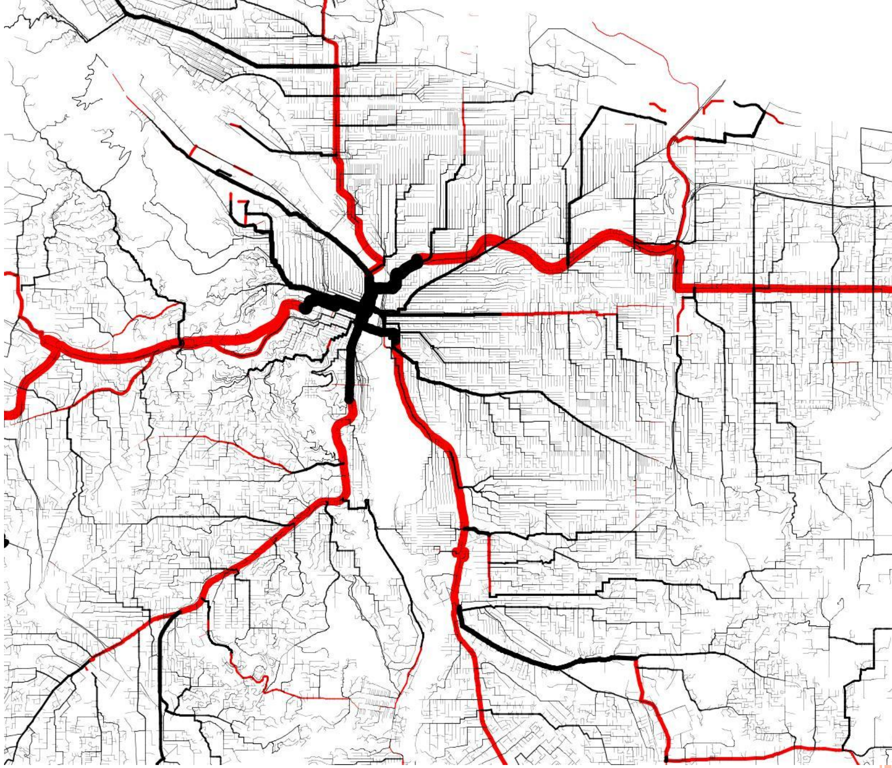
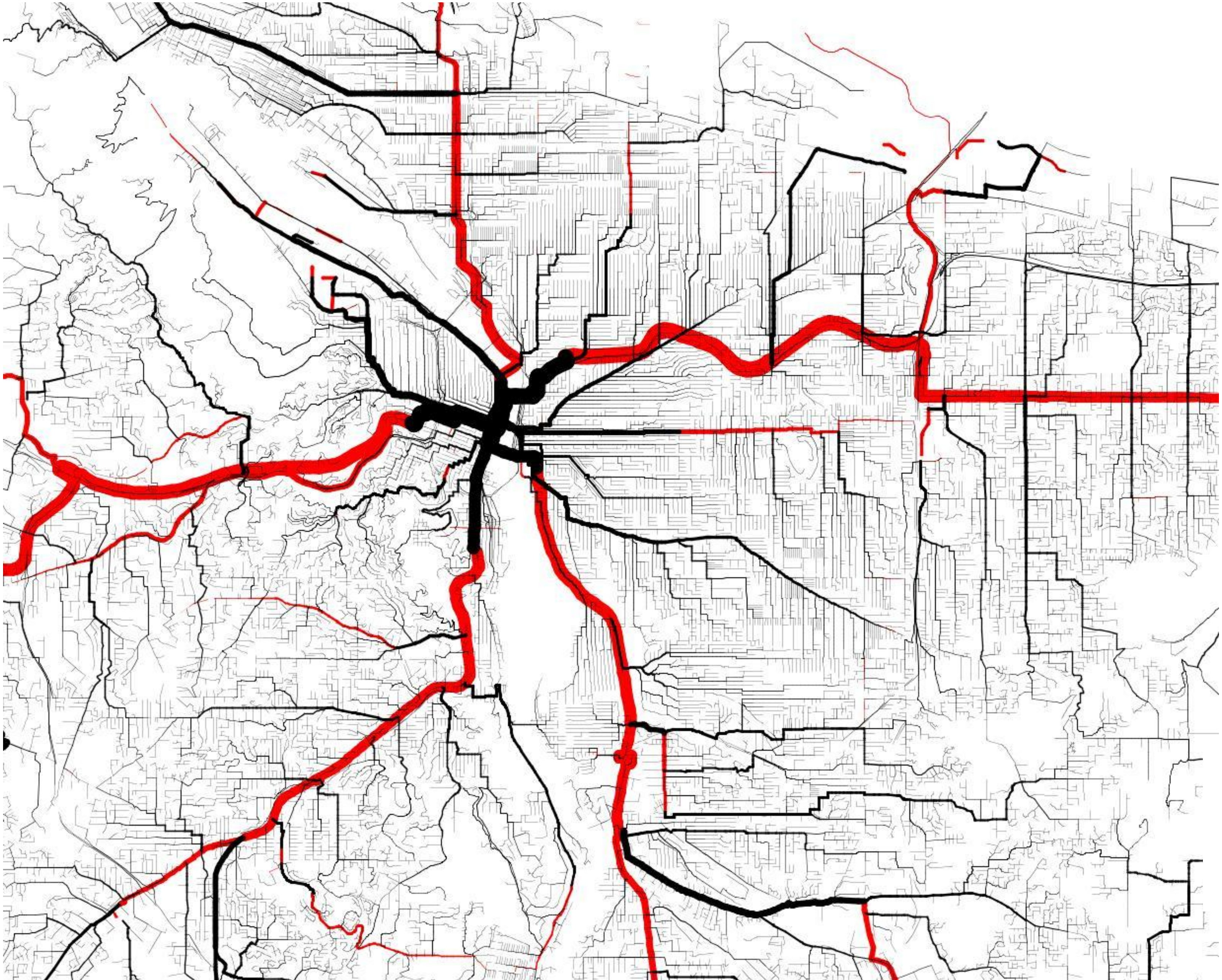


