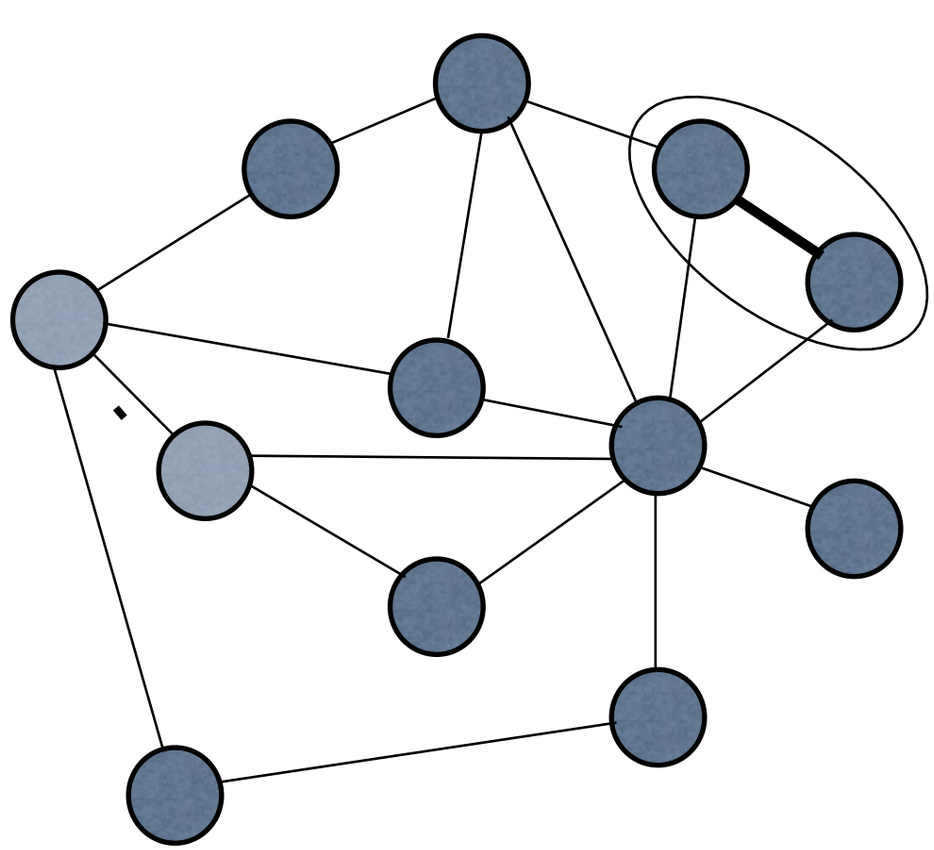
Honnan lehet tudni, hogy a következő él beválasztásával a feszítőerdő erdő (azaz körmentes) marad?

Példa



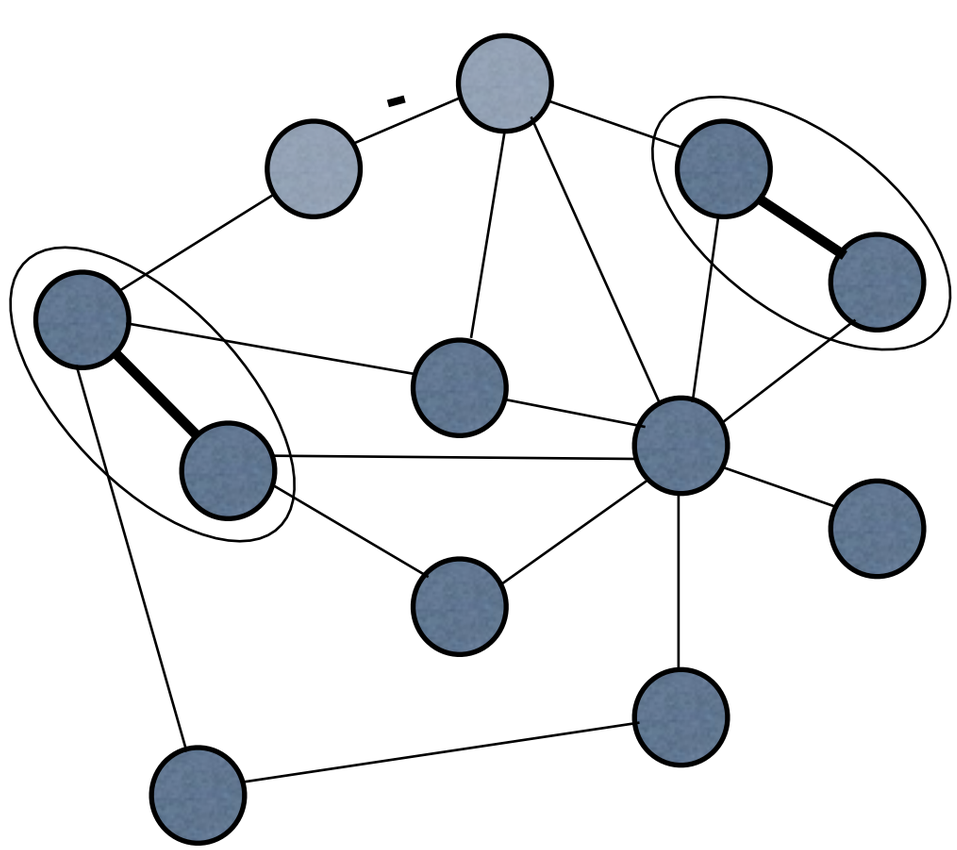
Kruskal algoritmus

felelevenítés



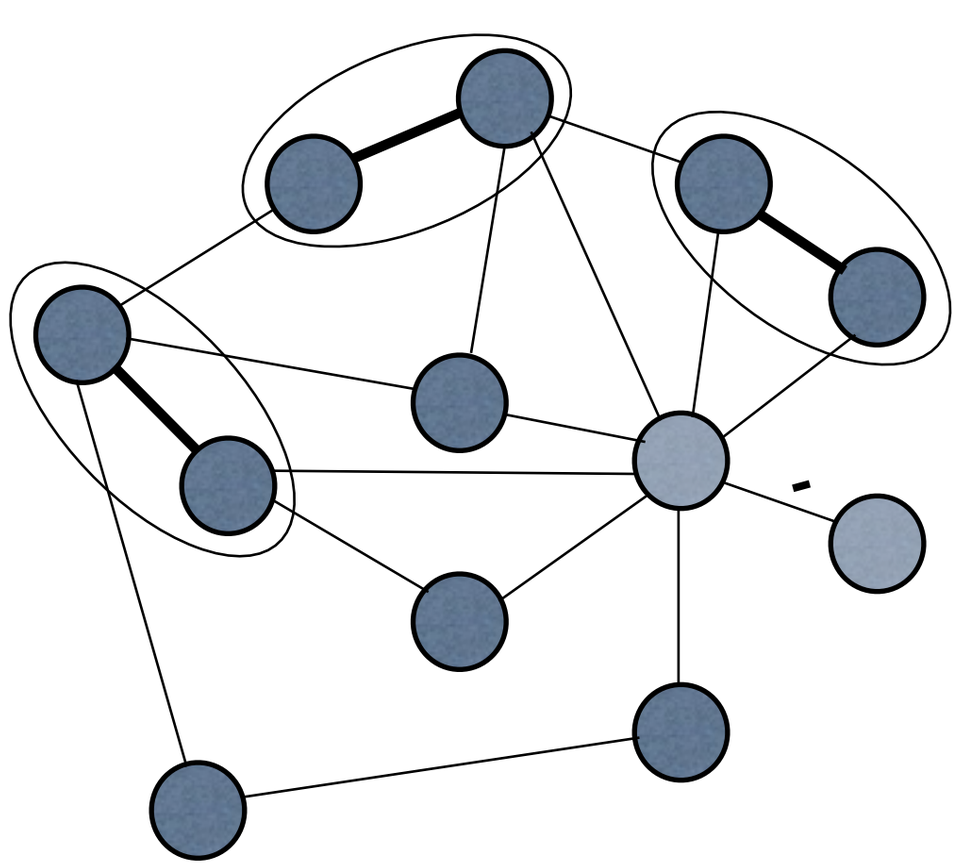
Kruskal algoritmus

felelevenítés



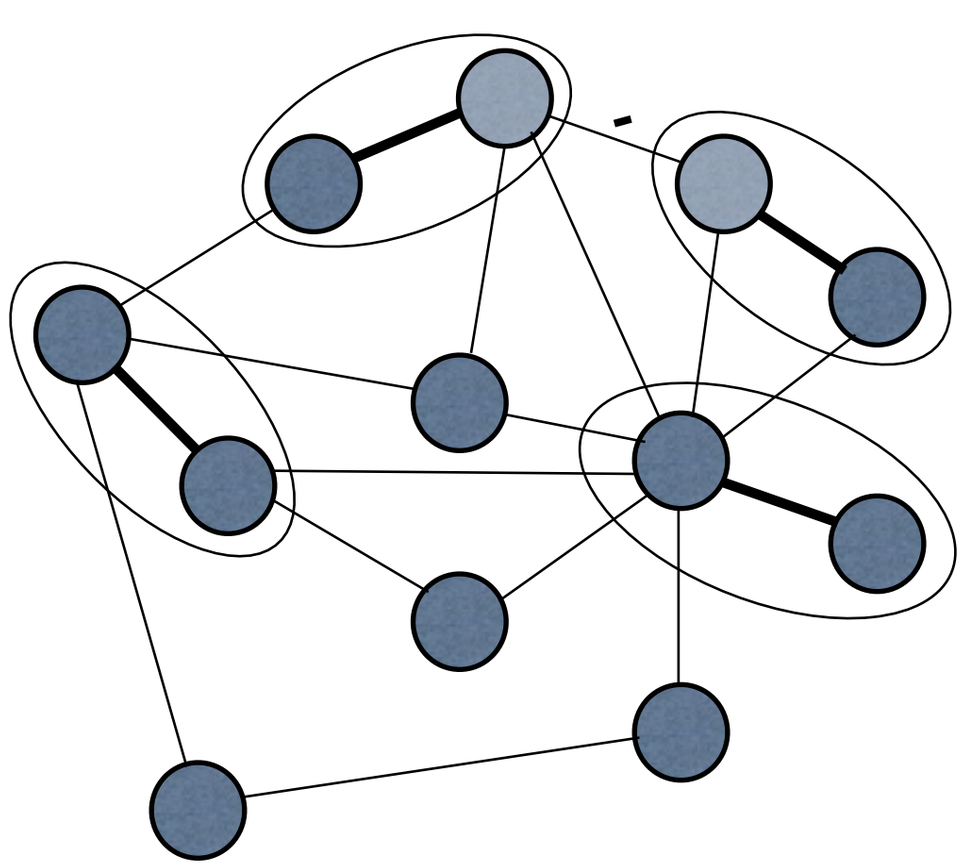
Kruskal algoritmus

felelevenítés



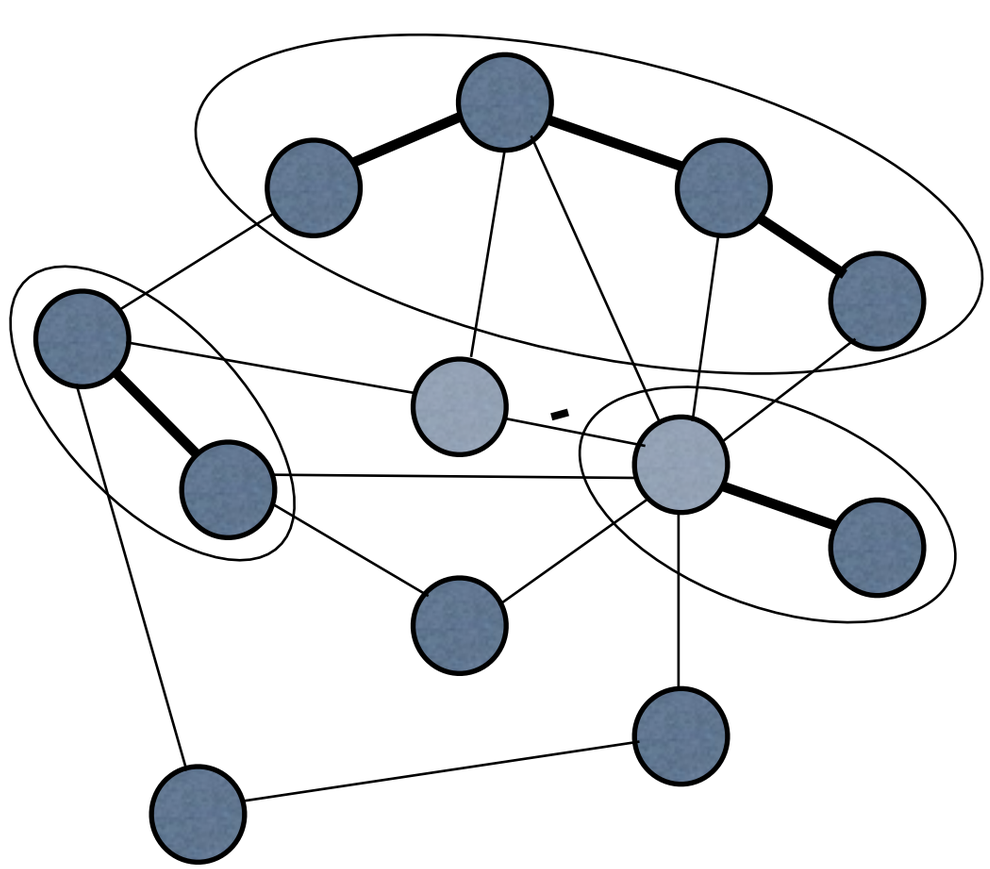
Kruskal algoritmus

felelevenítés



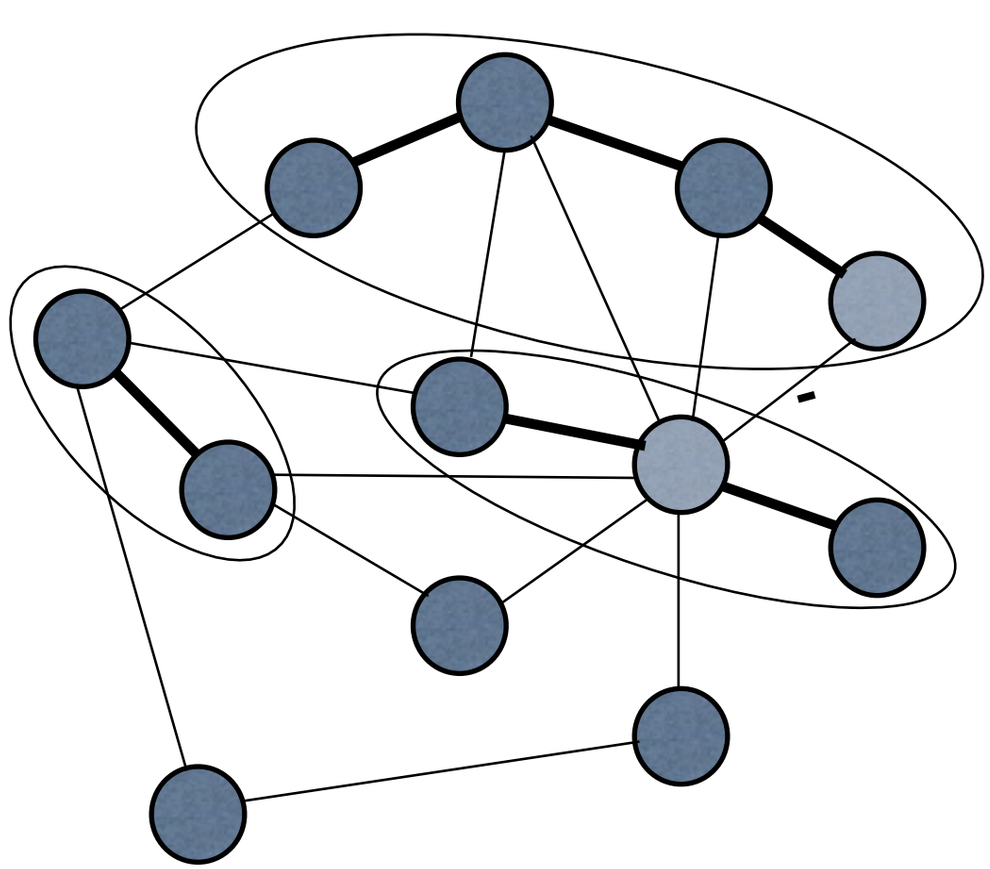
Kruskal algoritmus

felelevenítés



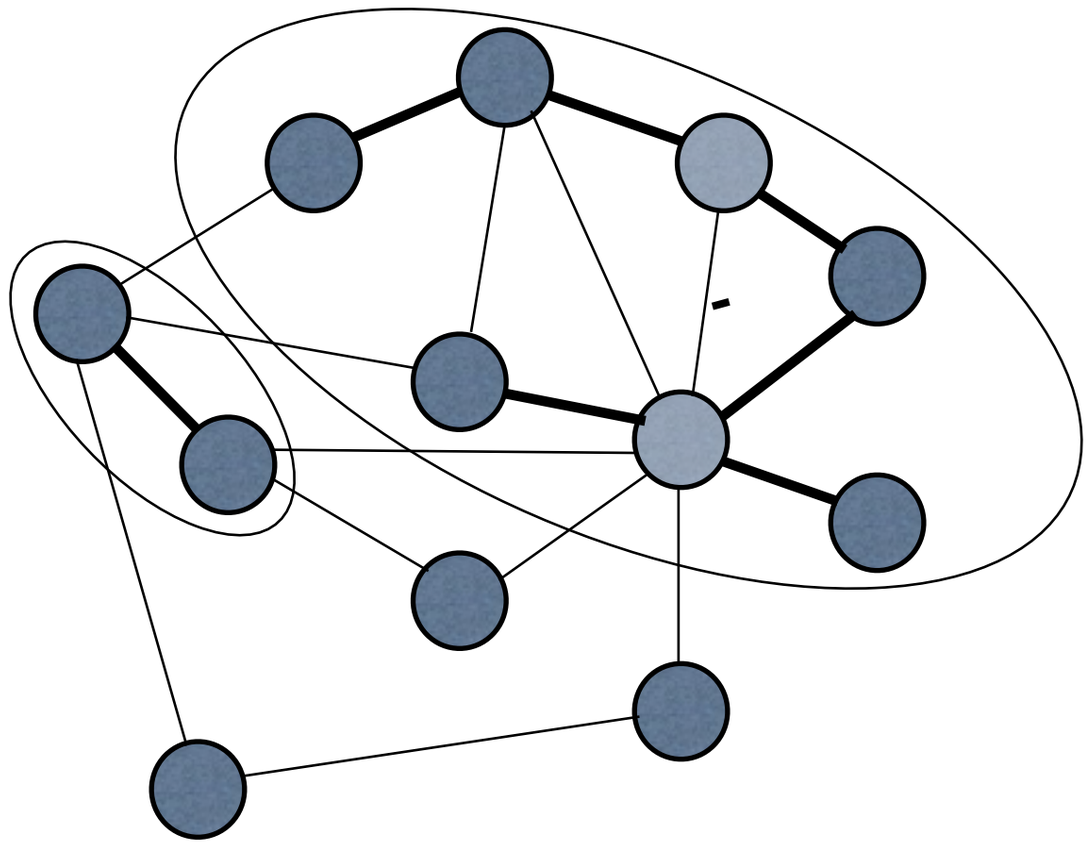
Kruskal algoritmus

felelevenítés



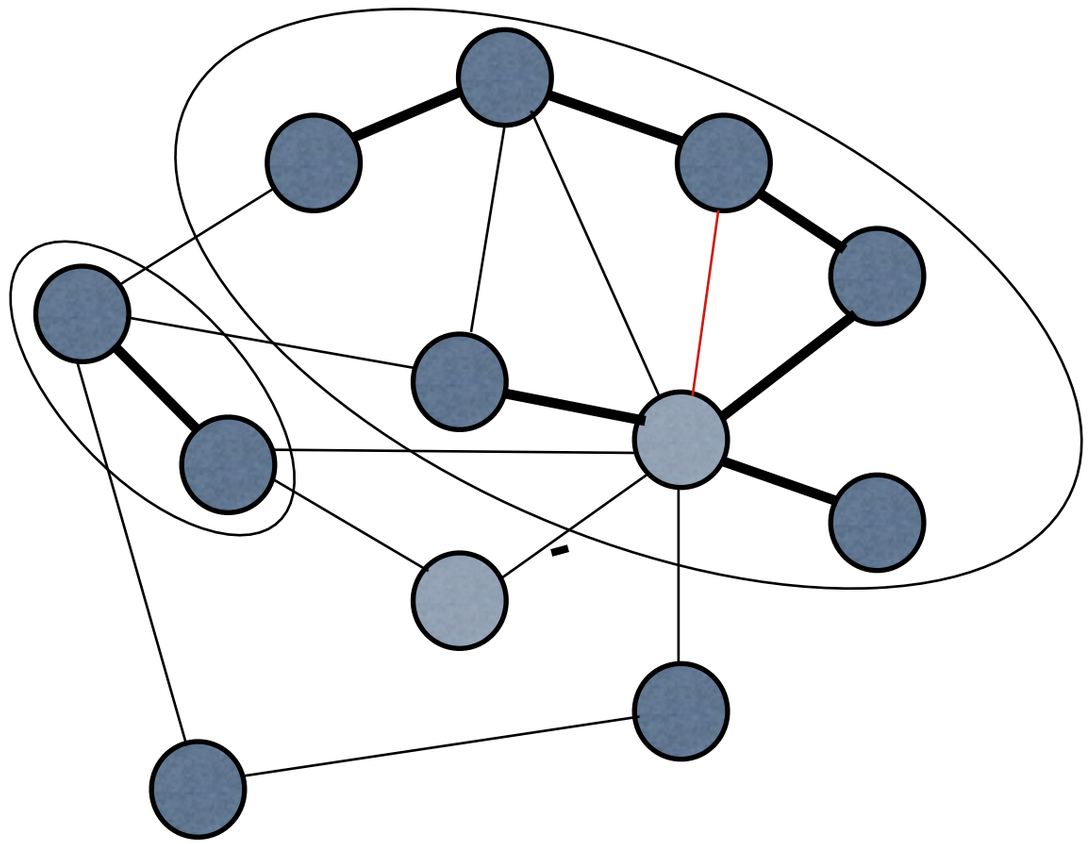
Kruskal algoritmus

felelevenítés



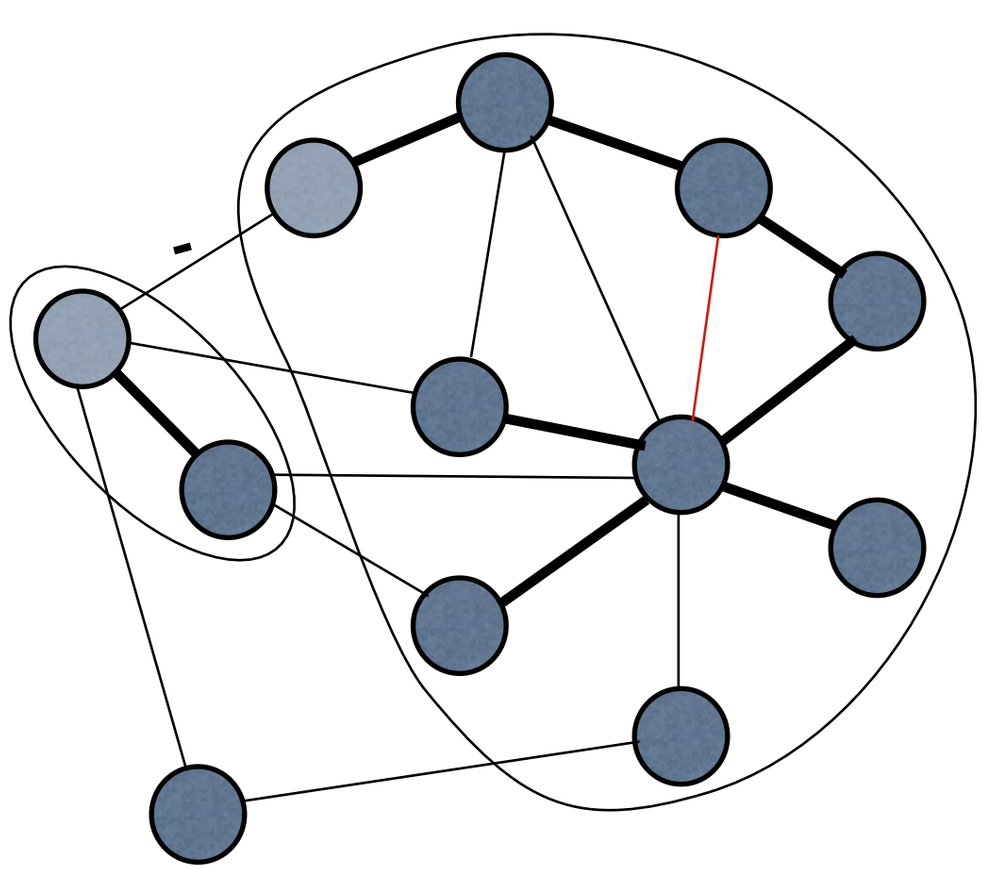
Kruskal algoritmus

felelevenítés



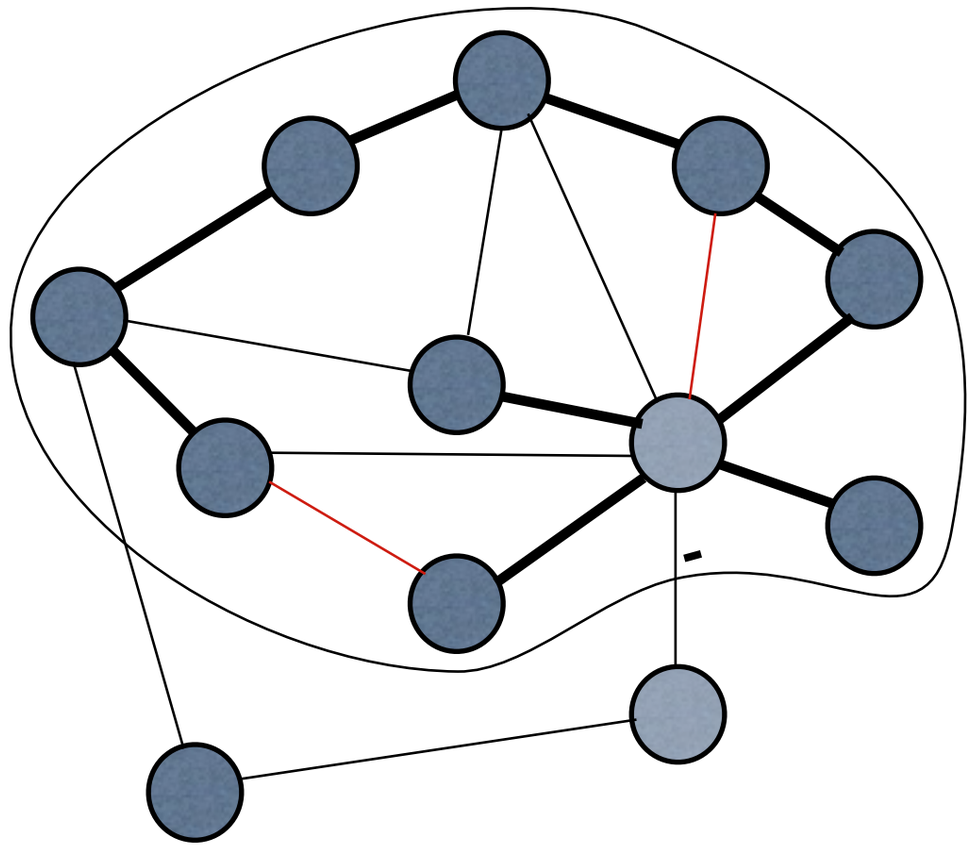
Kruskal algoritmus

felelevenítés



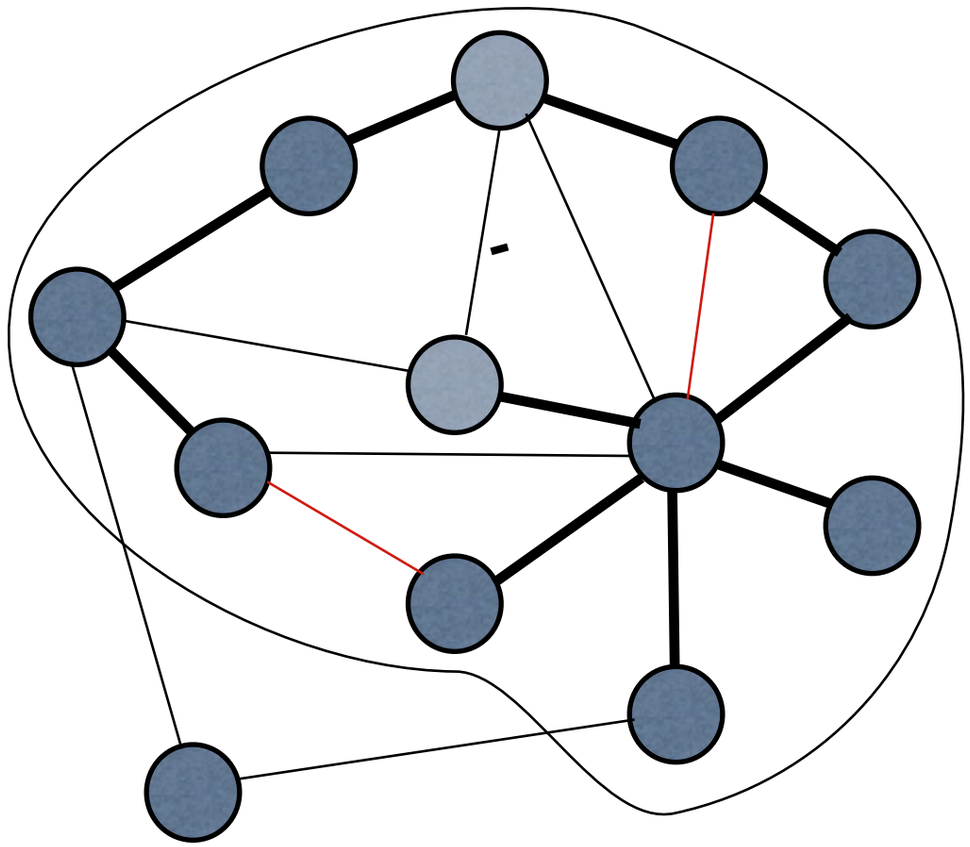
Kruskal algoritmus

felelevenítés



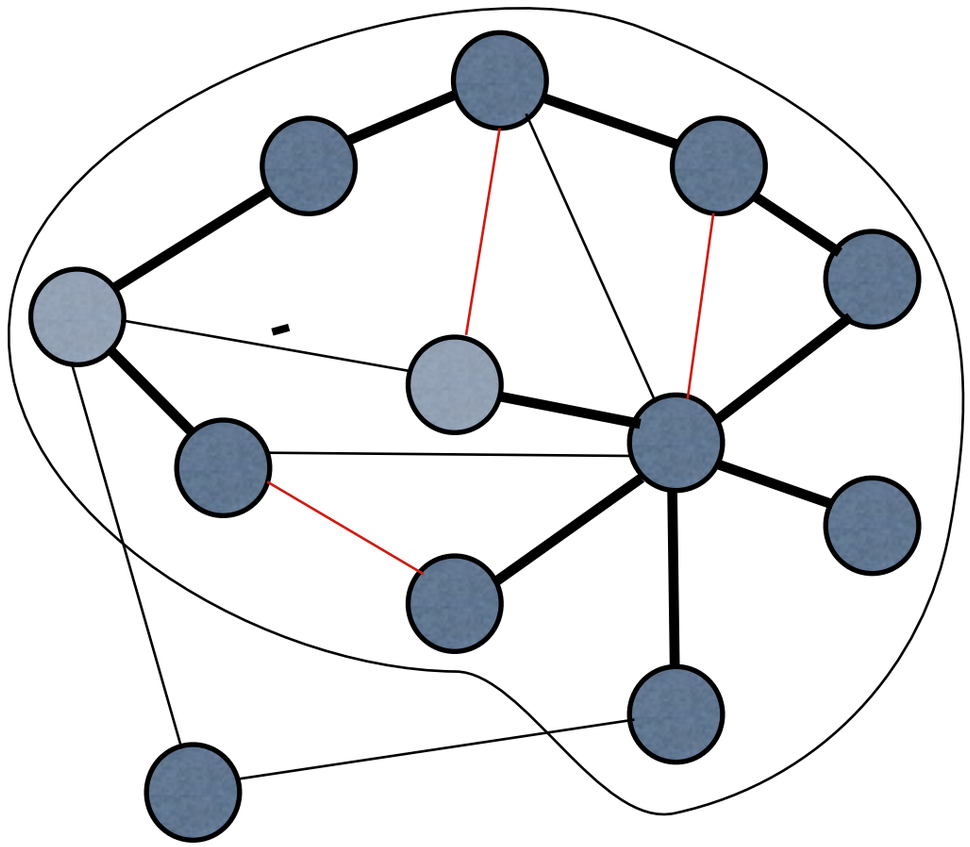
Kruskal algoritmus

felelevenítés



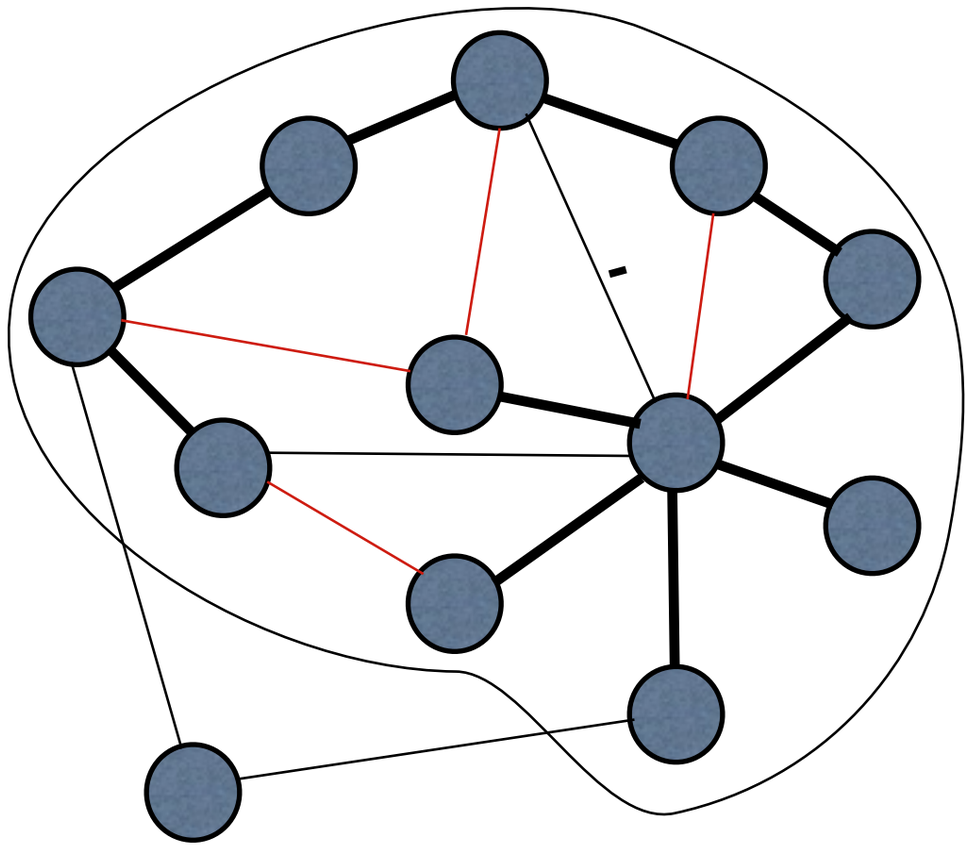
Kruskal algoritmus

felelevenítés



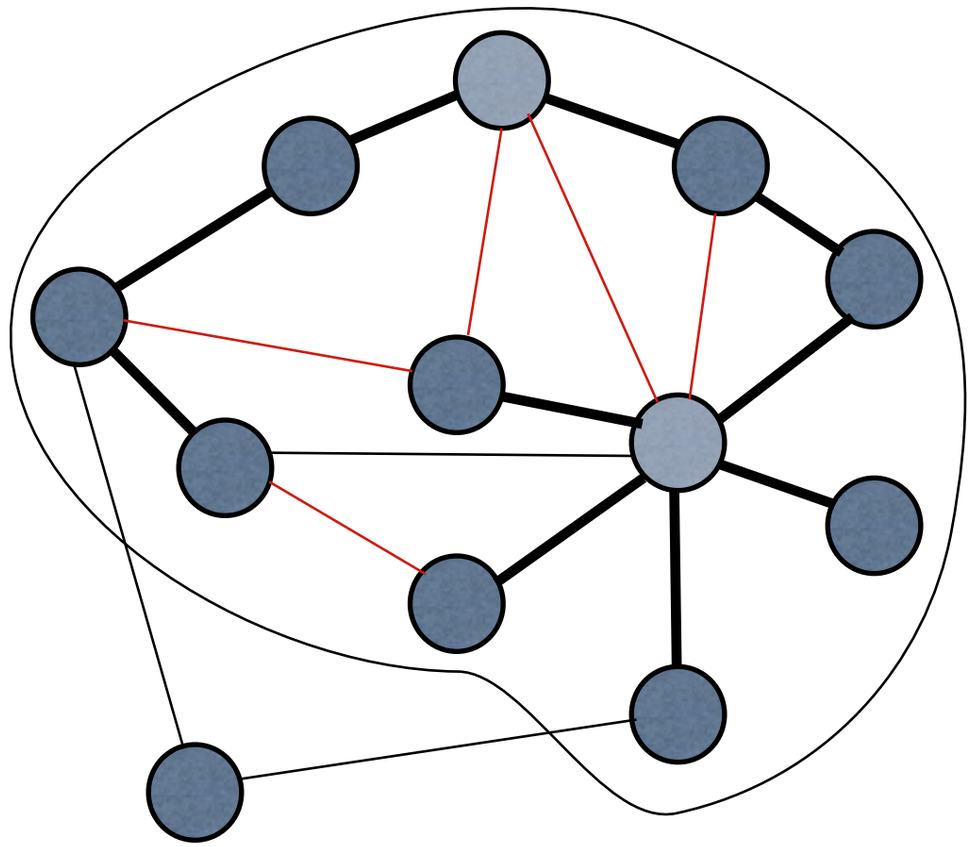
Kruskal algoritmus

felelevenítés



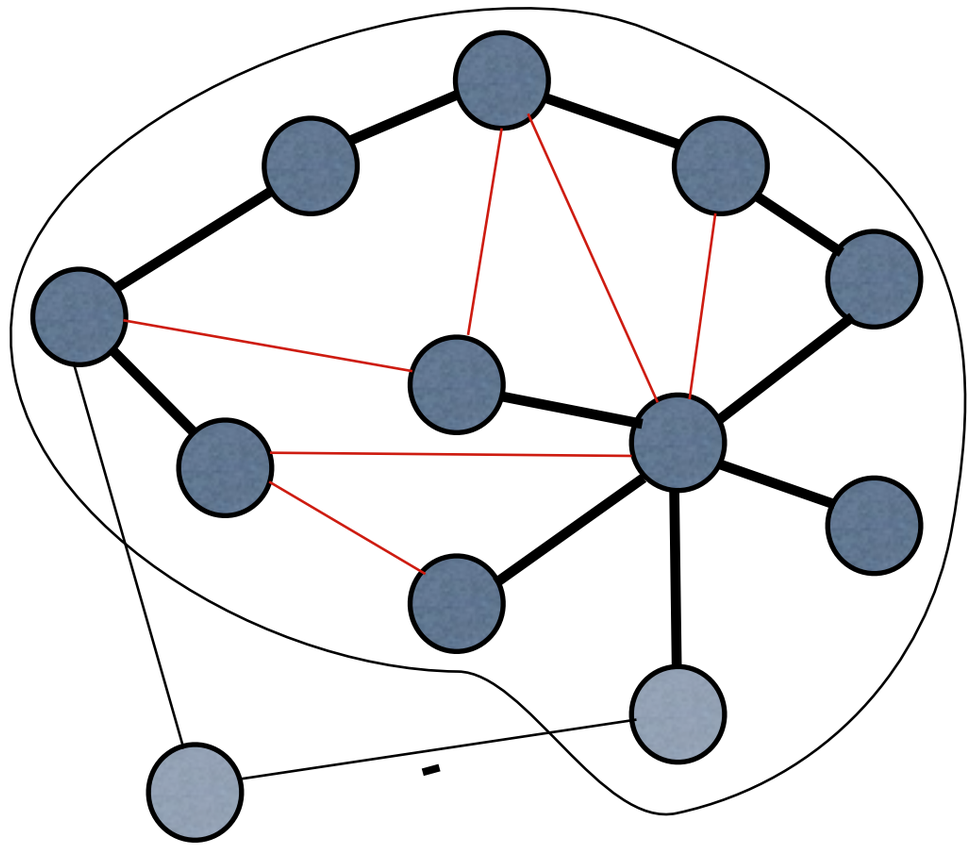
Kruskal algoritmus

felelevenítés