Minimalno vpeto drevo

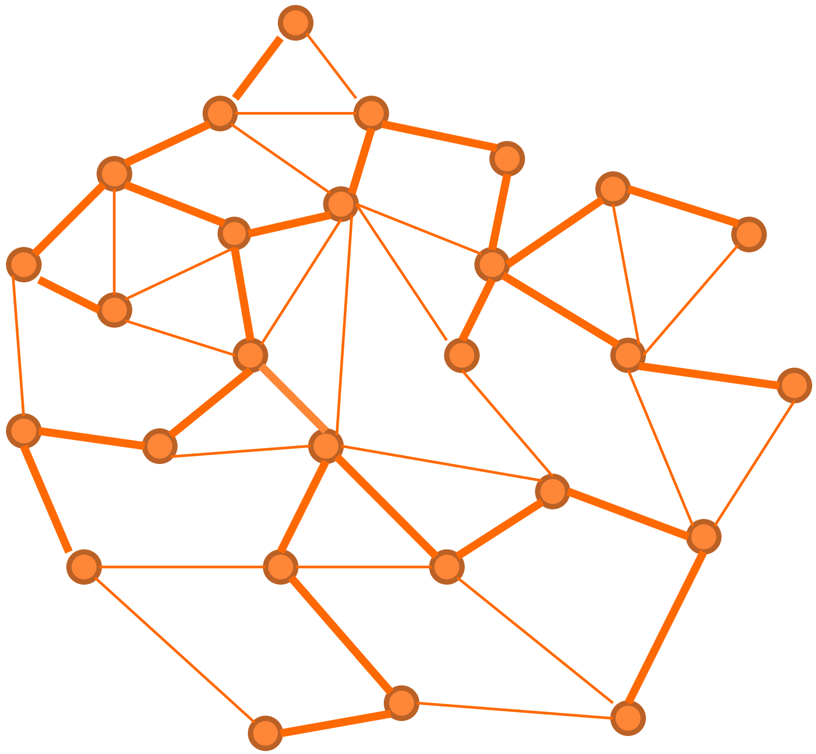
(minimum spanning tree)