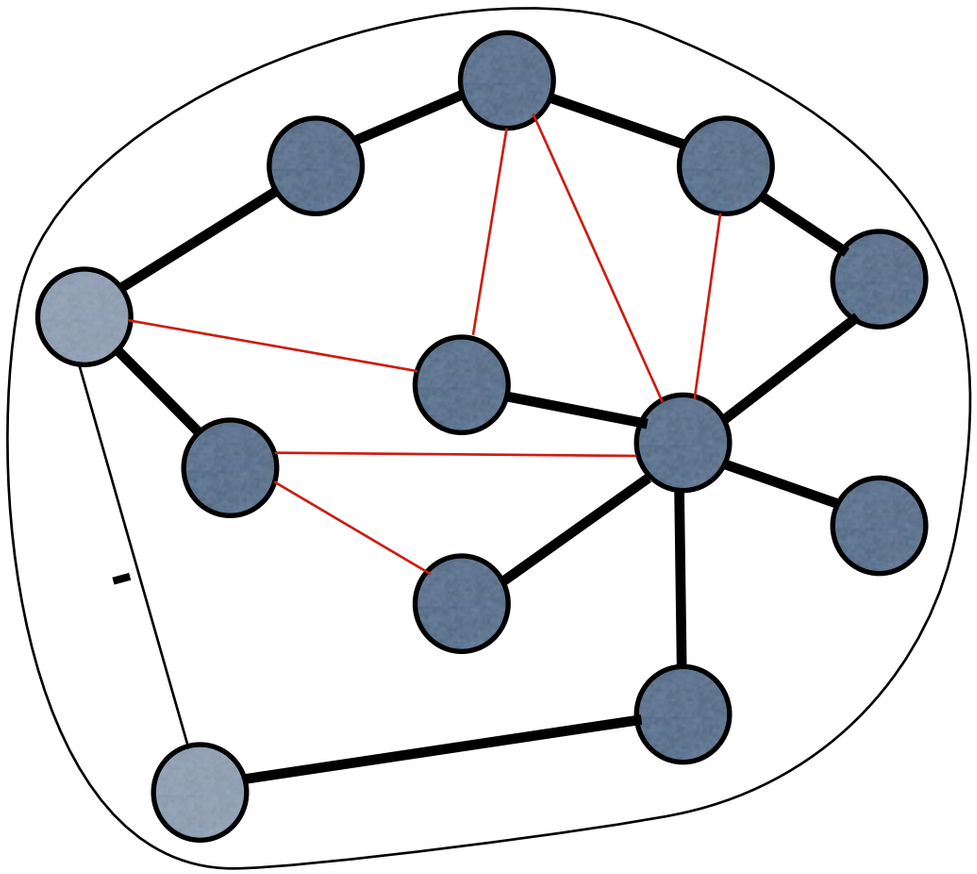
Kruskal algoritmus

felelevenítés



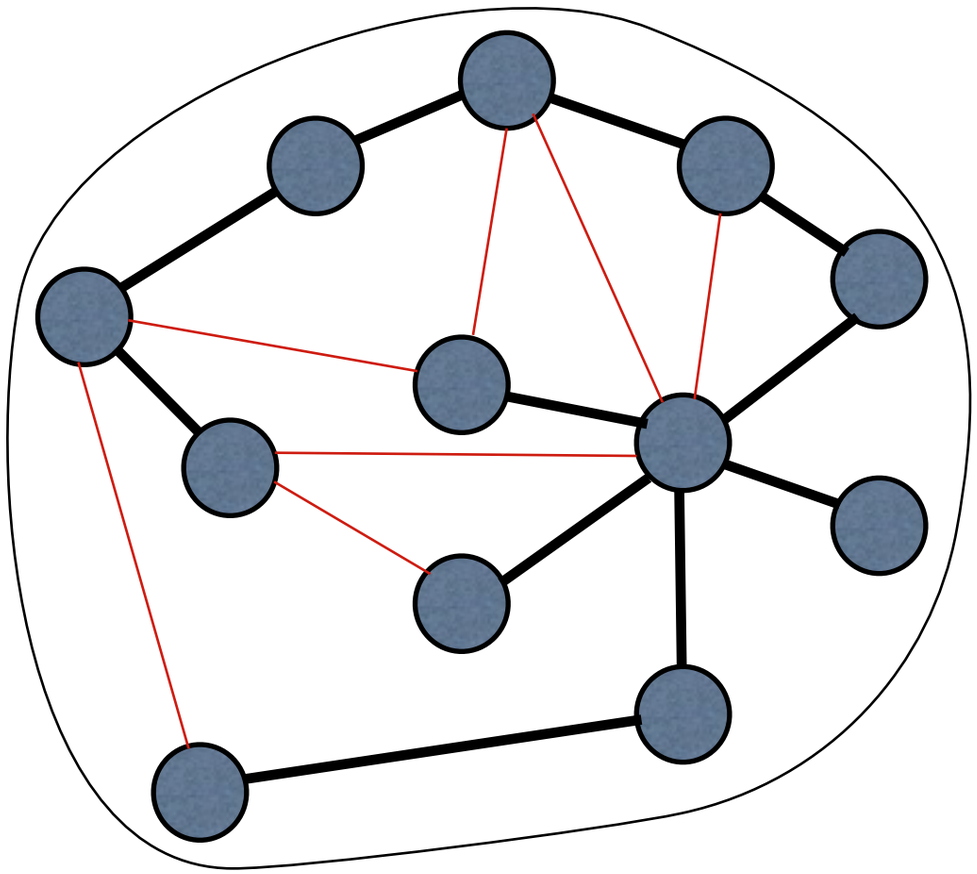
Kruskal algoritmus

felelevenítés



Kruskal algoritmus

felelevenítés



Kruskal algoritmus

felelevenítés

```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

Honnan lehet tudni, hogy az adott él 2 végén lévő pont egy halmazban van-e?

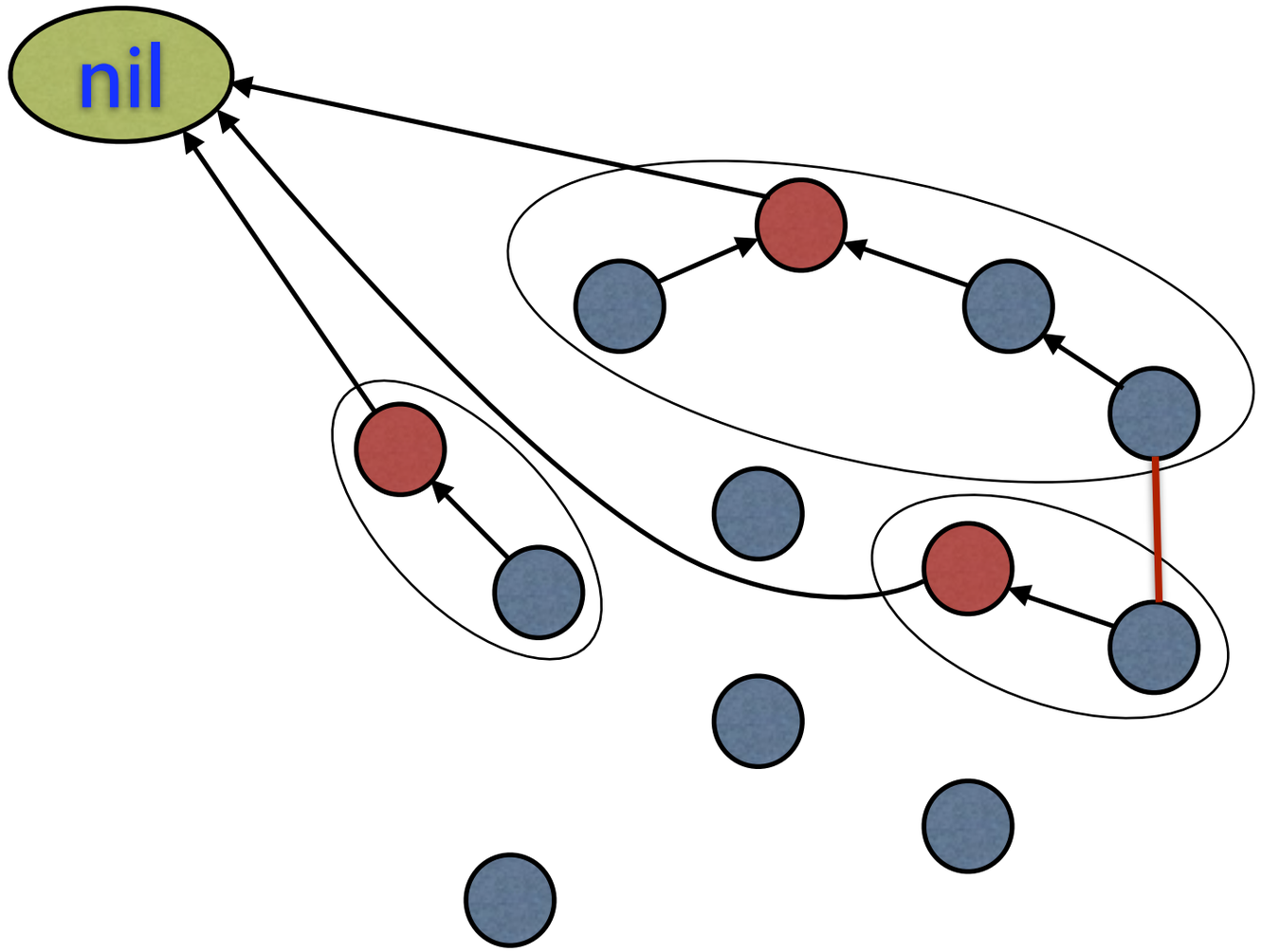
Hogyan tudom a kezdetben n halmazt hatékonyan egyesítgetni?

Halmazok listájában az összes halmazt végigkeresni O(n) lenne :(

Halmaz (piros-fekete fa) adatszerkezettel 2 halmaz egyesítése O(n) lenne :(

?

Halmazerdő adattípus



Kruskal algoritmus halmazerdővel

```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

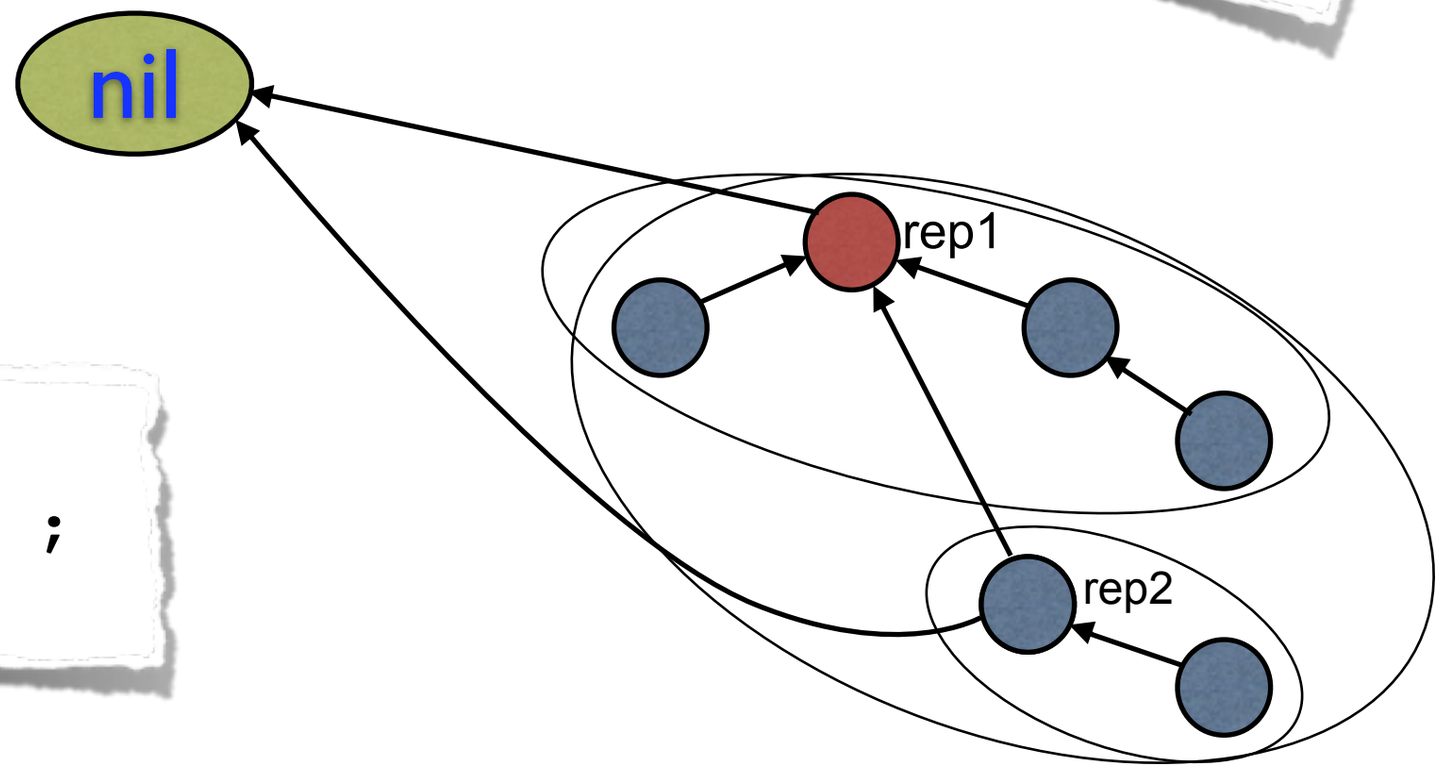
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
   }
   return q ;
}
```

Kruskal algoritmus halmazerdővel

```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
   }
   return q ;
}
```

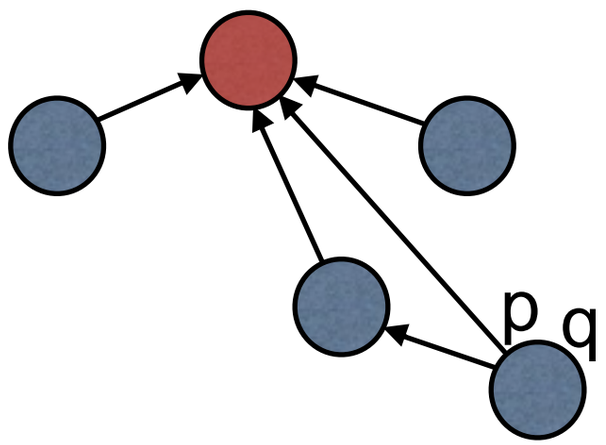
```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

```
unio(rep1,rep2) {
   rep2.kovetkezo=rep1 ;
}
```



Halmazerdő tömörítése

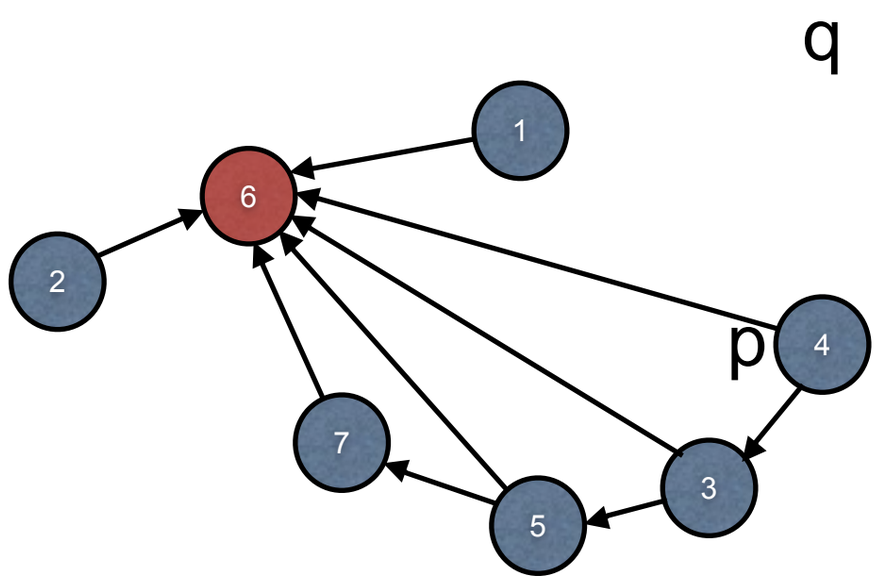
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      q=p ;
      p=p.kovetkezo ;
      q.kovetkezo=p.kovetkezo ;
   }
   return p ;
}
```



Halmazerdő hatékonyabb tömörítése

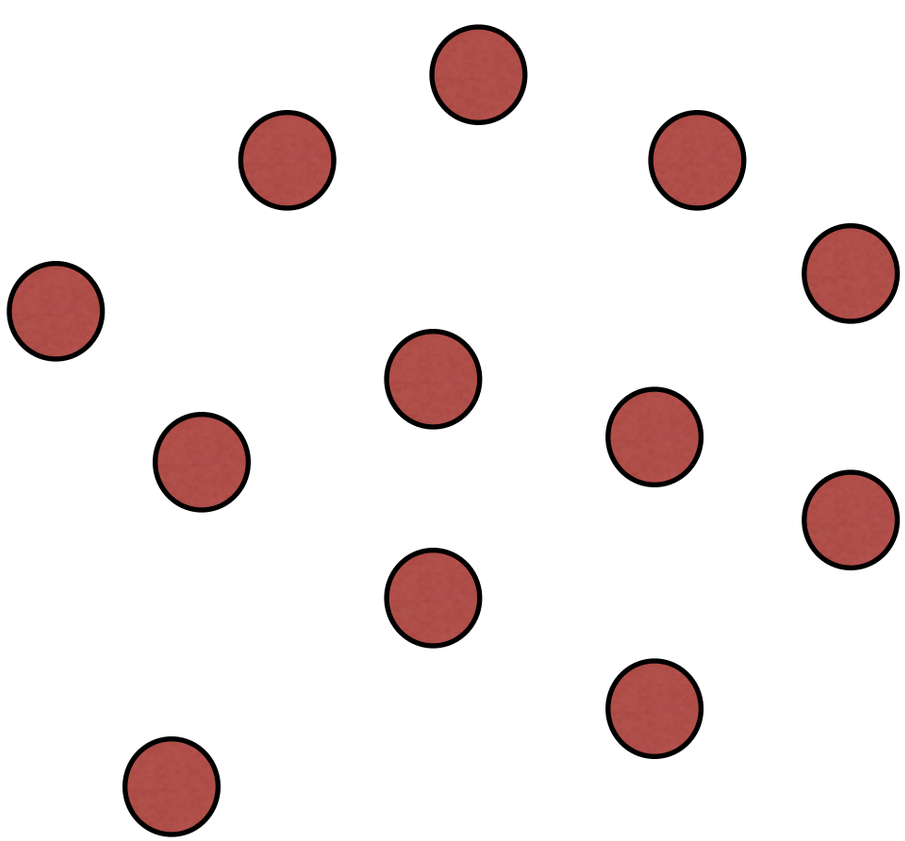
```
gpont rep(gpont p) {
   gpont q;
   while (p.kovetkezo!=nil) {
      if (p.kovetkezo.kovetkezo!=nil) sorba(p) ;
      p=p.kovetkezo ;
   }
   while (!sor_ures()) {
      q=sorbol() ;
      q.kovetkezo=p ;
   }
   return p ;
}
```

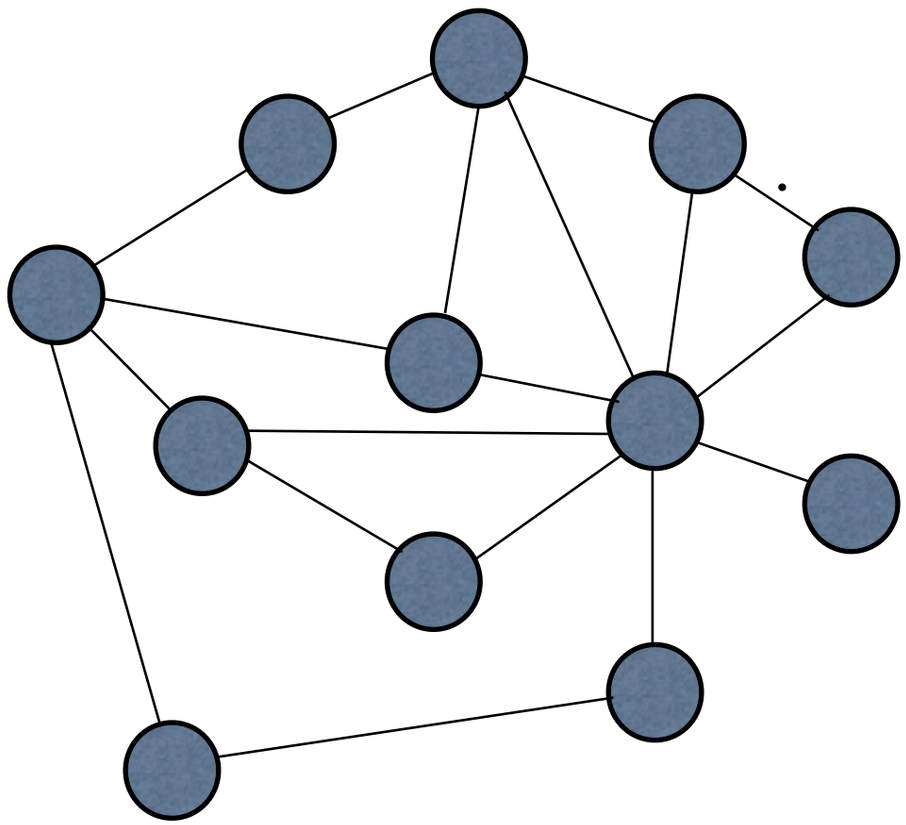
5 3 4



Halmazok egyesítése

Kruskal algoritmus

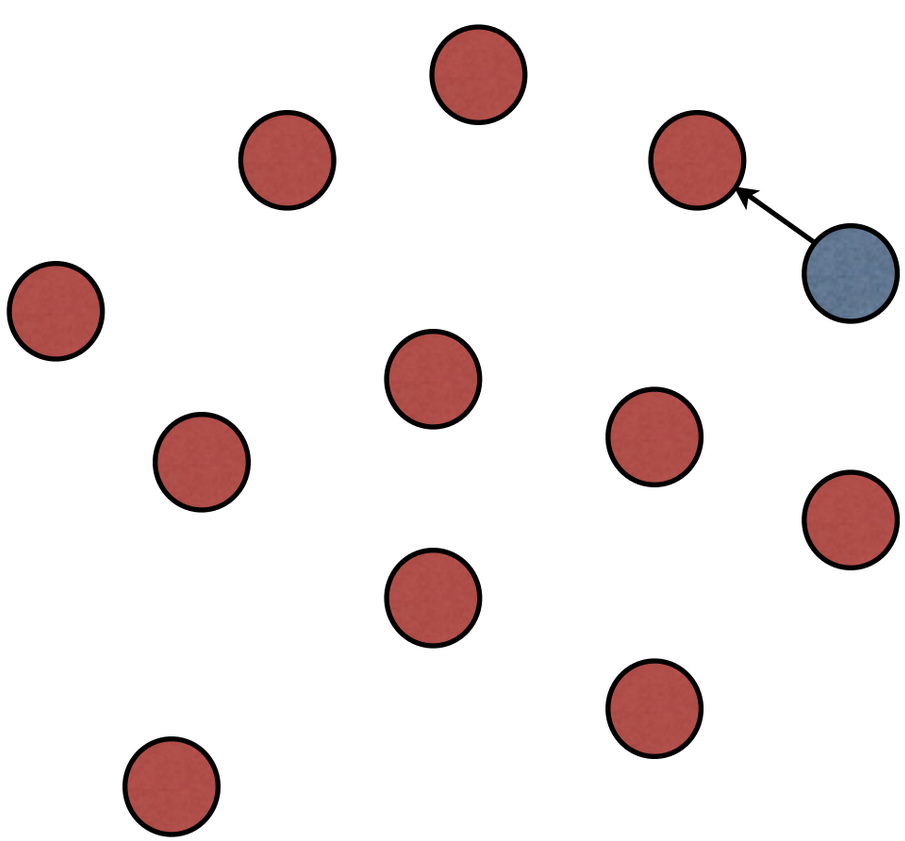


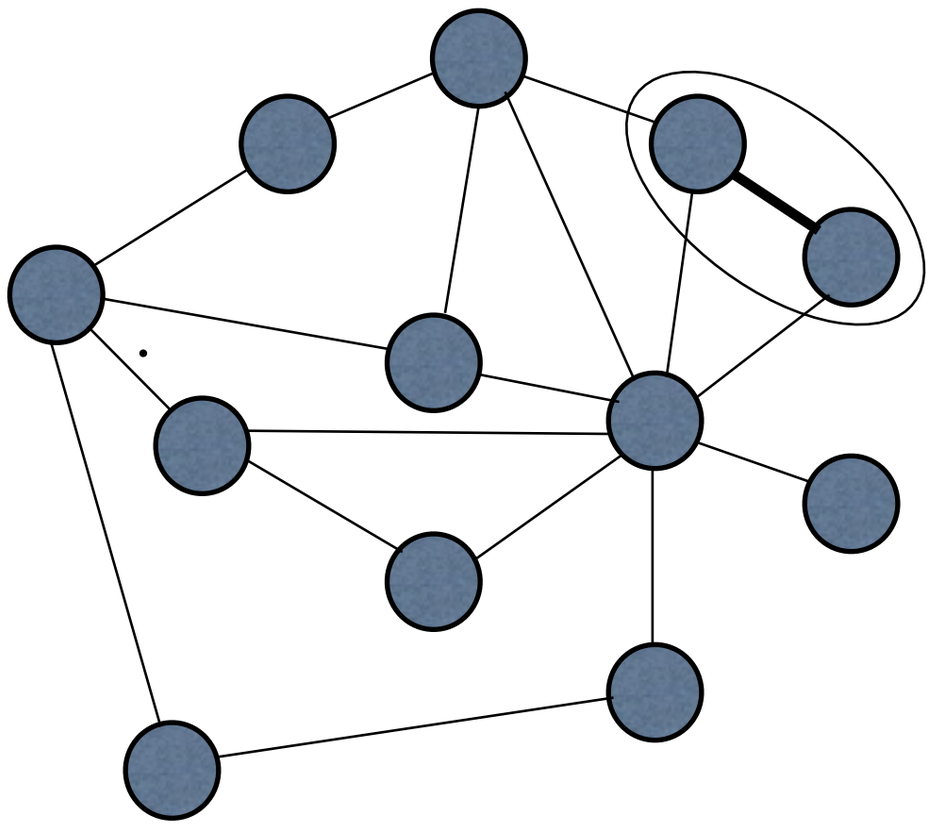


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

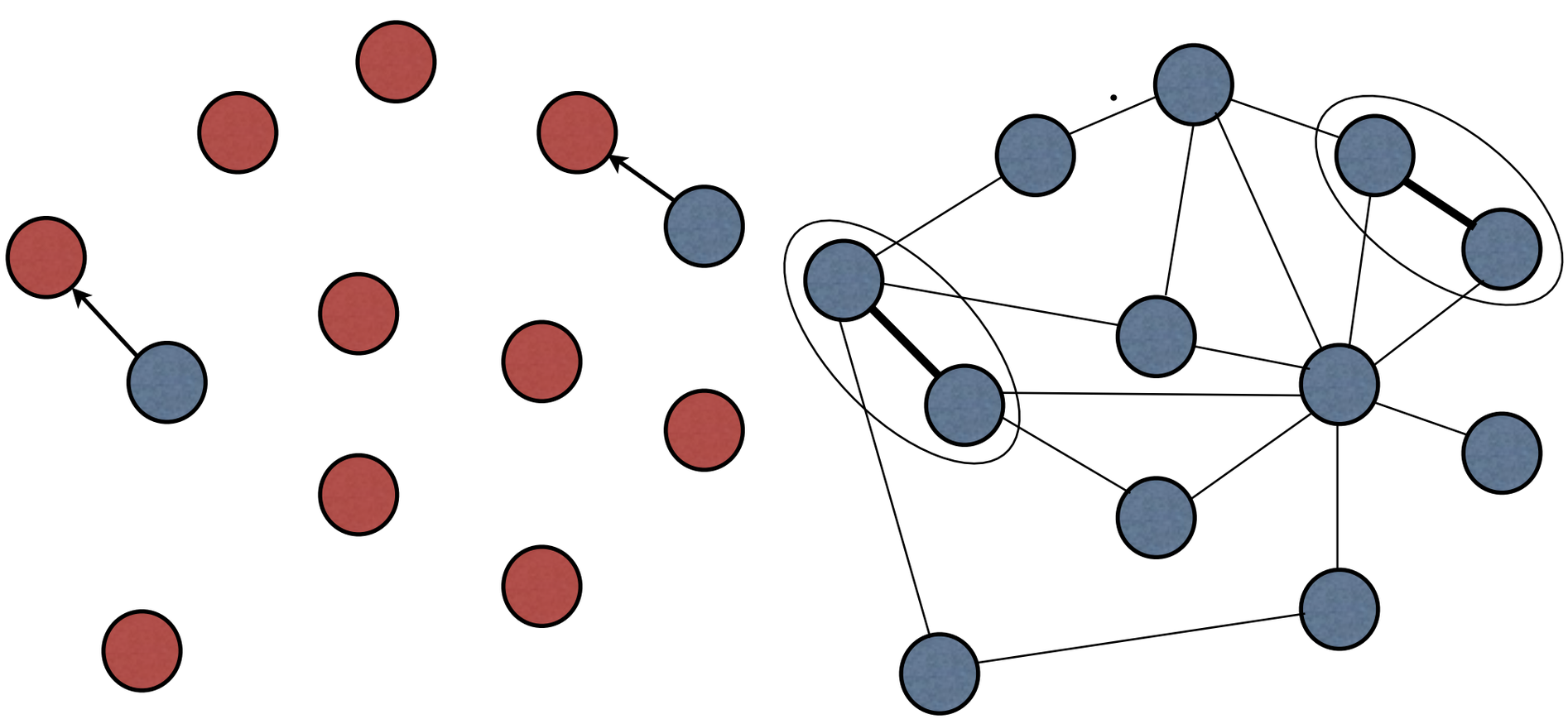




Halmazok egyesítése