- Primov algoritem
- Kruskalov algoritem

PROMET, ENERGIJA, KOMUNIKACIJE

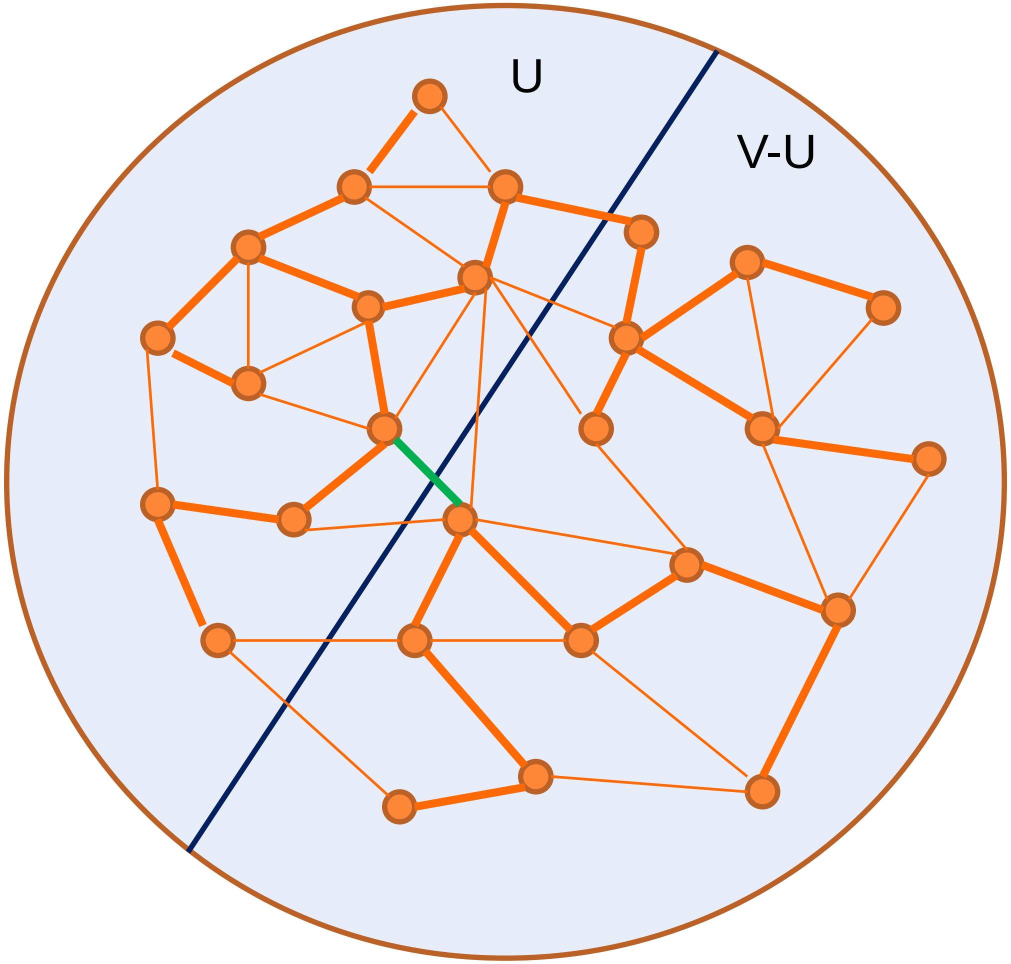


V MST JE KOREN KATEROKOLI VOZLIŠČE



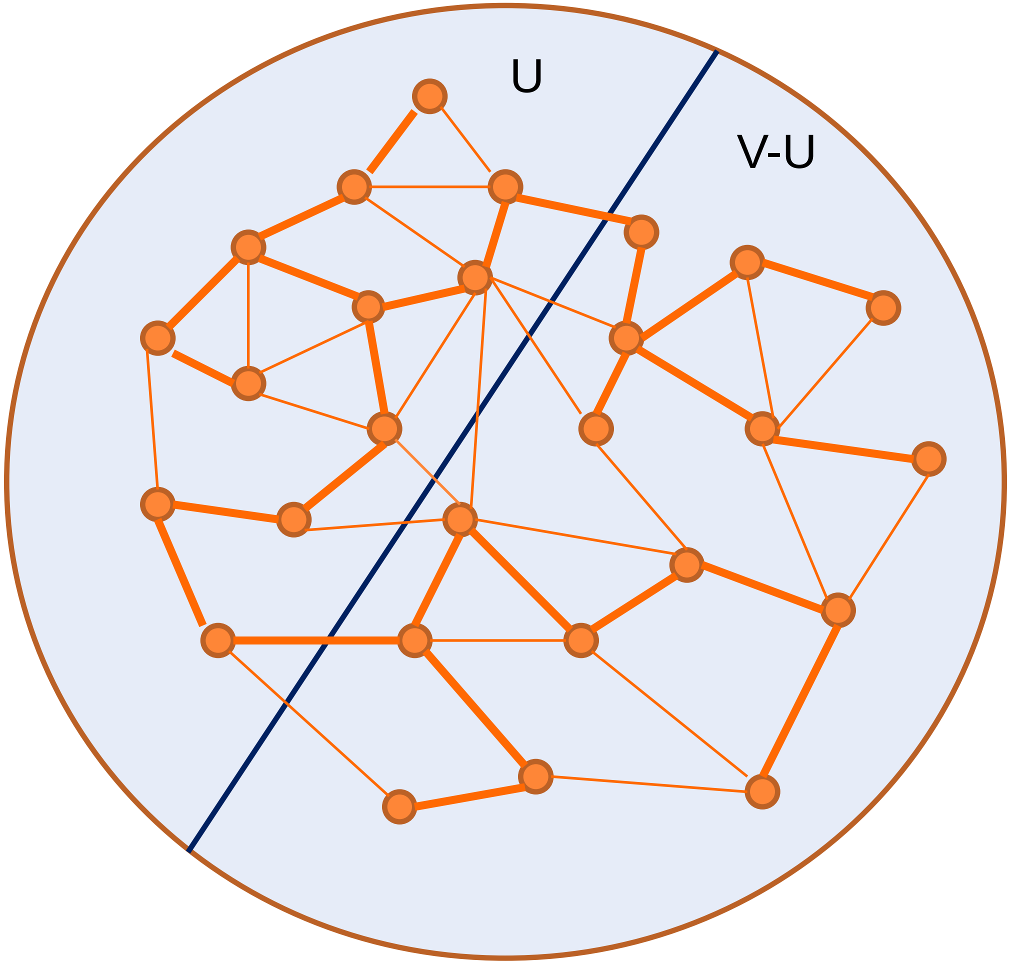
Lastnost neusmerjenega grafa

Najkrajša povezava med dvema poljubnima podmnožicama U in V-U je v MST.



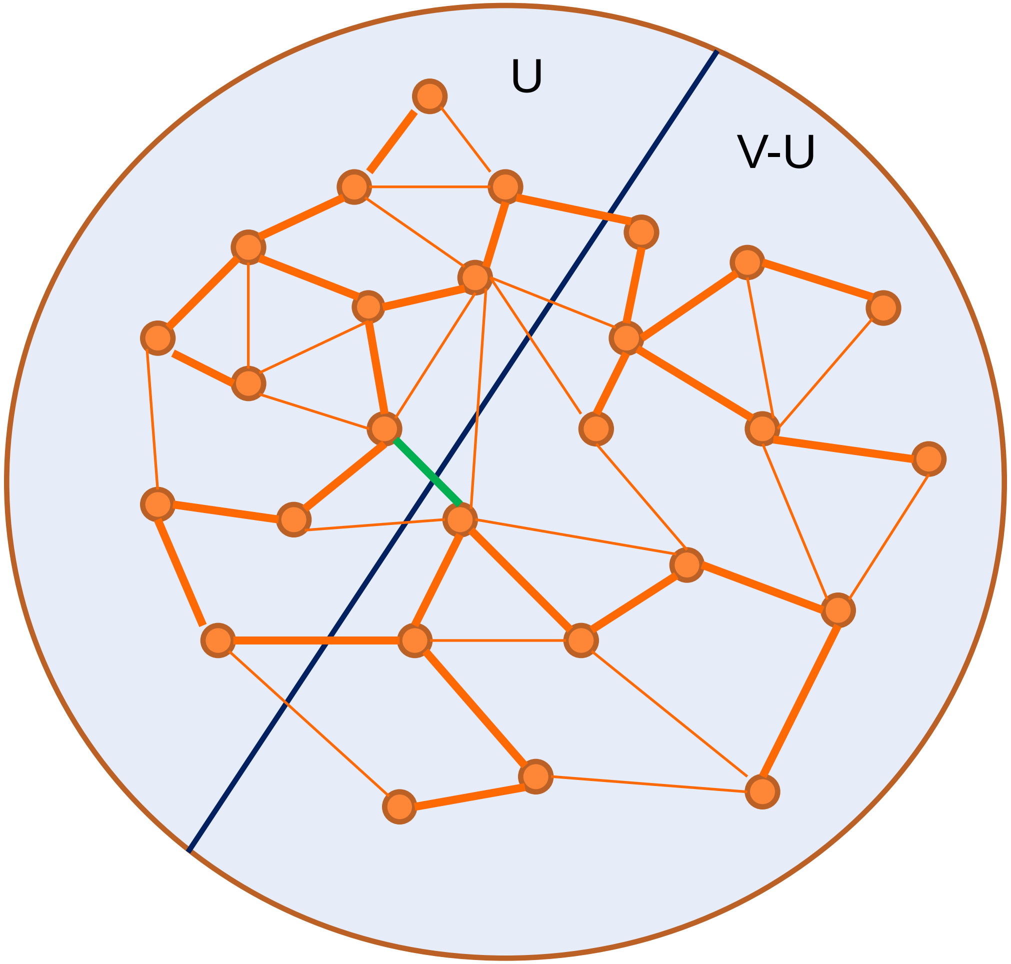
LASTNOST NEUSMERJENEGA GRAFA

Recimo, da ni. Torej je namesto nje daljša povezava med U in V-U.