halmazerdő felépítése

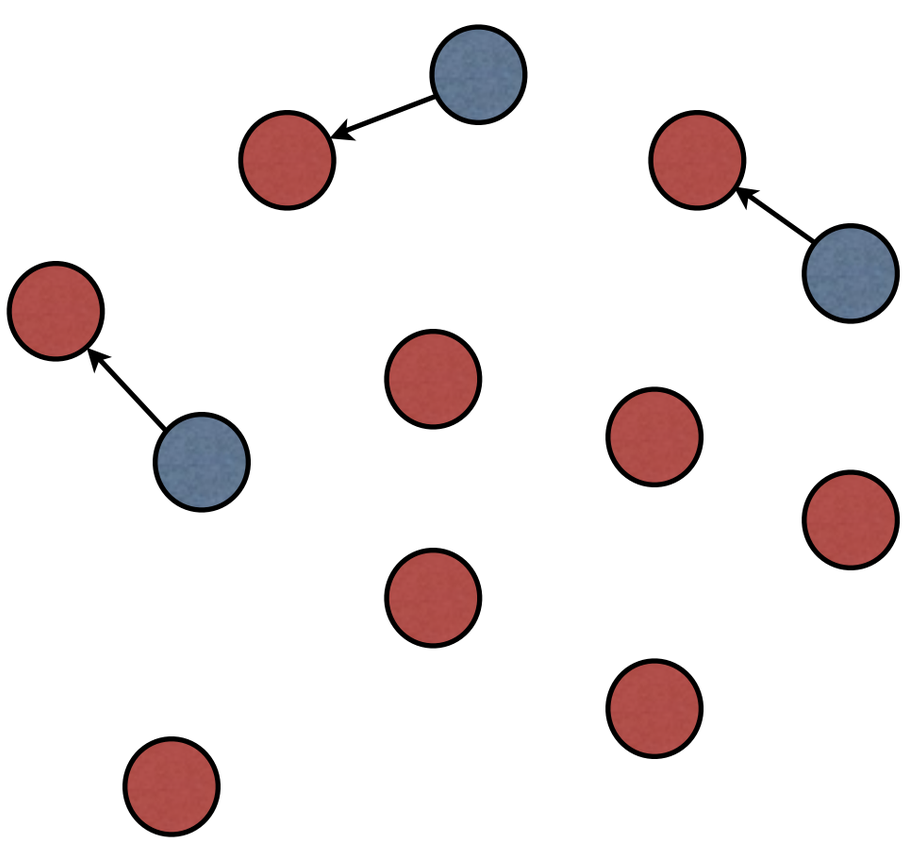
Kruskal algoritmus

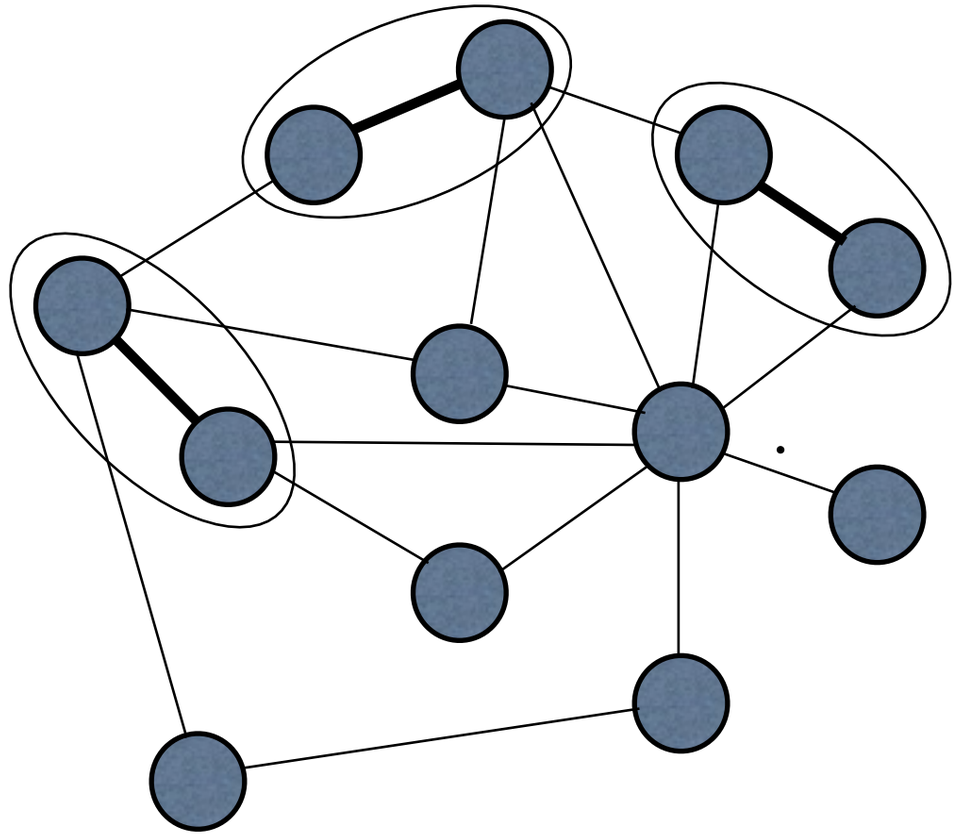


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

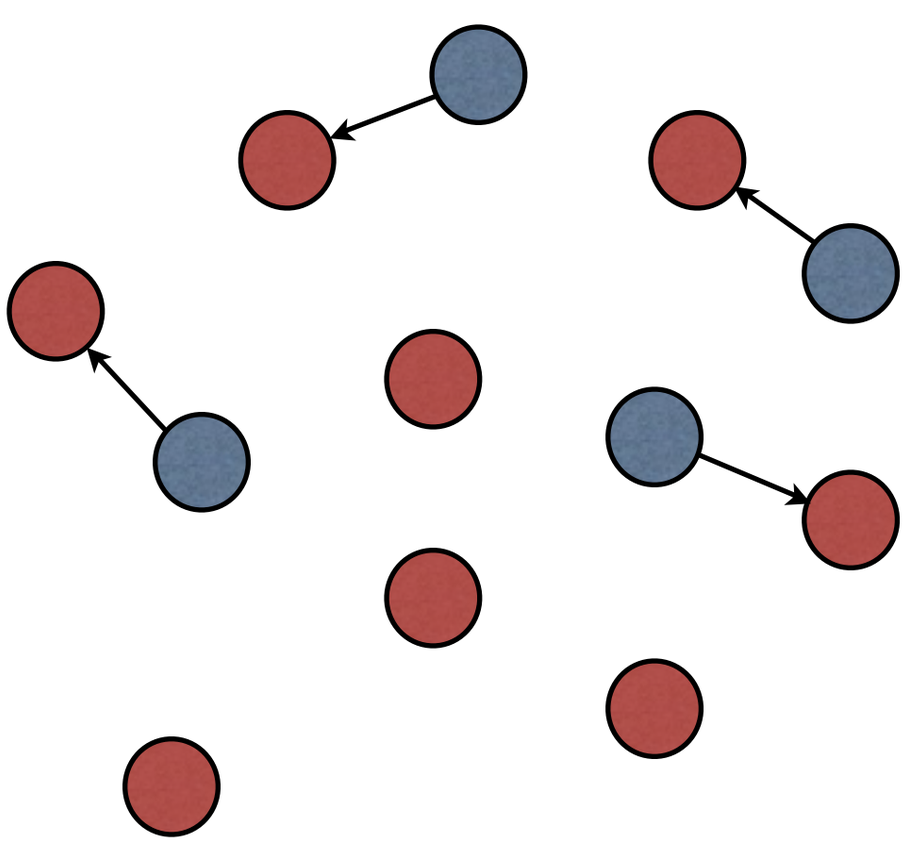


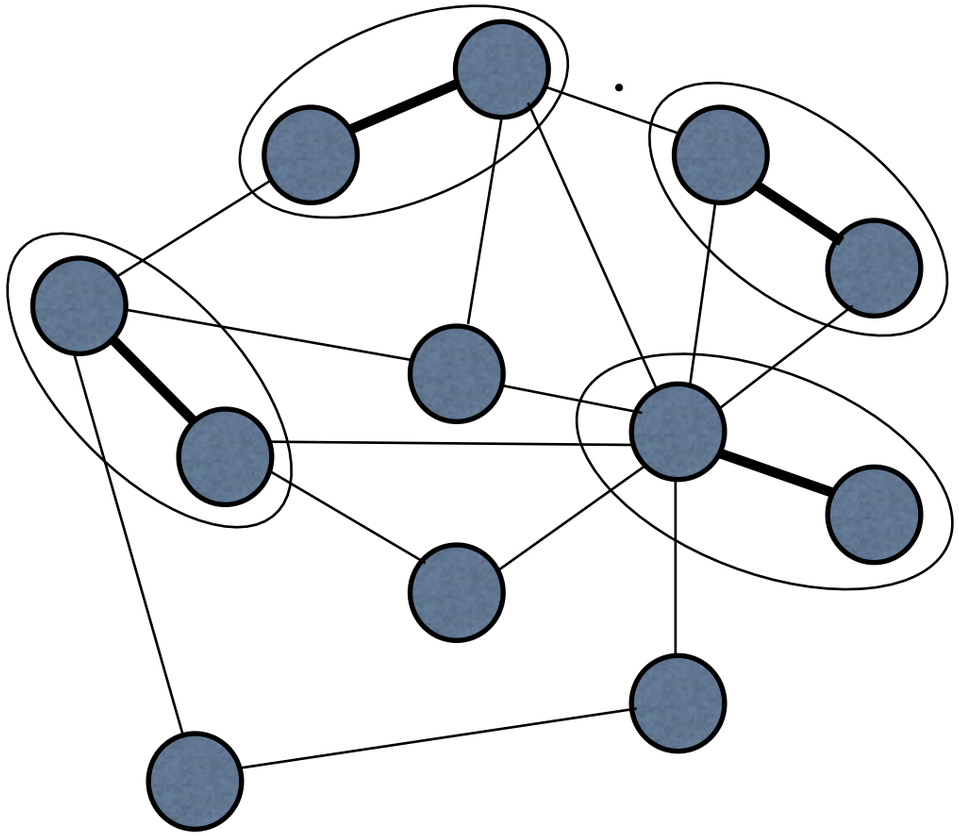


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

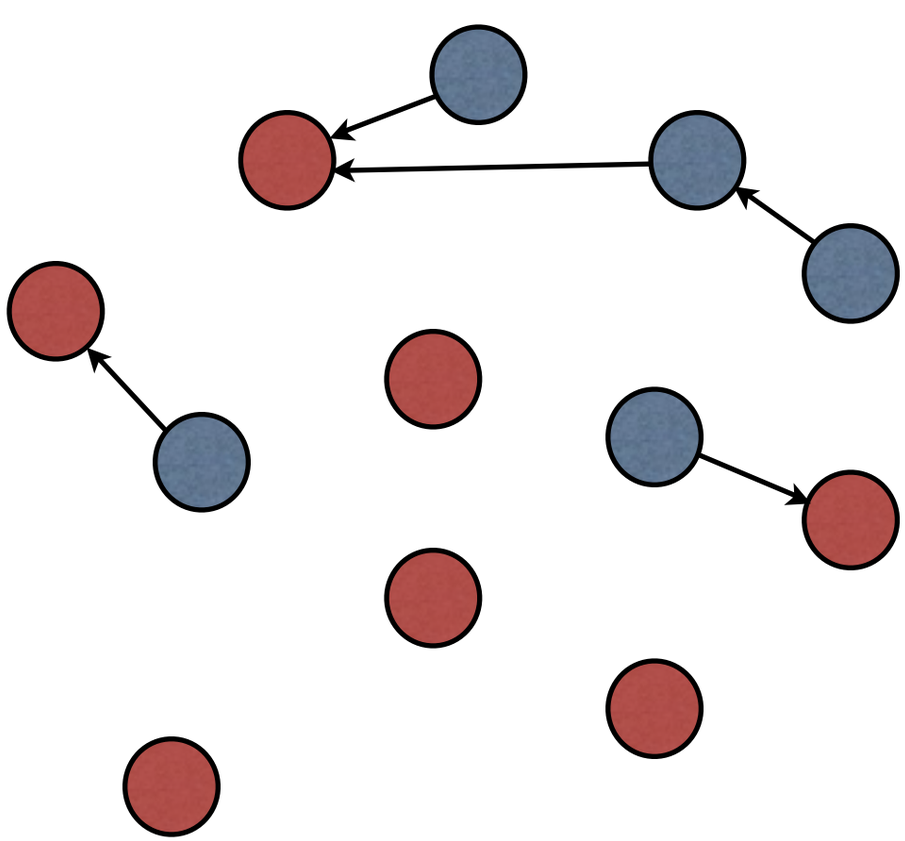


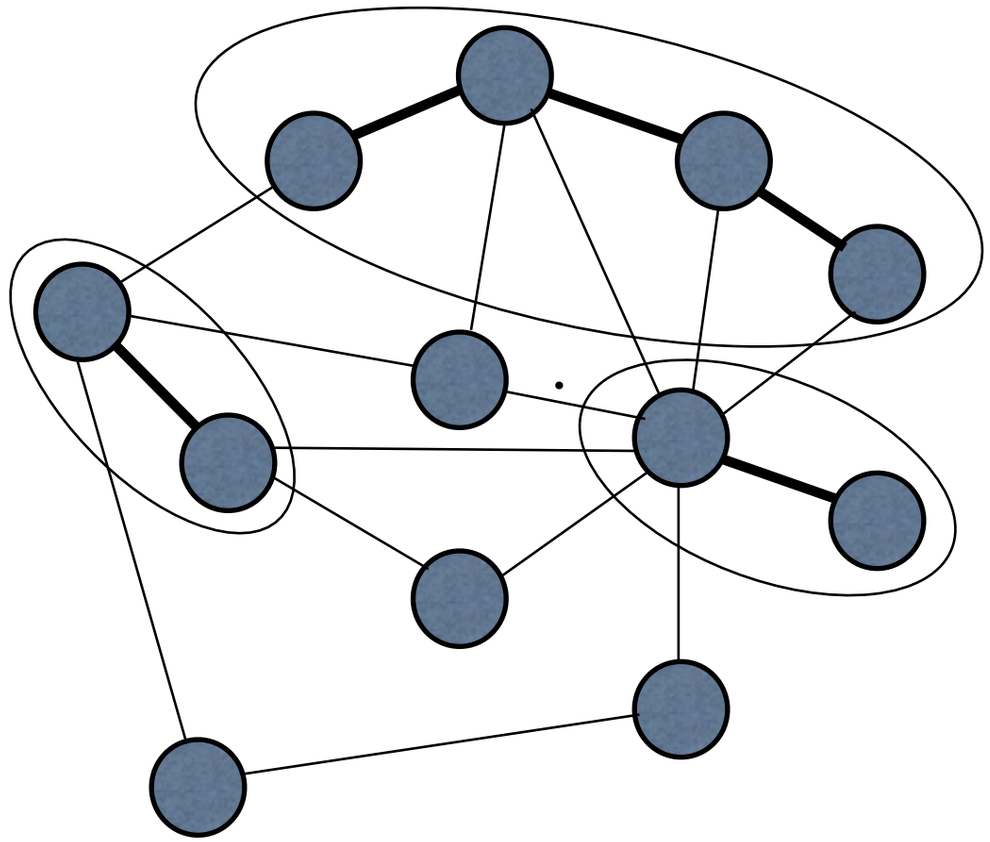


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

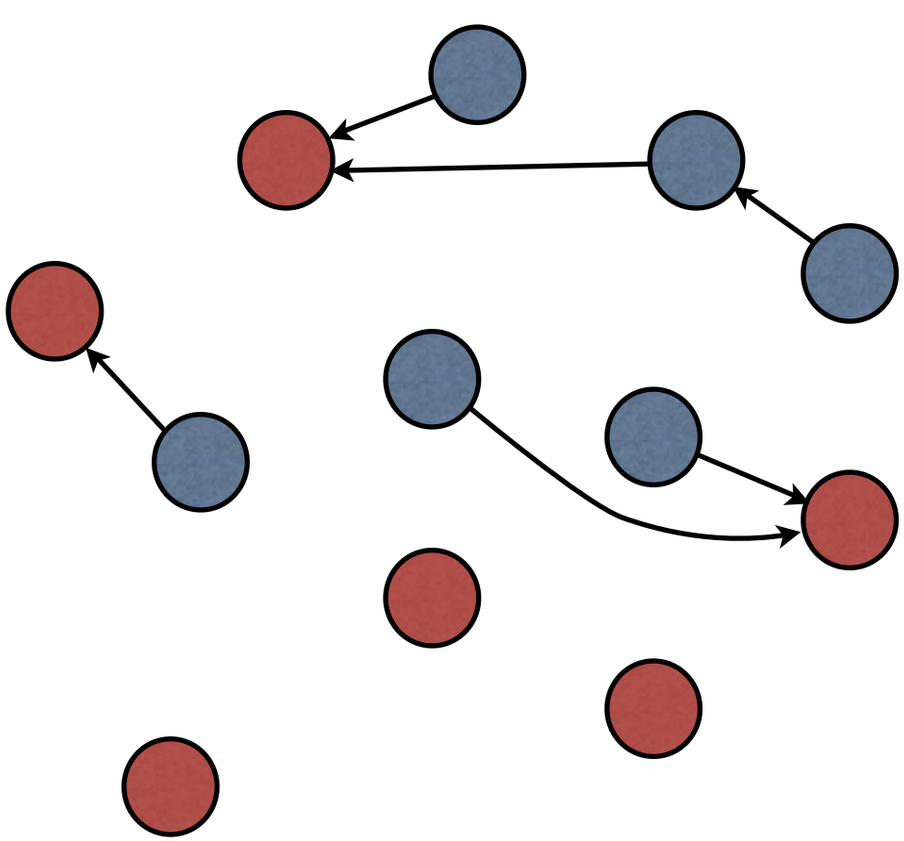


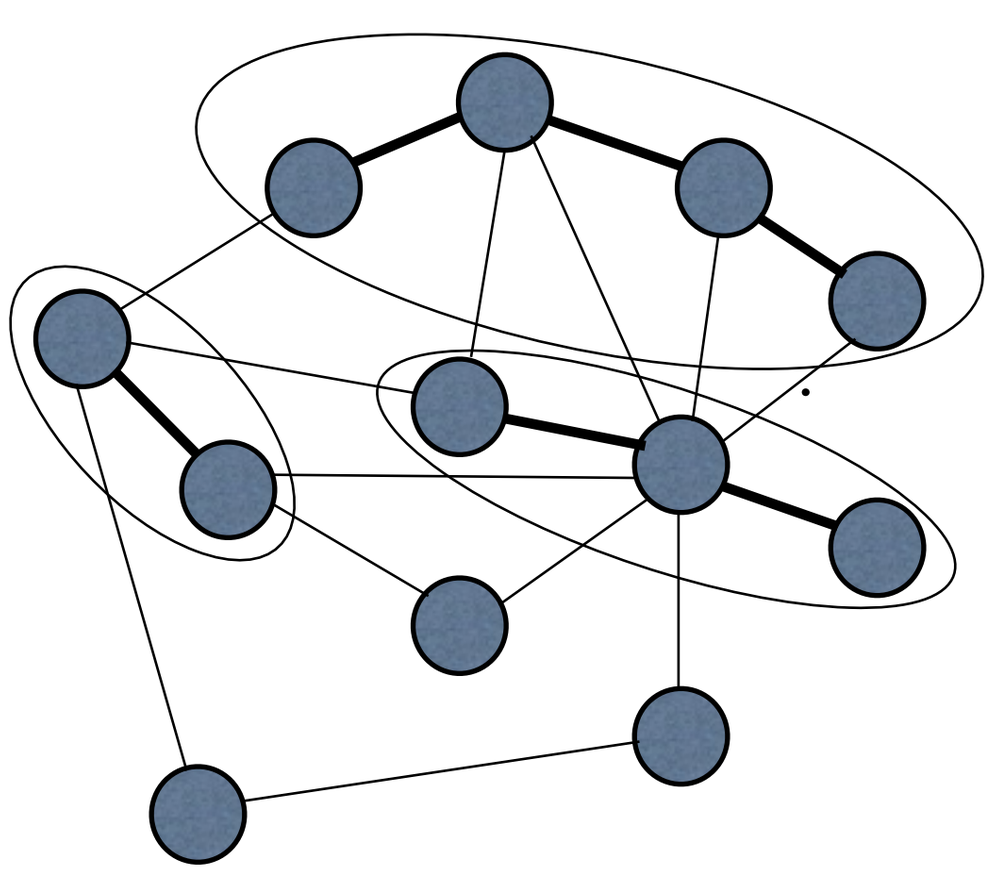


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

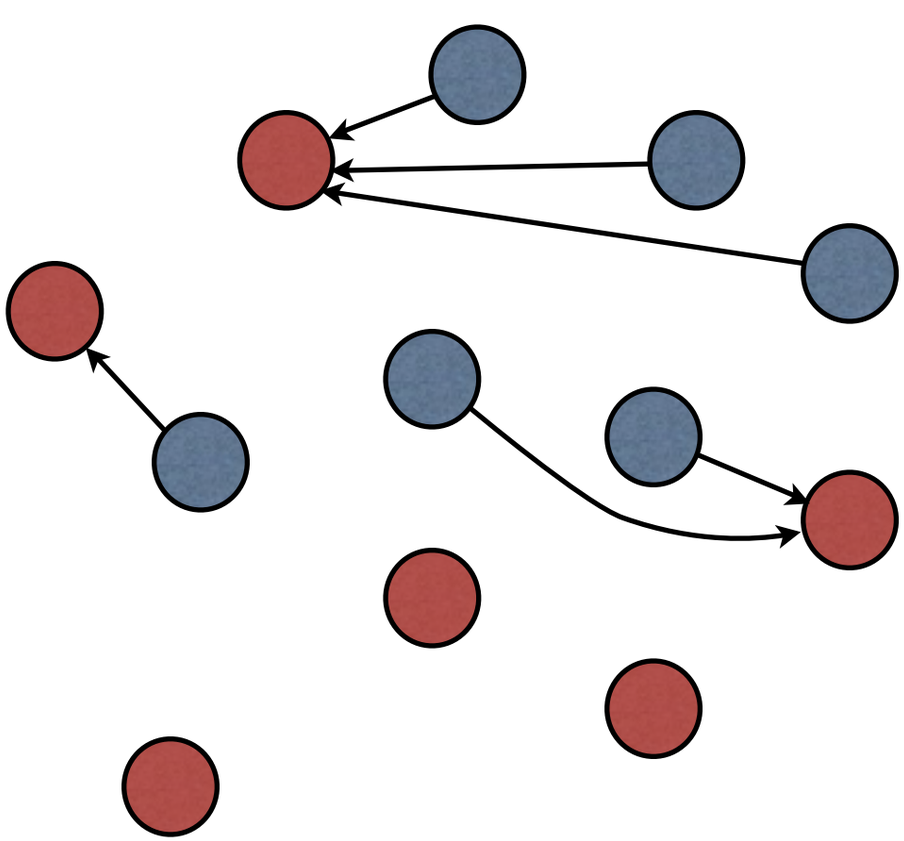


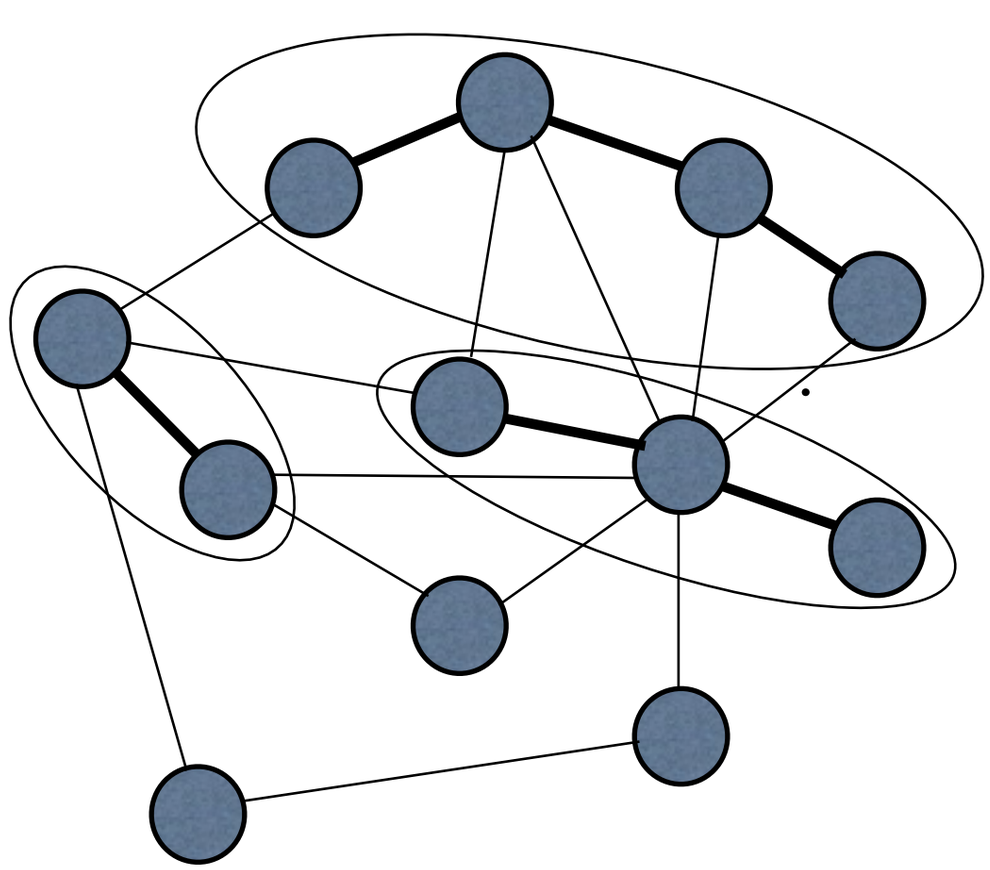


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

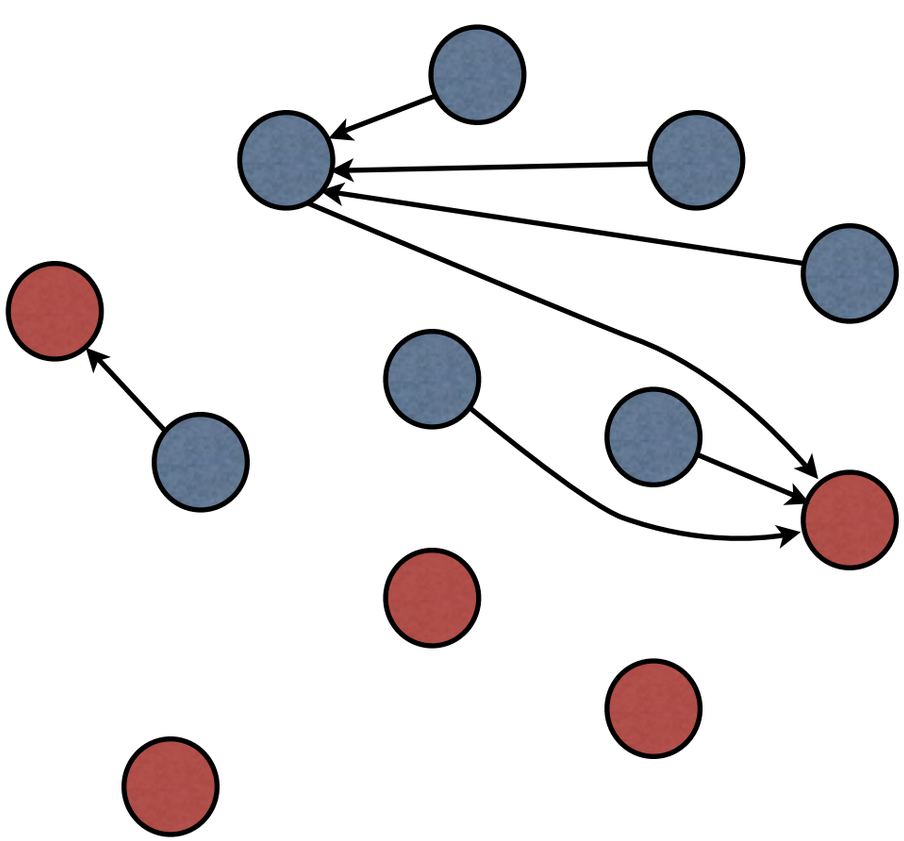


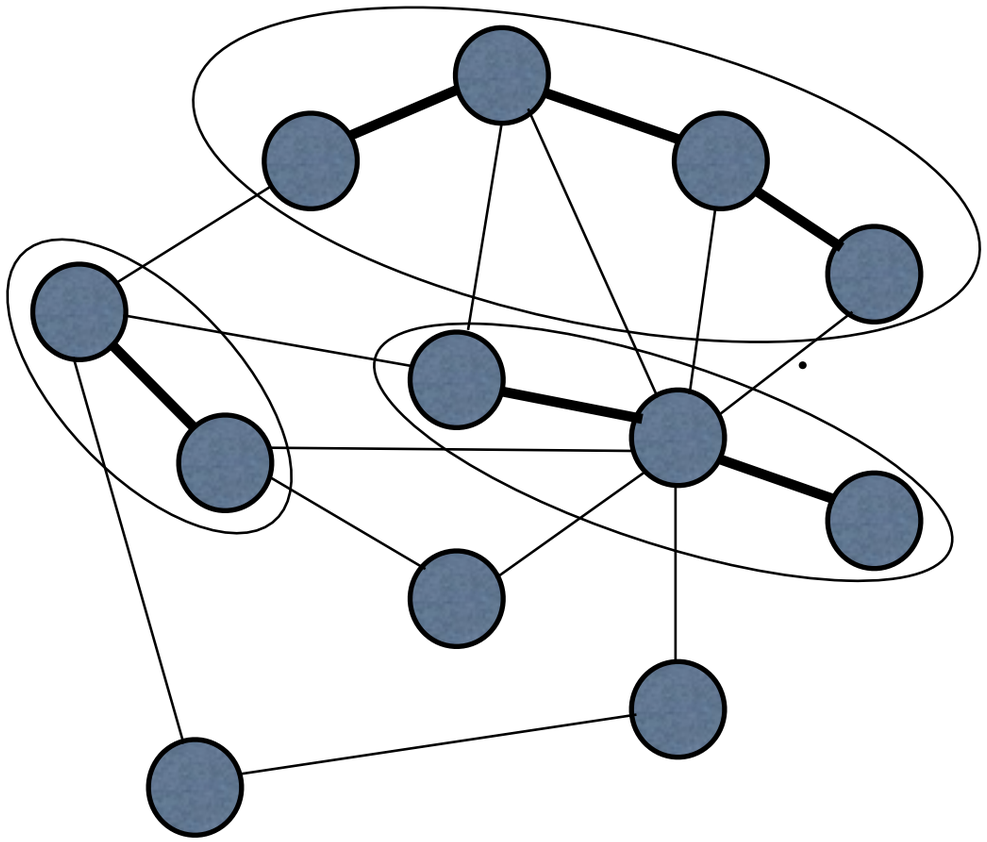


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

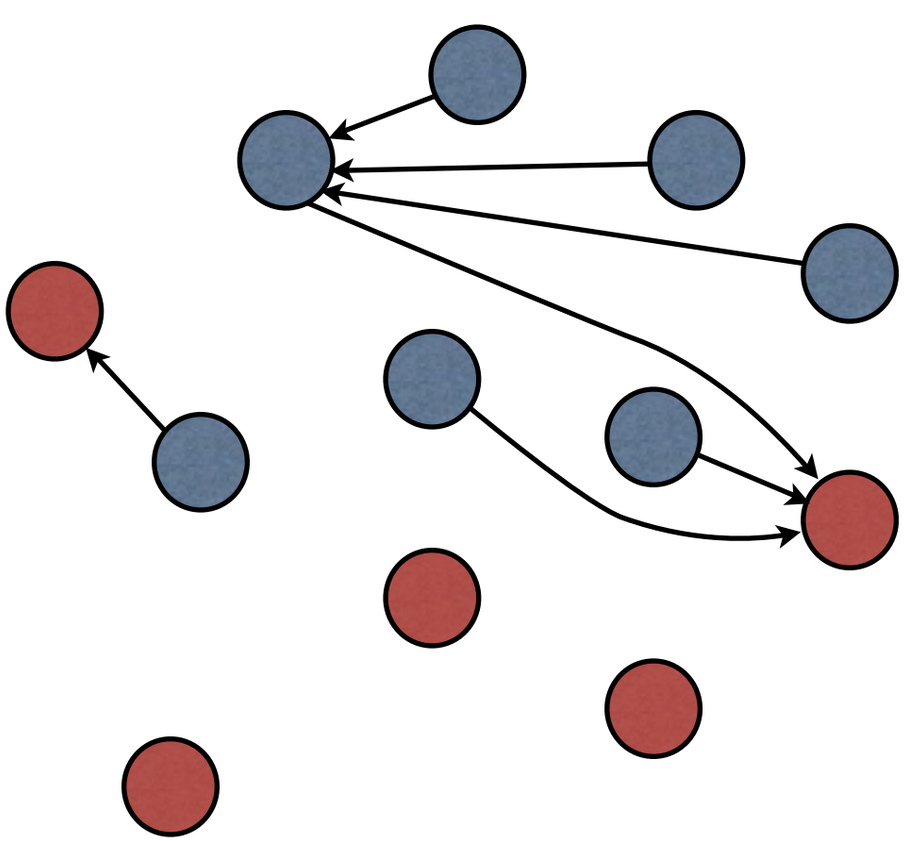


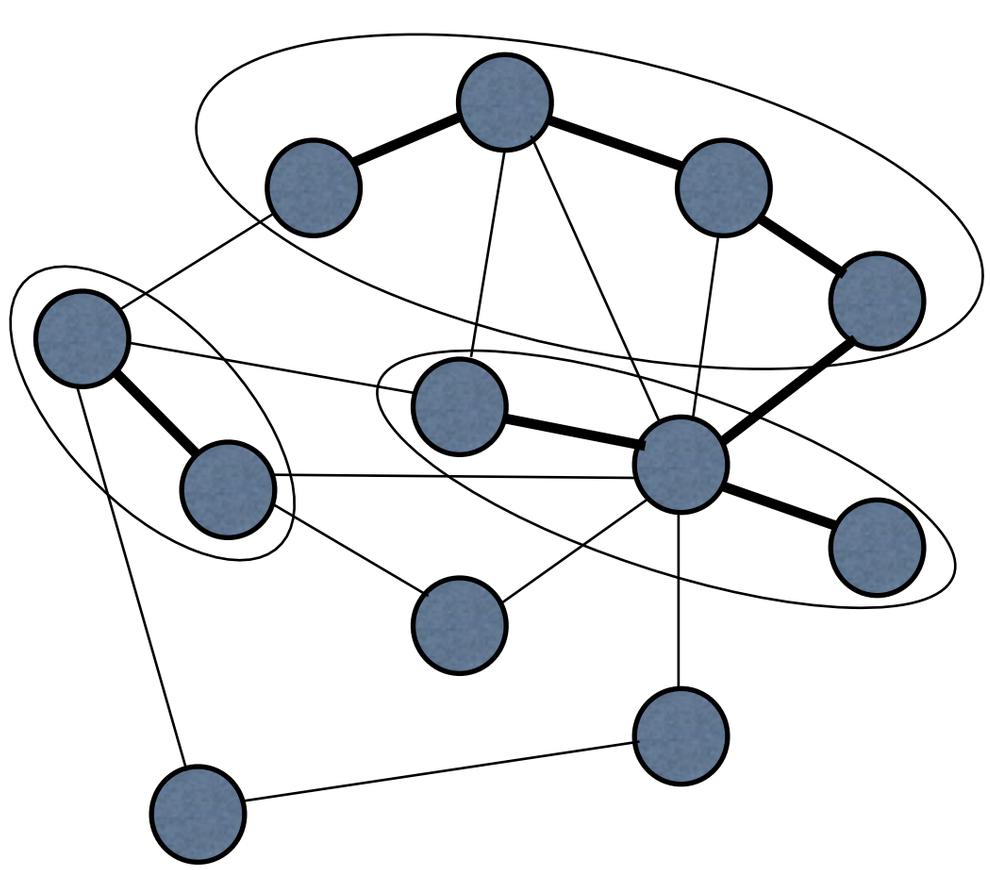


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

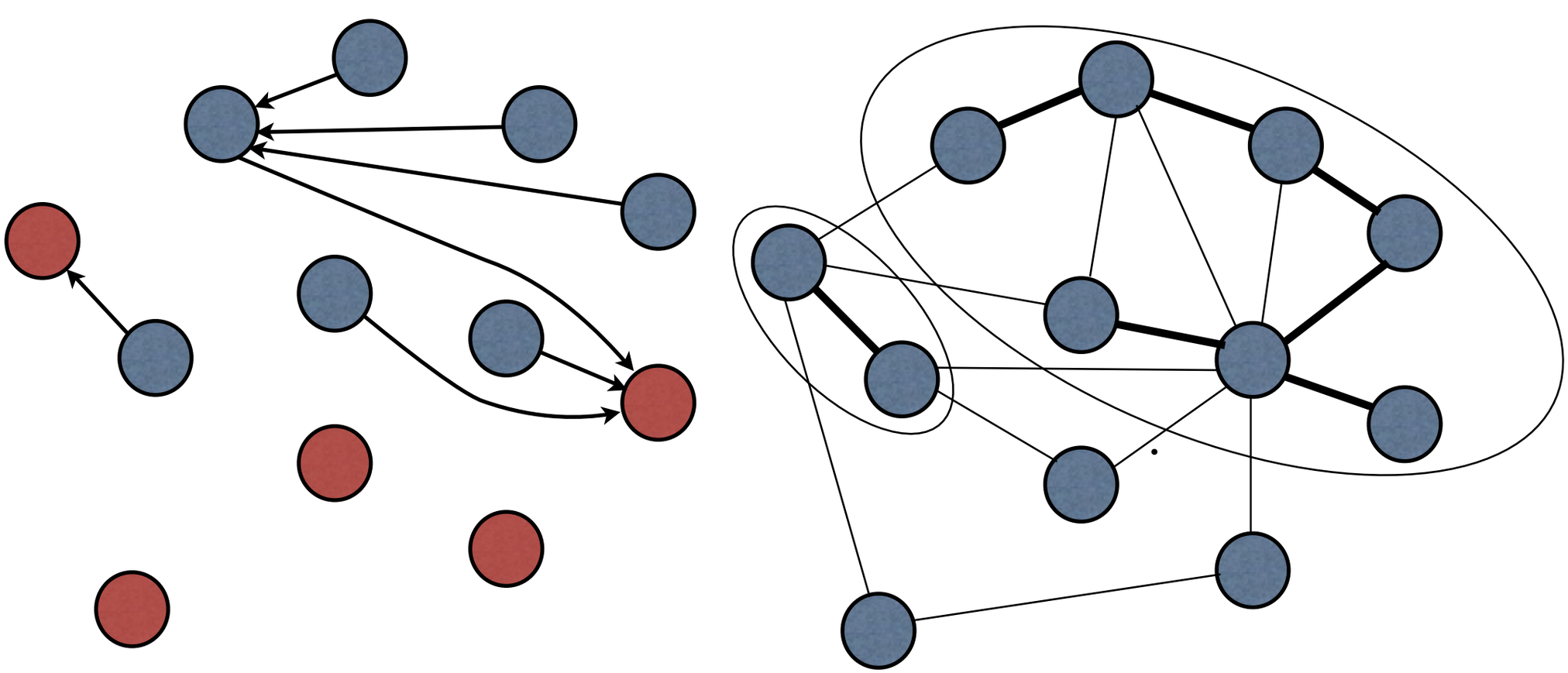




Halmazok egyesítése

halmazerdő felépítése

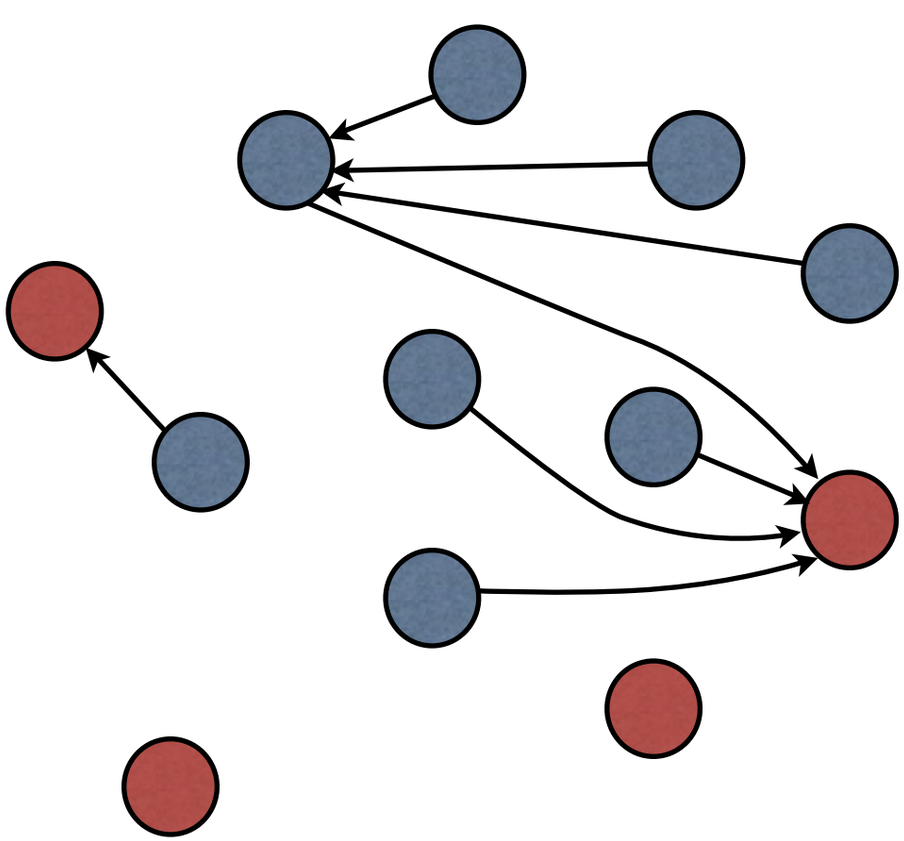
Kruskal algoritmus

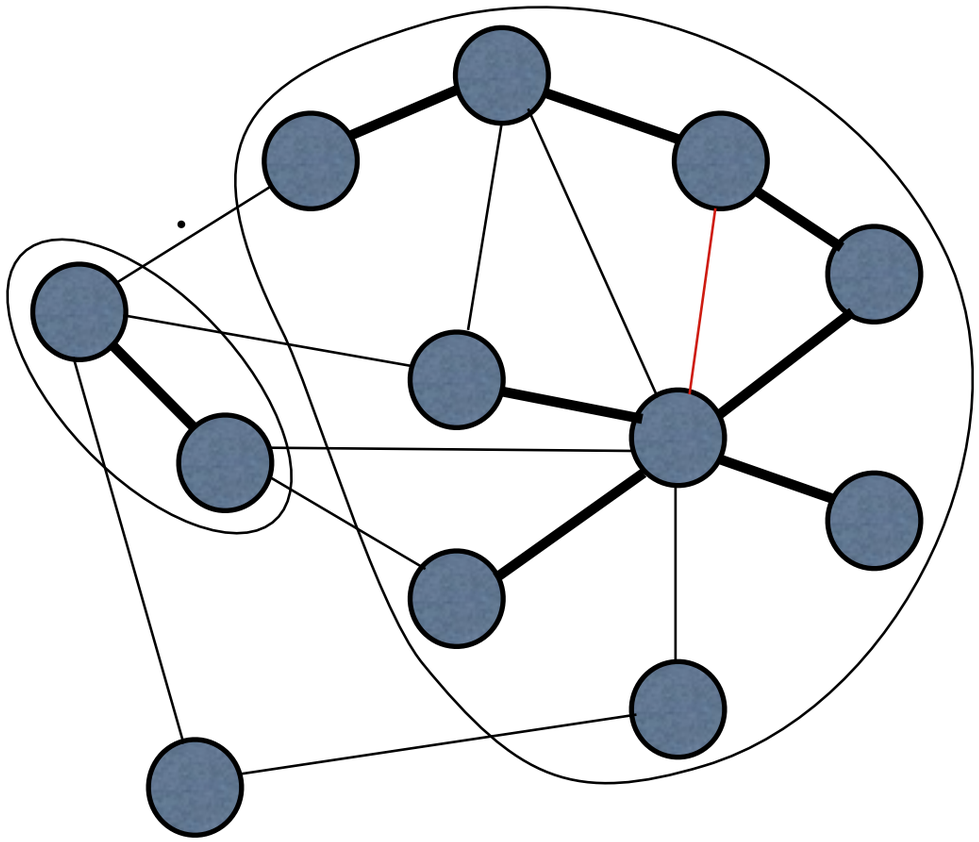


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

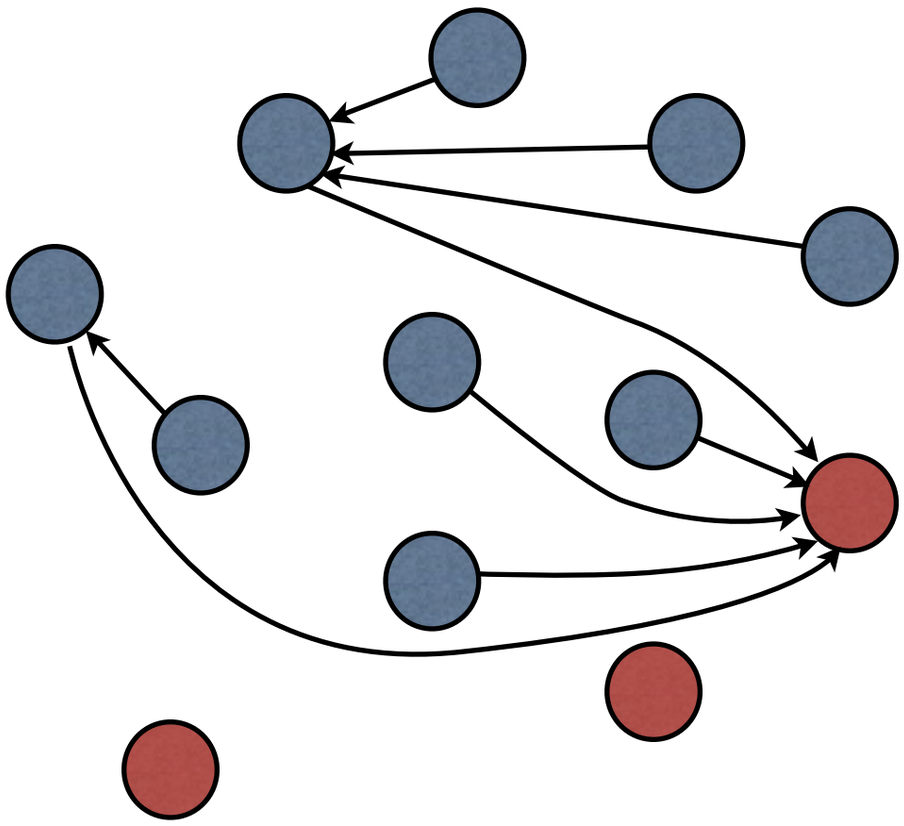


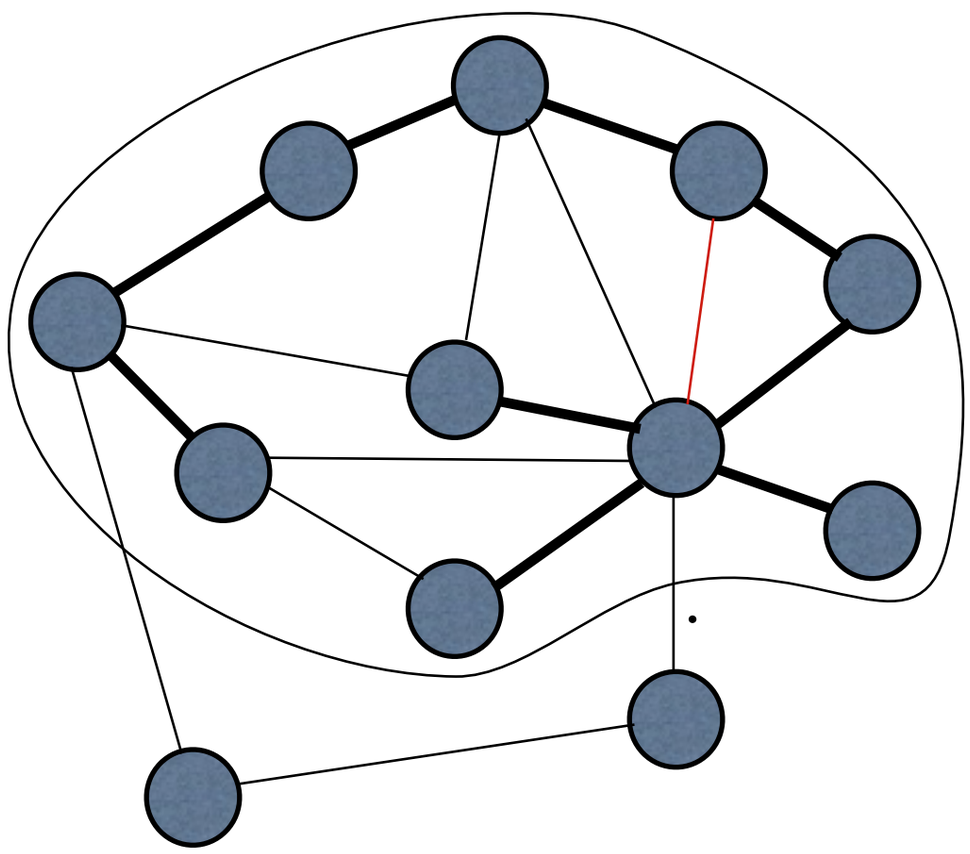


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

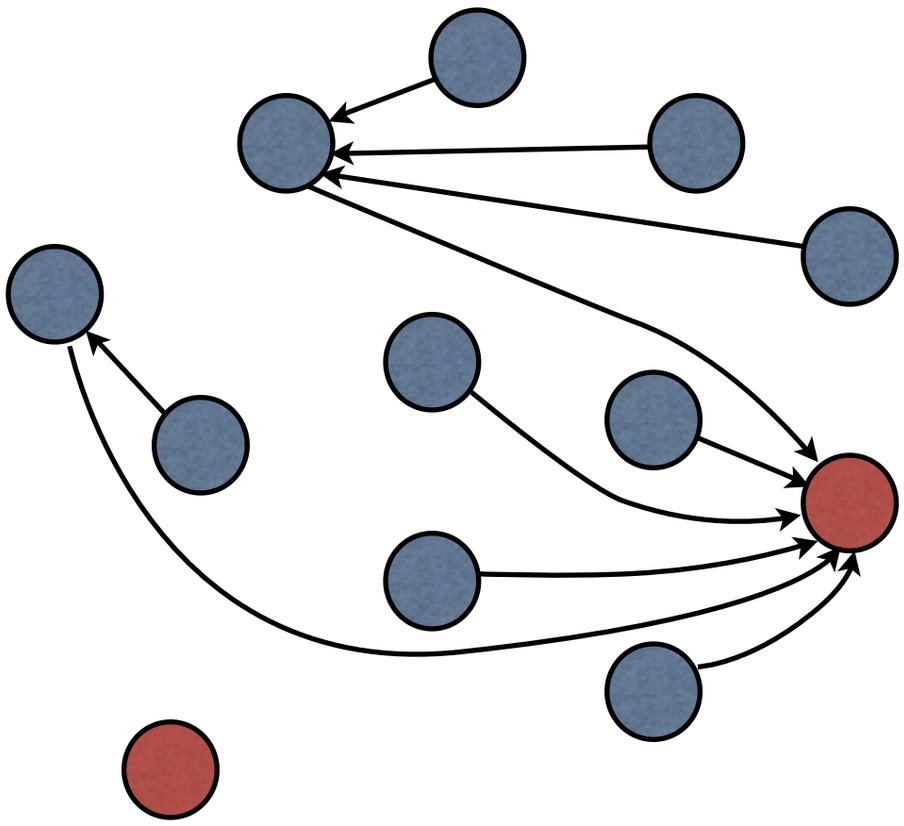


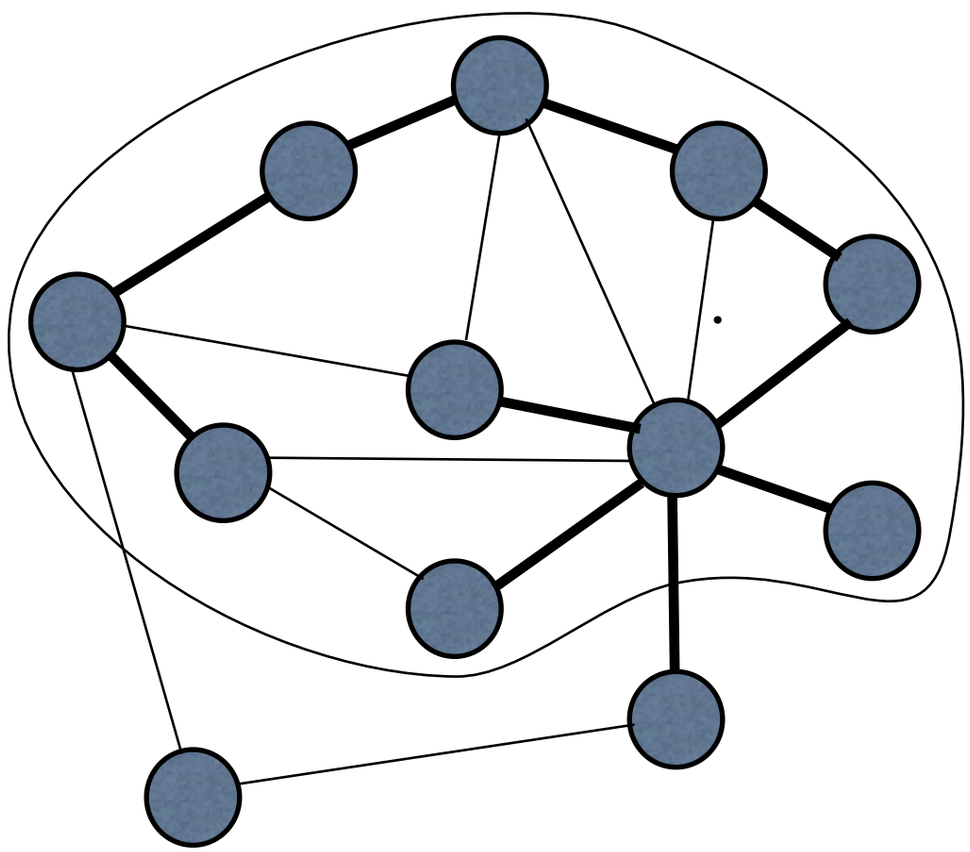


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

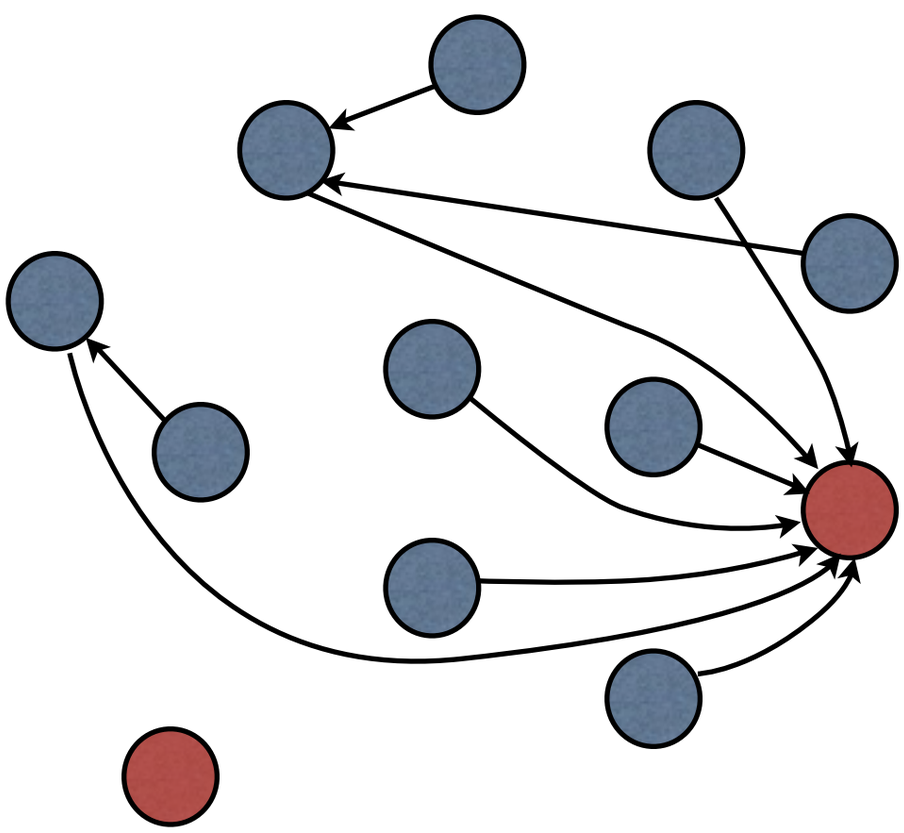


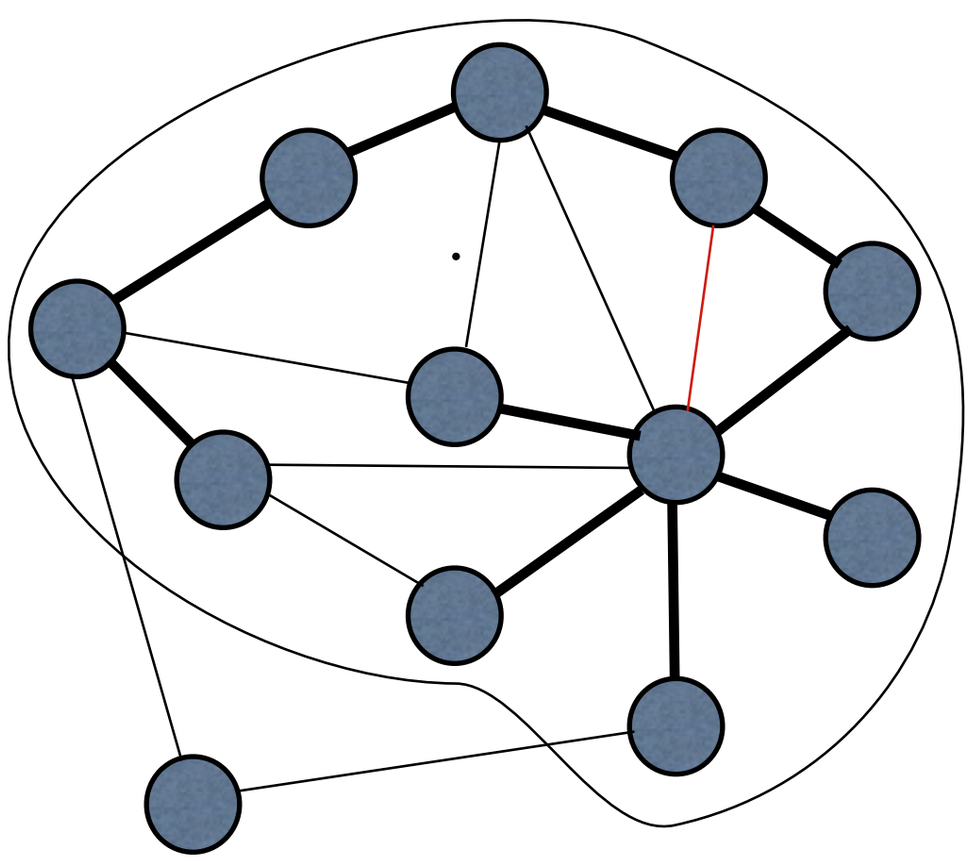