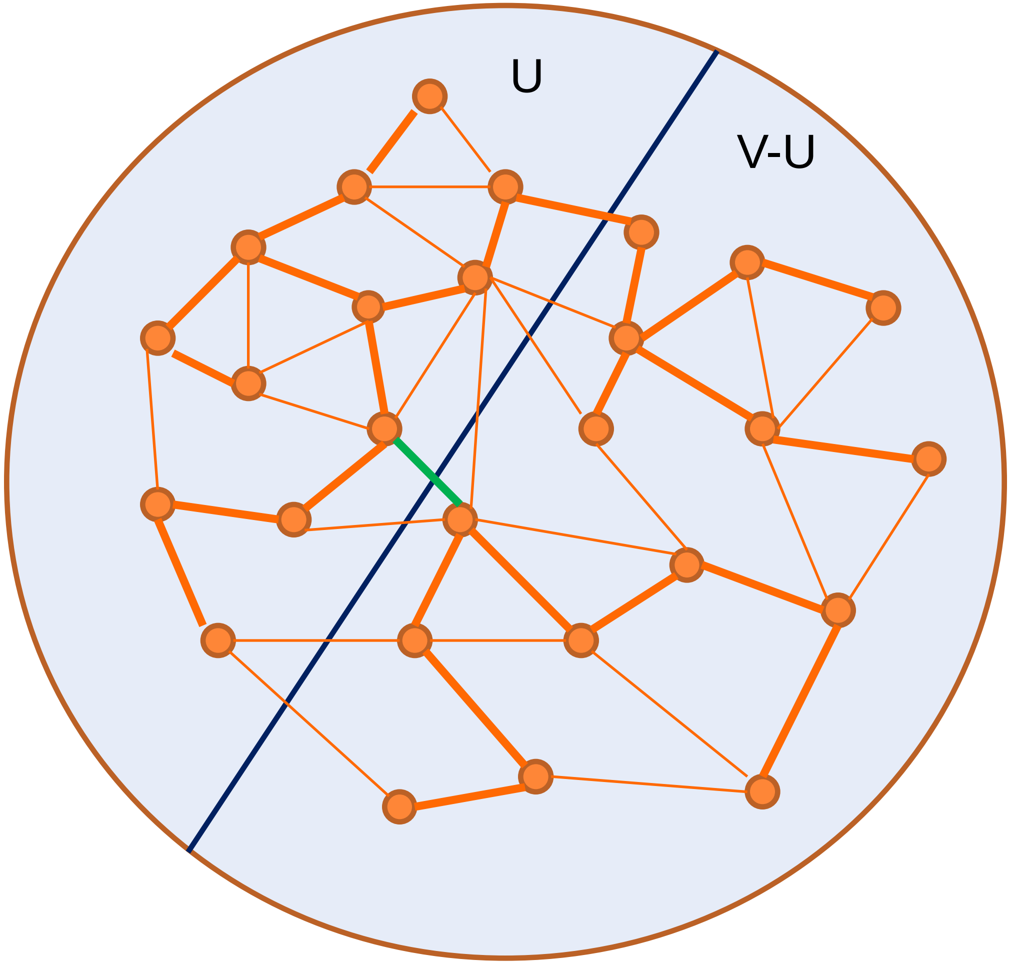
Lastnost neusmerjenega grafa

Dodajmo najkrajšo in dobimo cikel.



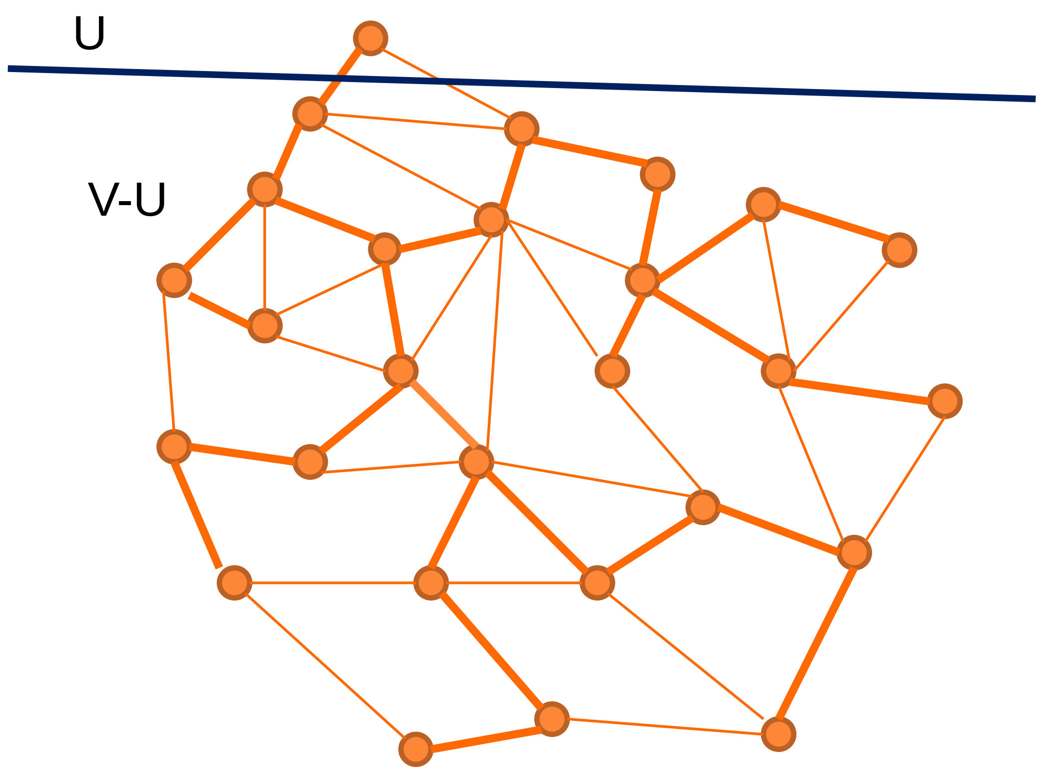
Lastnost neusmerjenega grafa

Cikla se znebimo tako, da zbrišemo daljšo → dobili smo manjši MST → protislovje!



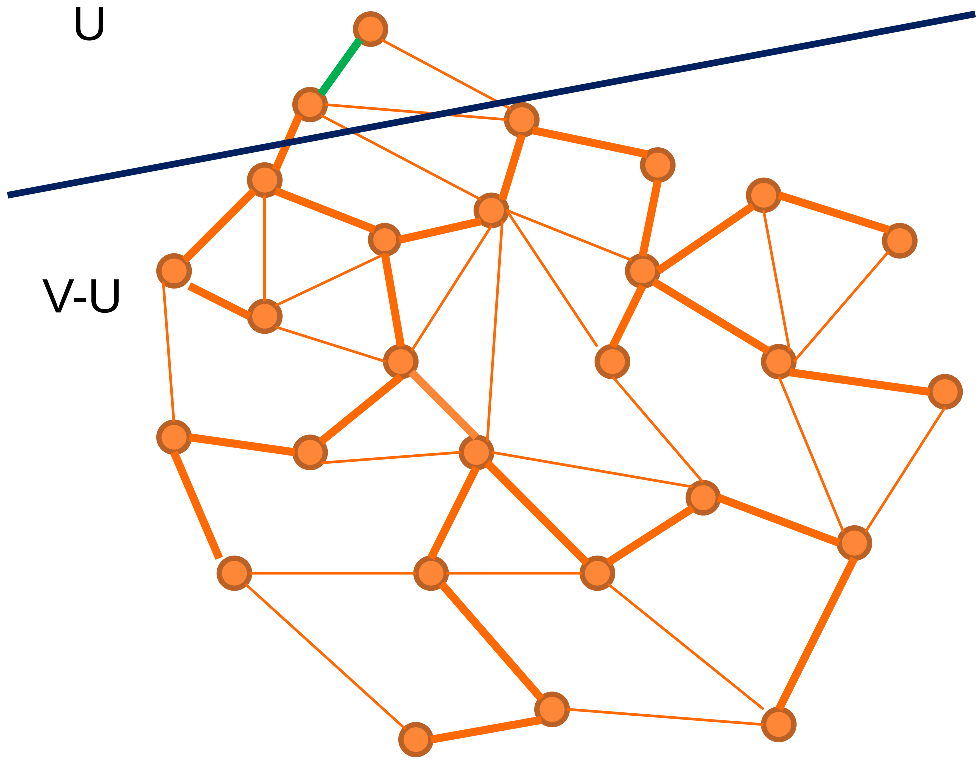
Primov algoritem

- Podmnožica U vsebuje na začetku eno samo (poljubno) vozlišče.
- V enem koraku dodamo najkrajšo povezavo med U in V-U.