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

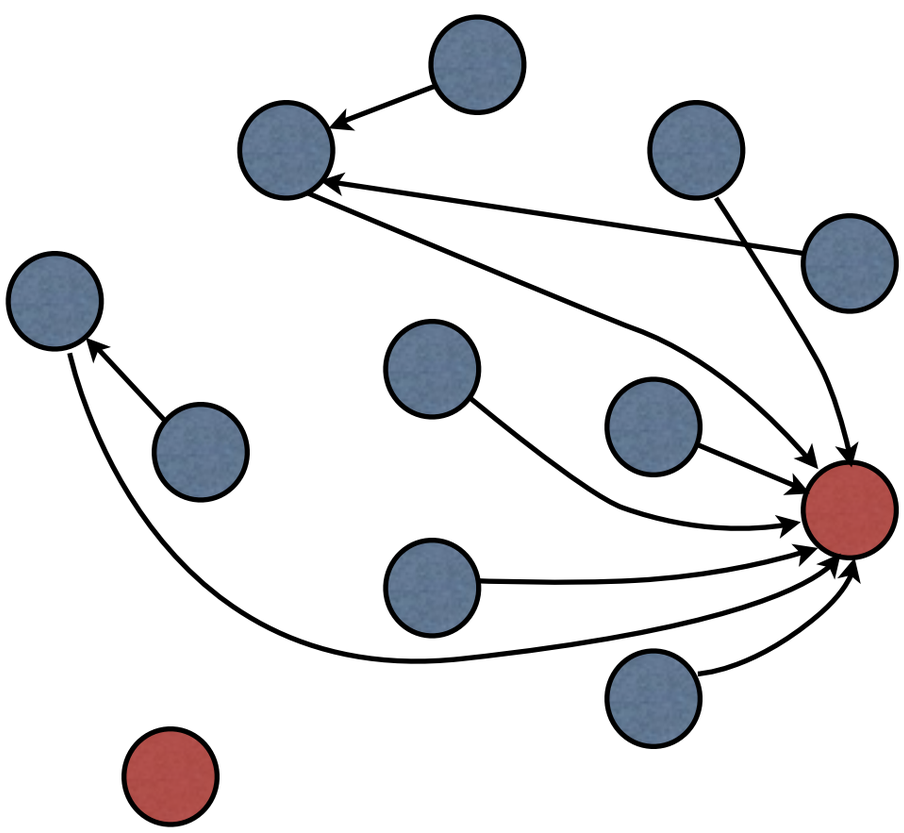


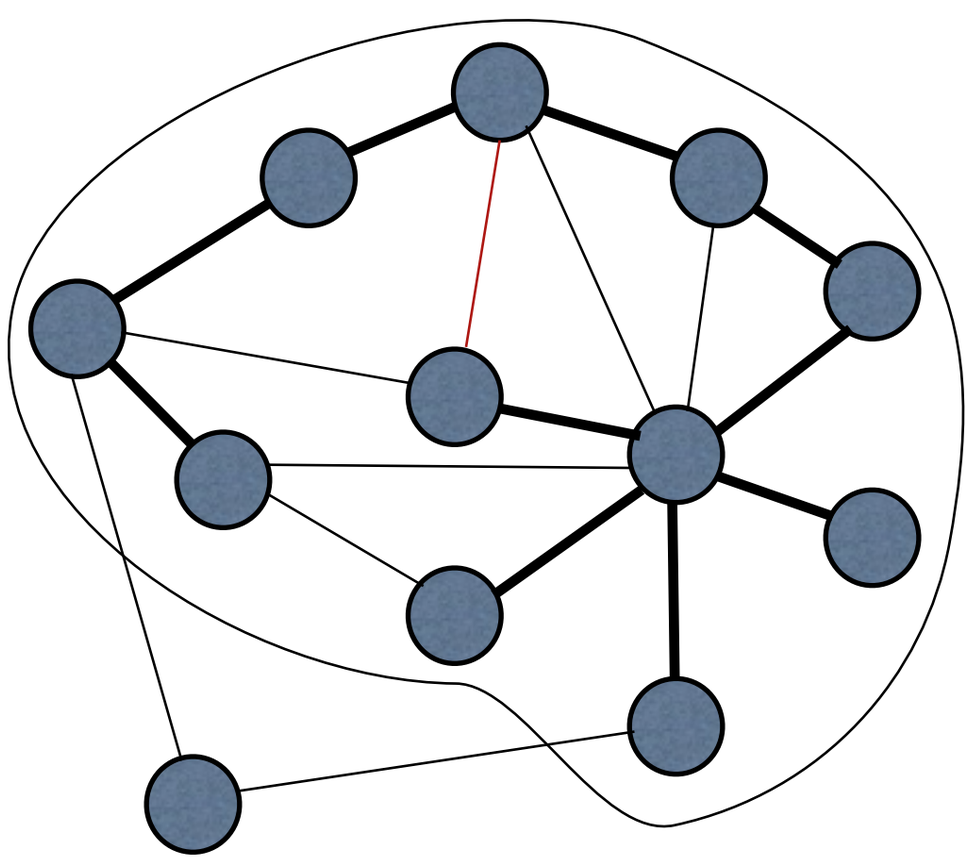


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

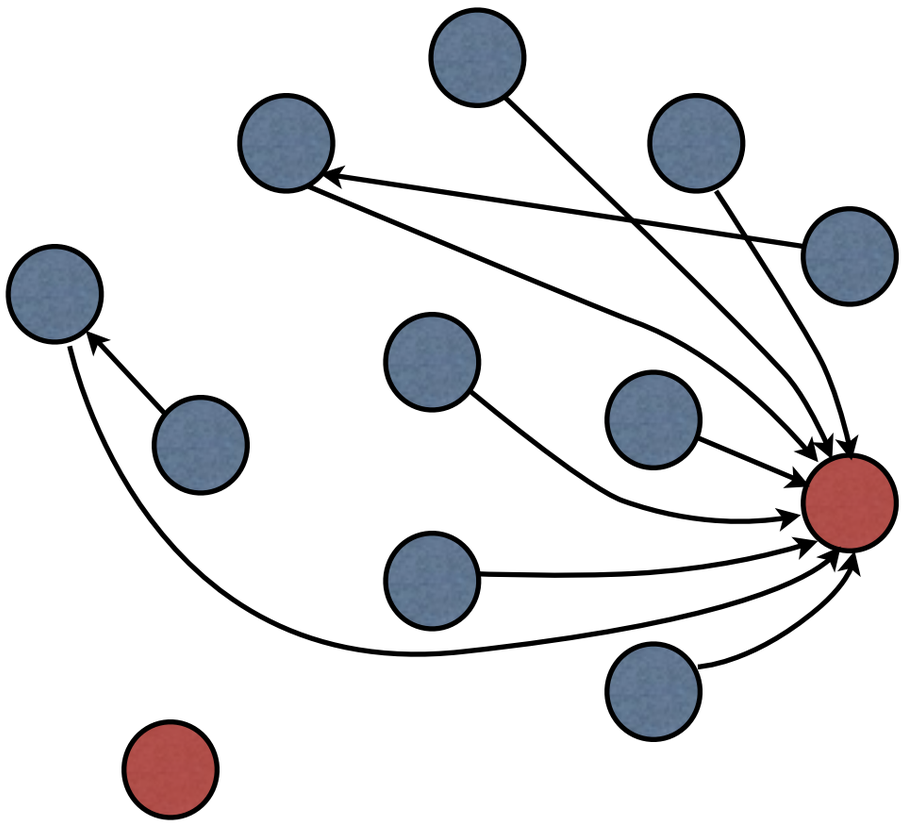


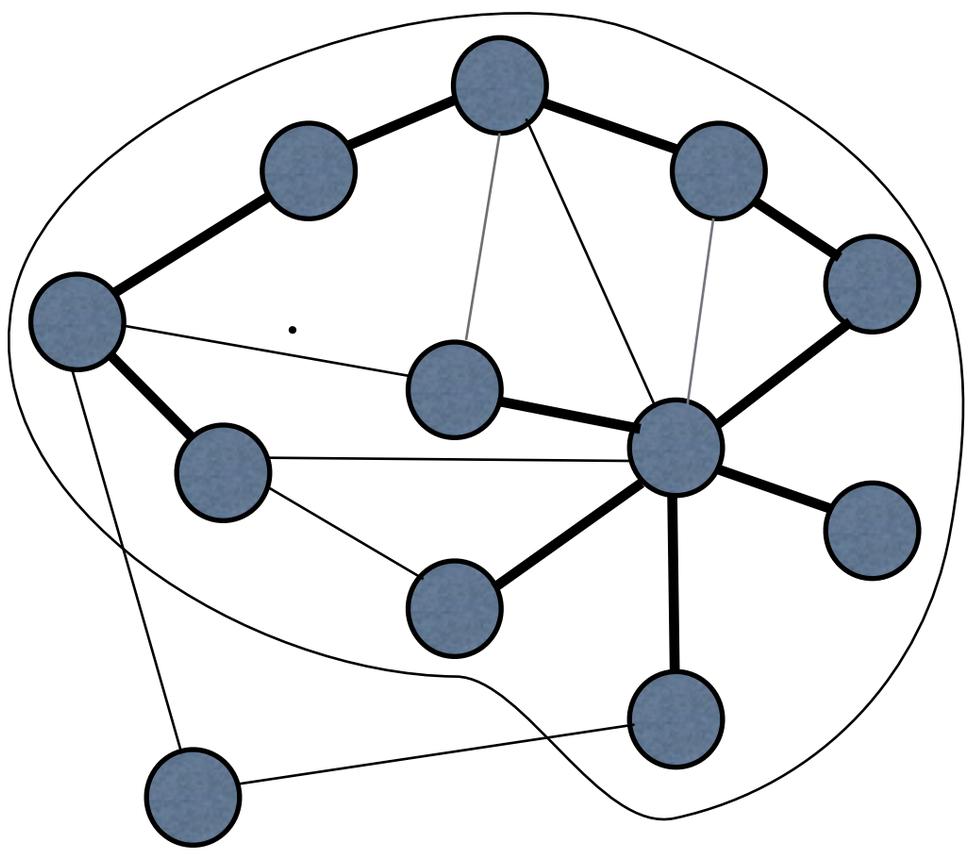


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

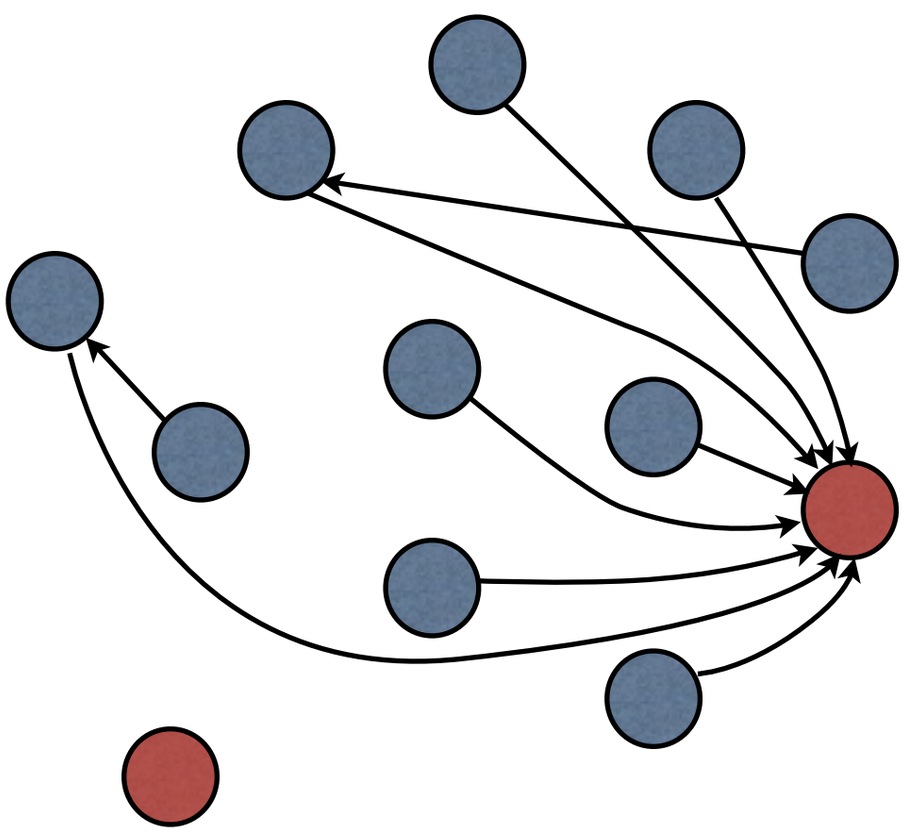


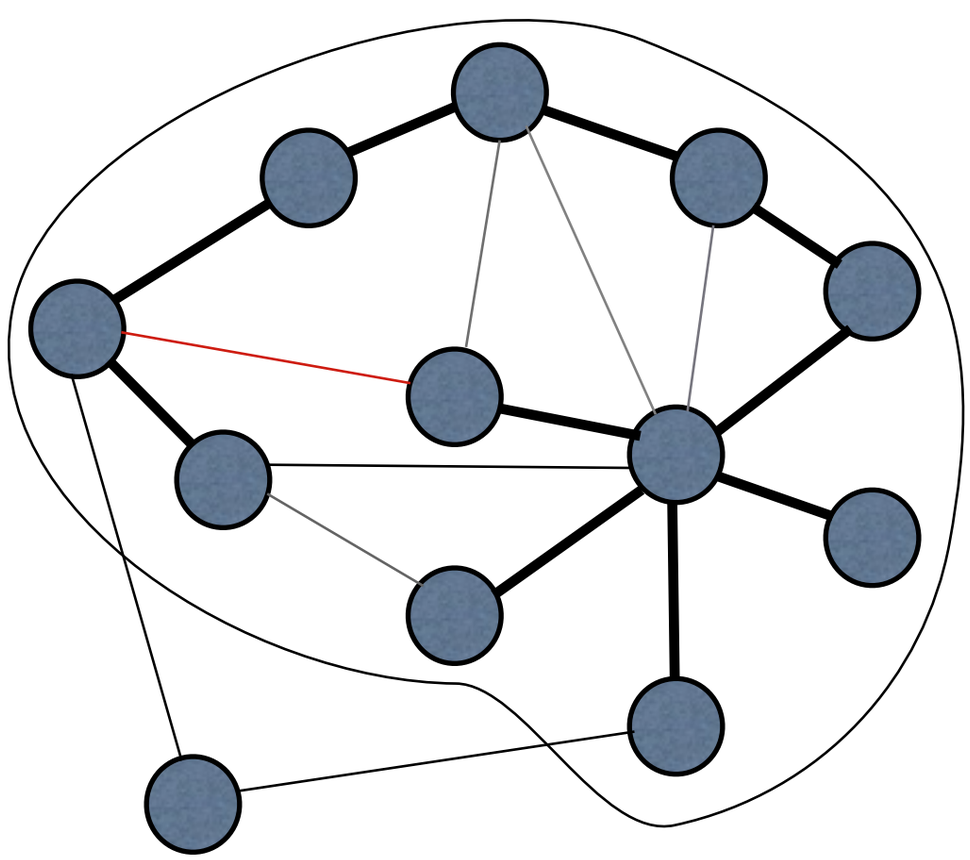


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

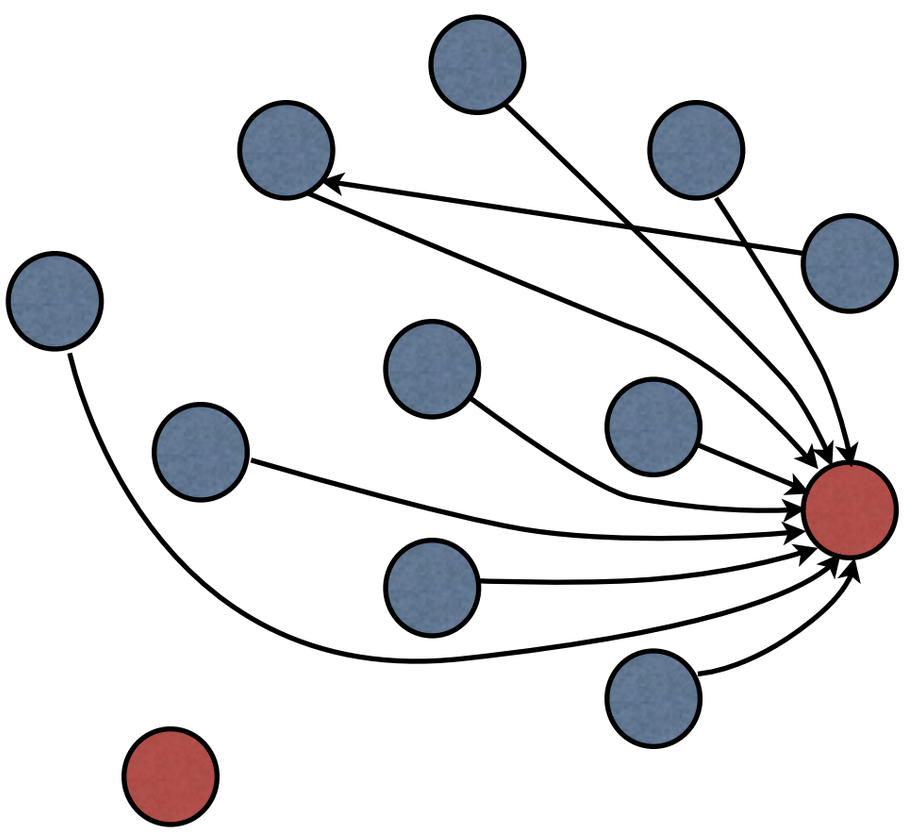


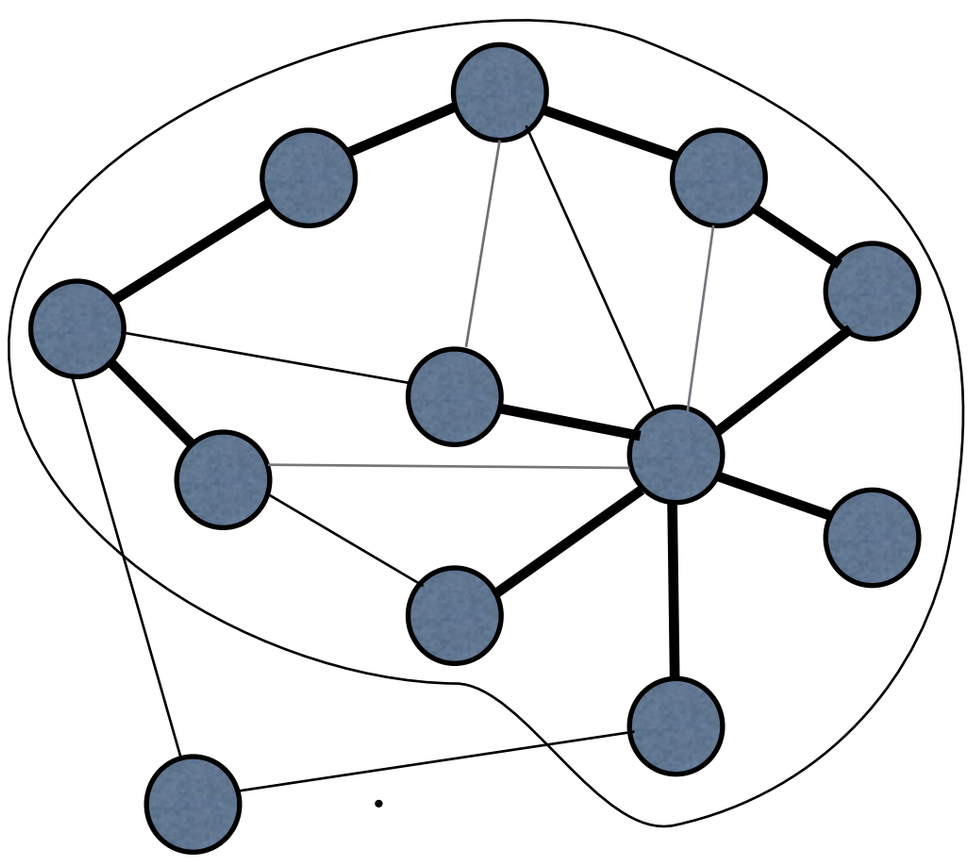


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus

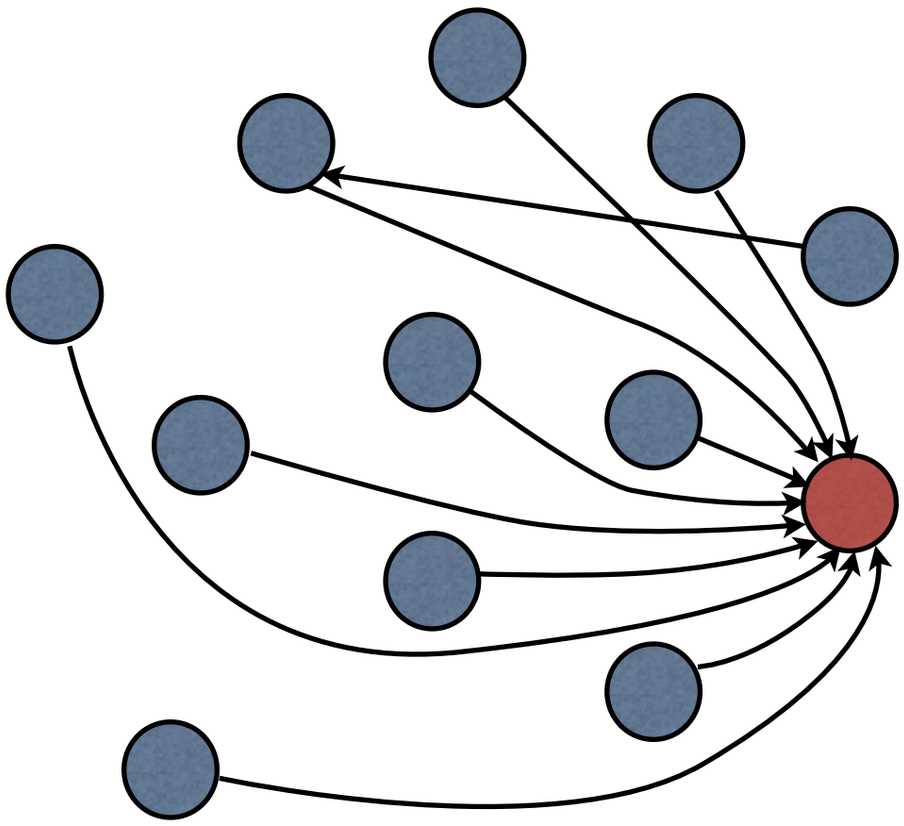


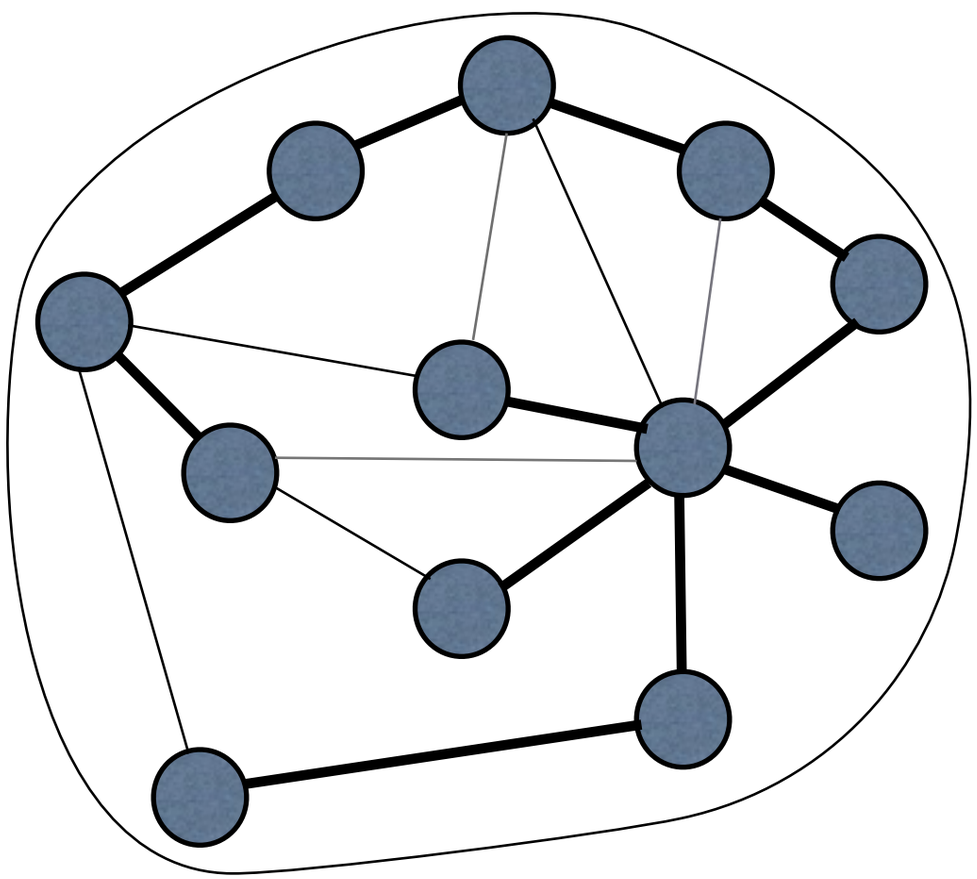


Halmazok egyesítése

halmazerdő felépítése

Kruskal algoritmus





Halmazerdő átlagos magassága

```
foreach (el from G) {
   rep1=rep(el.p) ;
   rep2=rep(el.q) ;
   if (rep1!=rep2) {
      hozzaad(el) ;
      unio(rep1,rep2) ;
   }
}
```

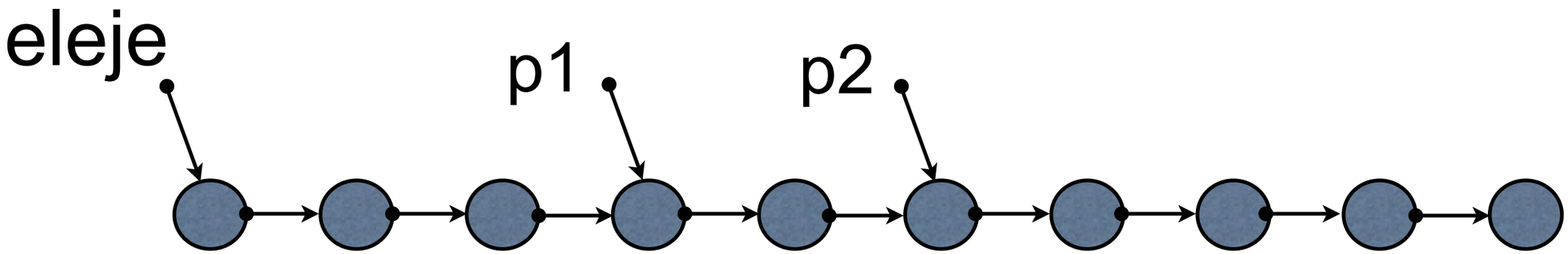
unio 1-el hosszabítja az utakat

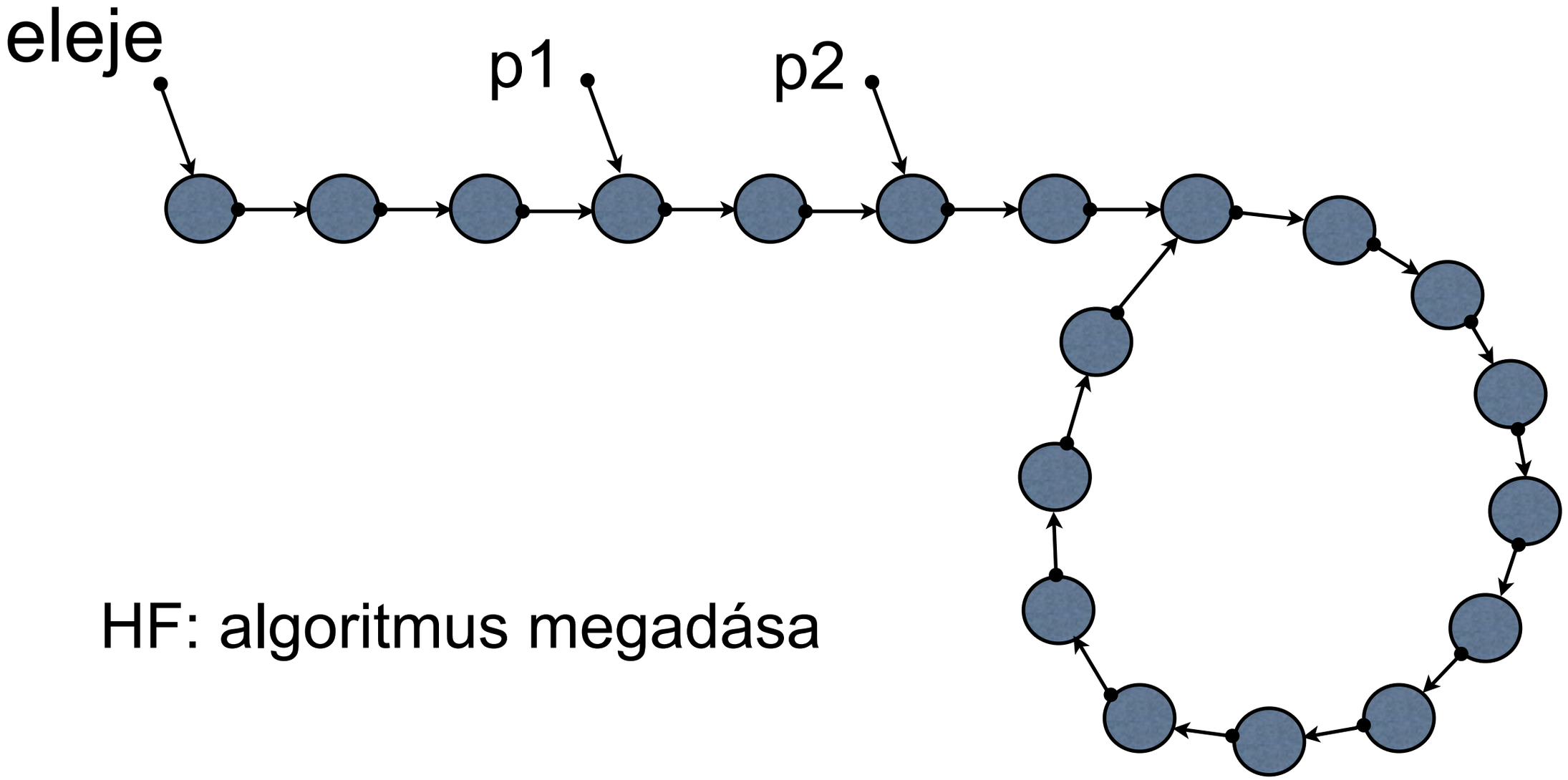
rep 1-hosszúra rövidíti az érintett pontokon az utakat

rep legalább 2-szer többször fut le mint unio

HF: halmazerdő átlagos magassága

Futási idő elemzése: **[2] 444-455**





HF: algoritmus megadása

Összefoglalás

absztrakt adatszerkezetek és műveleteik

verem és sor megvalósítása láncolt listával

halmaz, függvény absztrakt adatszerkezet

- bináris keresőfák, véletlen építésű bináris keresőfa, optimális bináris keresőfa
- rendezett-minta fák, intervallumfák
- AVL fák
- általános keresőfák, B-fák
- 2-3 fák, piros-fekete fák
- önszervező bináris keresőfák
- hasítótáblázatok
- ugrólisták

amortizációs költségelemzés

- összesítéses módszer
- potenciálmódszer

prioritási sor megvalósítása

- binomiális kupac, Fibonacci kupac

speciális (problémaspecifikus) adatszerkezetek

- halmazerdő

O(f(x))*	elem (kulcs) beszúrása	adott elem törlése**	adott kulcsú elem keresése	maximum keresés	rákövetkező elem	k. elem keresése	egyesítés	vágás**
rendezetlen tömb	1	1	n	n	n	?	n+m	n
kulcs-indexelt tömb	1	1	1	n	n	?	min(n,m)	n
rendezett tömb	n	n	log n	1	1	n	n+m	n
rendezetlen láncolt lista	1	1	n	n	n	?	1	n
rendezett láncolt lista	n	1	n	1	1	n	n+m	1
ugrólista	log n	log n	log n	1	1	n	n+m	1
ön-kiegyensúlyozó keresőfa	log n	log n	log n	log n	1, log n	log n	(n+m) log n	n log n
binomiális kupac	log n	log n	n	1	n	?	log n	log n
hasító tábla	1	1	1	n	n	?	n+m	n

* amortizációs költség
** keresés nélkül