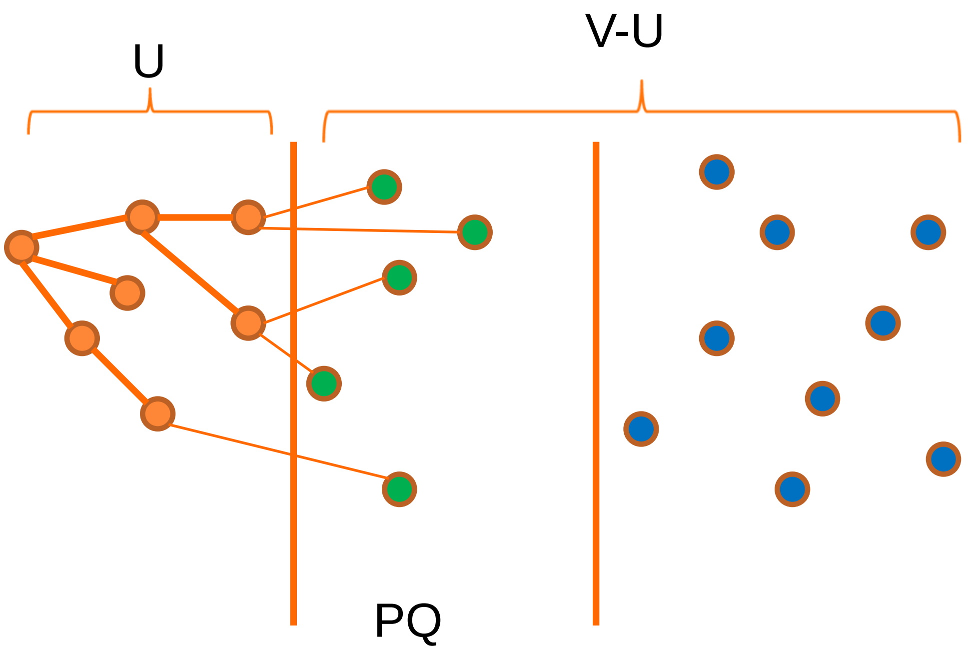
Primov algoritem

- Podmnožica U vsebuje na začetku eno samo (poljubno) vozlišče.
- V enem koraku dodamo najkrajšo povezavo med U in V-U.



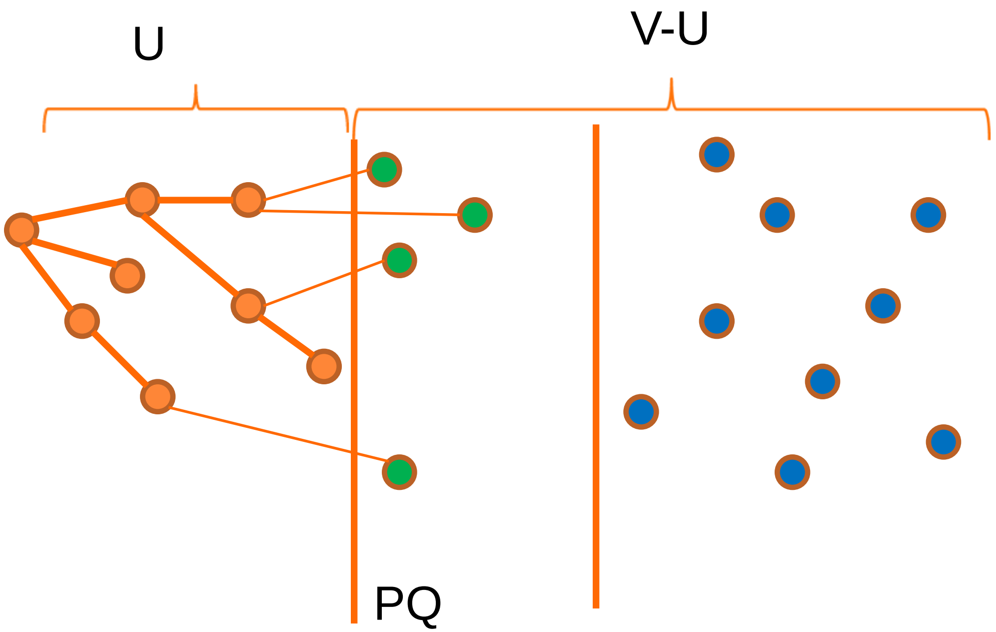
PRIMOV ALGORITEM - IDEJA

- Primov algoritem je **požrešen** in zelo podoben algoritmu Dijkstra (le da je graf neusmerjen)
- gradimo MSTod poljubnega začetnega vozlišča
- vsakič iz množice vozlišč, ki še niso v drevesu, izberemo tisto z **najkrajšo povezavo** od nekega vozlišča v MST



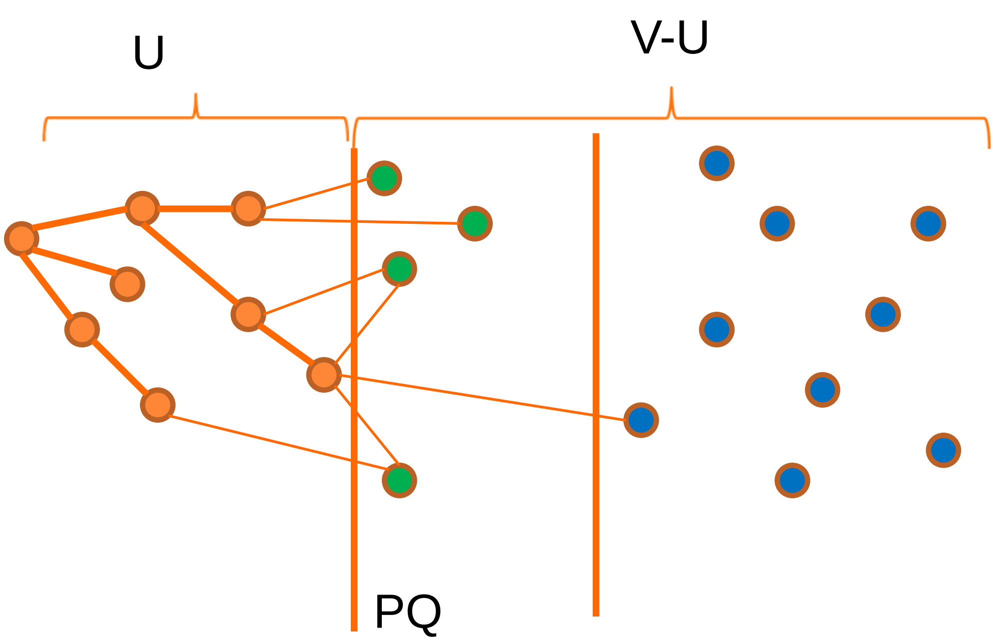
Primov algoritem - ideja

- Primov algoritem je **požrešen** in zelo podoben algoritmu Dijkstra (le da je graf neusmerjen)
- gradimo MSTod poljubnega začetnega vozlišča
- vsakič iz množice vozlišč, ki še niso v drevesu, izberemo tisto z **najkrajšo povezavo** od nekega vozlišča v MST



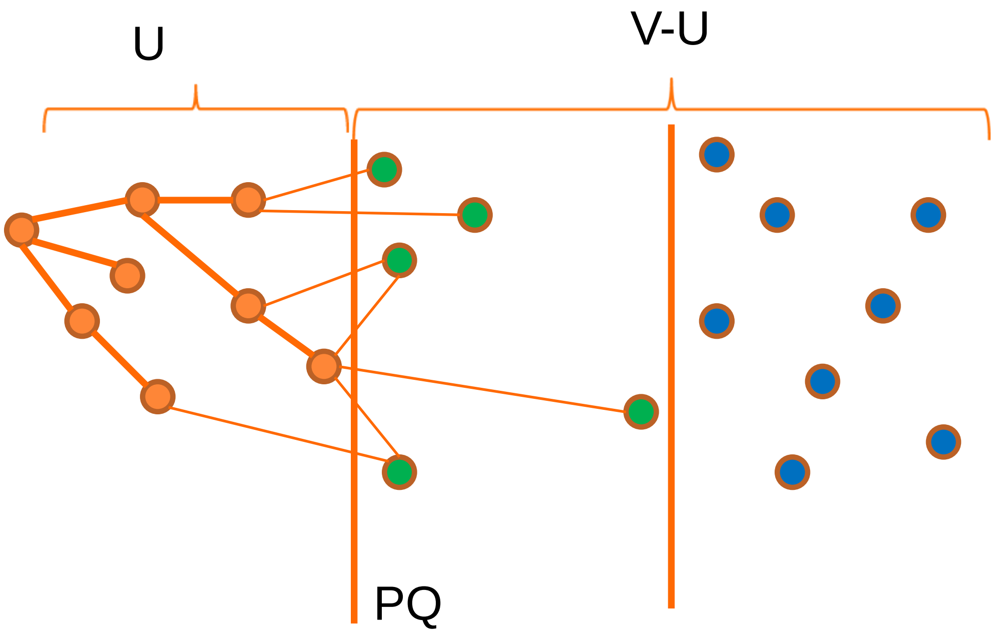
PRIMOV ALGORITEM - IDEJA

- gradimo MSTod poljubnega začetnega vozlišča
- vsakič iz množice vozlišč, ki še niso v drevesu, izberemo tisto z **najkrajšo povezavo** od nekega vozlišča v MST
- zatem pogledamo sosede dodanega vozlišča:
    1. če je sosed že v MST, ga ignoriramo
    2. če je sosed že v prioritetni vrsti, mu posodobimo prioriteto
    3. sicer ga dodamo v prioritetno vrsto



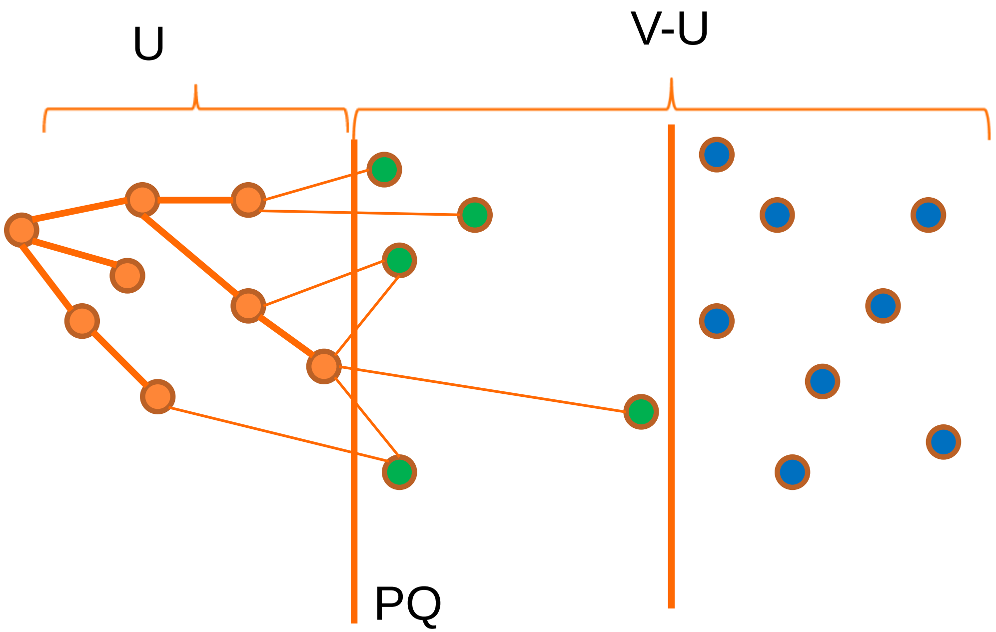
Primov algoritem - ideja

- gradimo MSTod poljubnega začetnega vozlišča
- vsakič iz množice vozlišč, ki še niso v drevesu, izberemo tisto z **najkrajšo povezavo** od nekega vozlišča v MST
- zatem pogledamo sosede dodanega vozlišča:
  1. če je sosed že v MST, ga ignoriramo
  2. če je sosed že v prioritetni vrsti, mu posodobimo prioriteto
  3. sicer ga dodamo v prioritetno vrsto



Primov algoritem - implementacija

- za izbiro vozlišča *v* z najkrajšo razdaljo uporablja algoritem prioritetno vrsto vozlišč, za katera je že znana dolžina povezave od nekega vozlišča v MST
- v prioritetni vrsti se hranijo dolžine **najkrajših povezav** za vsako vozlišče
- **z napredovanjem algoritma se te povezave lahko skrajšajo,** zato je potrebno uvesti še operacijo zmanjšanja prioritete



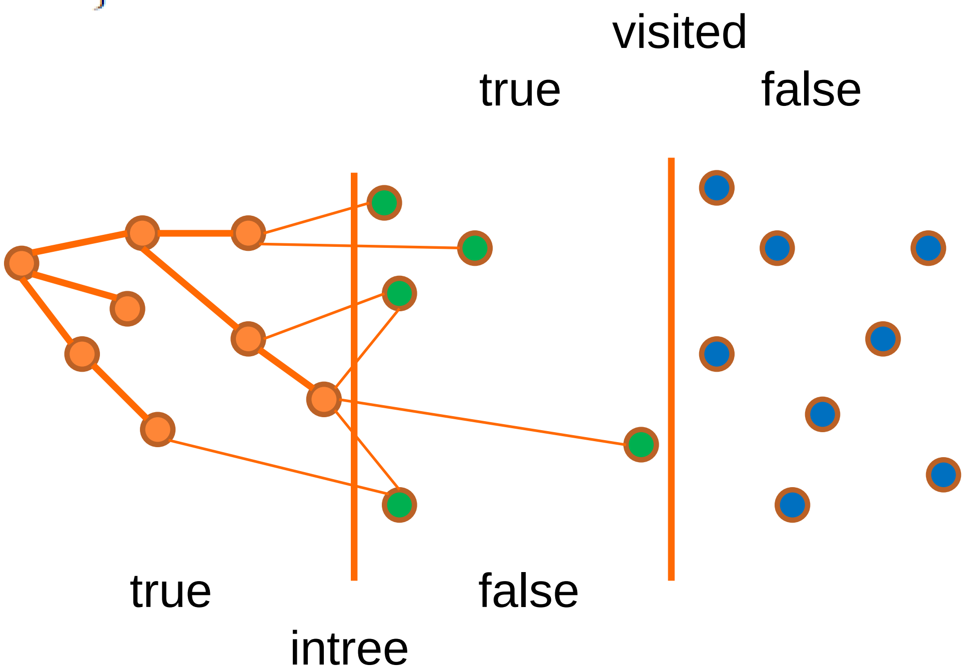
ZMANJŠANJE PRIORITETE ELEMENTA V VRSTI

- DECREASE_KEY(x, New, Q)
- zmanjša prioriteto elementa *x* na New
- v kopici operacijo implementiramo tako, da element z zmanjšano prioriteto zamenjujemo z očetom
- postopek se ustavi, bodisi če je oče manjši od elementa ali če element pride v koren kopice
- časovna zahtevnost je reda *O*(log n) pod pogojem, da imamo direkten dostop do elementa v kopici
- **vsako vozlišče hrani svoj položaj (indeks) v kopici**

Primov algoritem - implementacija

```
class PrimVertex extends VertexAdj implements HeapPosNode {
    boolean visited;
    boolean intree ;
    PrimVertex parent;
    double distance ;
    int heapIndex ;
 } // class PrimVertex
```

rezultat algoritma



Primov algoritem - implementacija

```
public void prim(UGraph g) {
    PQDecrease q = new HeapPos() ; // urejena po distance
    PrimVertex v, w ;
    Edge e;

  // nobeno vozlisce se ni bilo pregledano
  for (PrimVertex t = (PrimVertex)g.firstVertex(); t!= null;
            t = (PrimVertex)g.nextVertex(t))
        t.visited = false;
  //inicializiraj prvo vozlisce in ga dodaj v priorit. vrsto
  v = (PrimVertex)g.firstVertex();
  v.visited = true;
  v.parent = null;
  v.intree = false;
  v.distance = 0;
  q.insert(v);
```

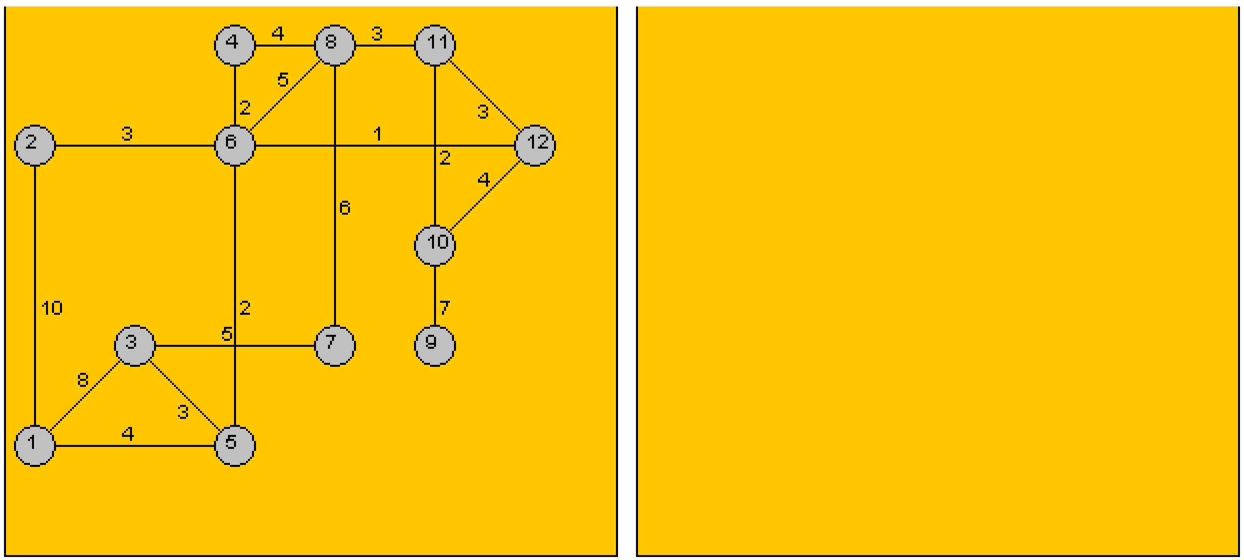
```
  while ( !q.empty() ) {
    v = (PrimVertex)q.deleteMin();
    v.intree = true;
    e = g.firstEdge(v);
    while (e != null) {
      w = (PrimVertex)g.adjacentPoint(e, v);
      if (! w.visited) {
          w.visited = true; // je v kopici
          w.intree = false; // se ni v drevesu
          w.parent = v; // potencialni oce
          // trenutna najkrajsa povezava do drevesa:
          w.distance = ((Double)e.evalue).doubleValue();
          q.insert(w) ;
        }
       else // visited
          if (! w.intree && ((Double)e.evalue).doubleValue() <
           w.distance) { //nasli smo krajso povezavo do drevesa
              w.parent = v; // novi potencialni oce
              // popravi razdaljo v prioritetni vrsti:
              q.decreaseKey(w, e.evalue) ;
          }
          e = g.nextEdge(v, e) ;
    } // while e
  } // while !empty
} // Prim
```

```
while ( !q.empty() ) {
  v = (PrimVertex)q.deleteMin();
  v.intree = true;
  e = g.firstEdge(v);
  while (e != null) {
    w = (PrimVertex)g.adjacentPoint(e, v);
    if (! w.visited) {
        w.visited = true; // je v kopici
        w.intree = false; // se ni v drevesu
        w.parent = v; // potencialni oce
        // trenutna najkrajsa povezava do drevesa:
        w.distance = ((Double)e.evalue).doubleValue();
        q.insert(w) ;
      }
     else // visited
        if (! w.intree && ((Double)e.evalue).doubleValue() <
         w.distance) { //nasli smo krajso povezavo do drevesa
           w.parent = v; // novi potencialni oce
           // popravi razdaljo v prioritetni vrsti:
           q.decreaseKey(w, e.evalue) ;
        }
        e = g.nextEdge(v, e) ;
  } // while e
} // while !empty
} // Prim
```

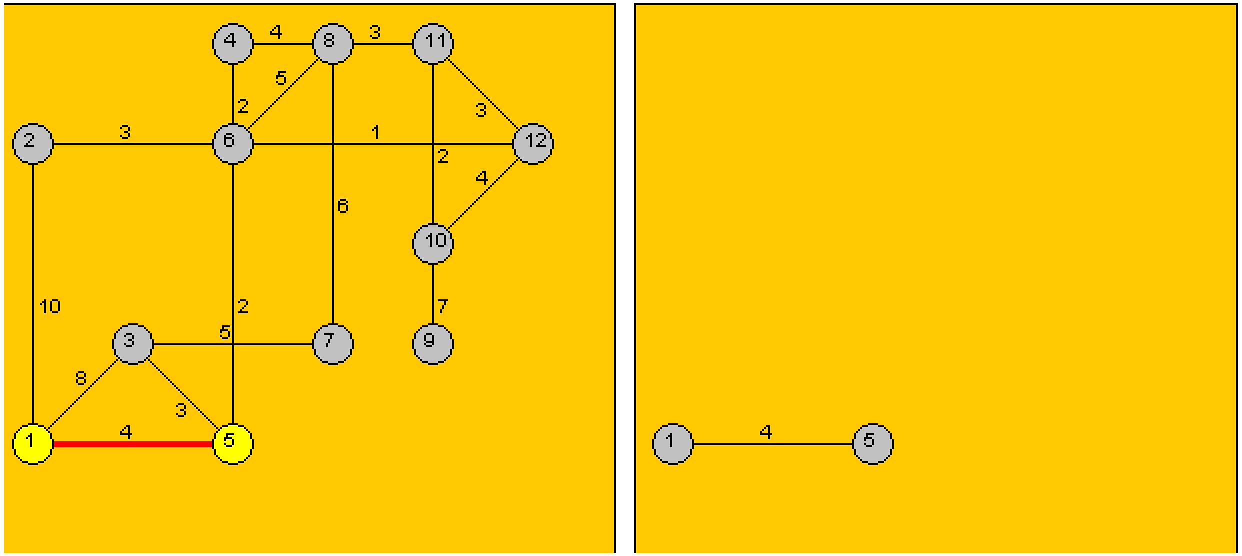
O(m log n)

Primov algoritem je **požrešen**, pa vseeno zagotavlja optimalno rešitev!

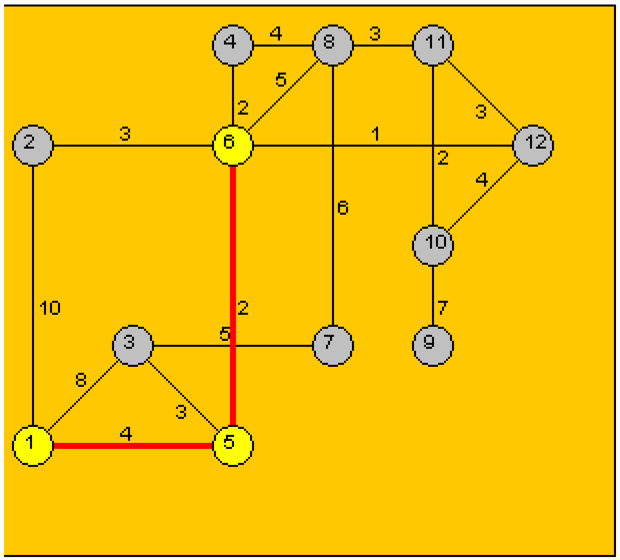
PRIMER

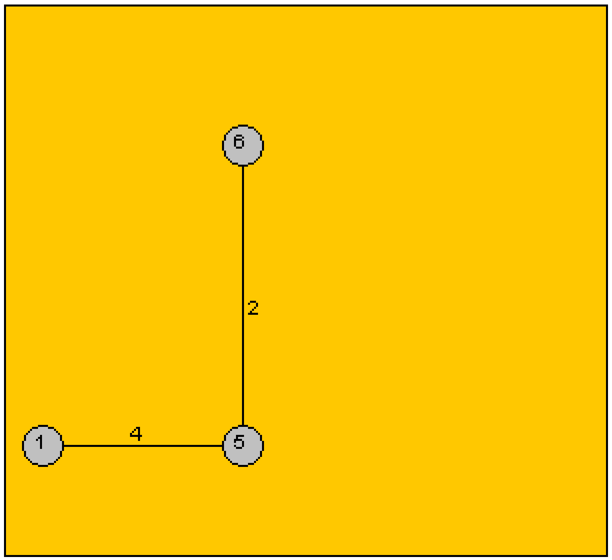


Primer

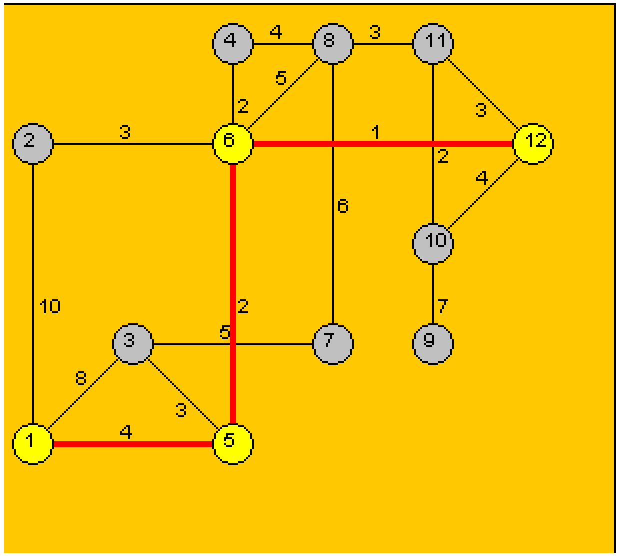


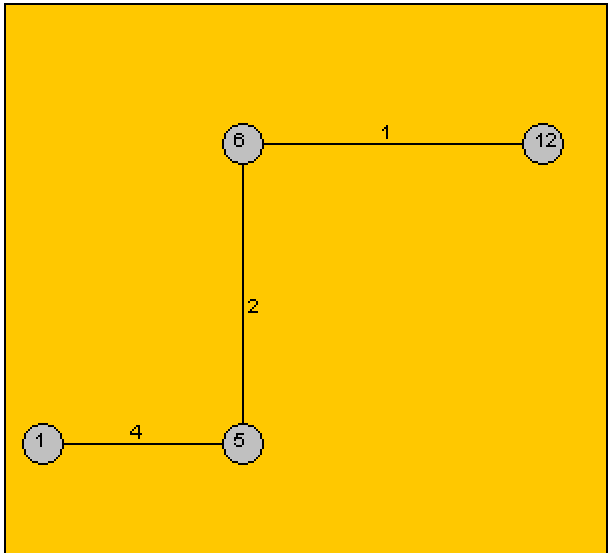
PRIMER



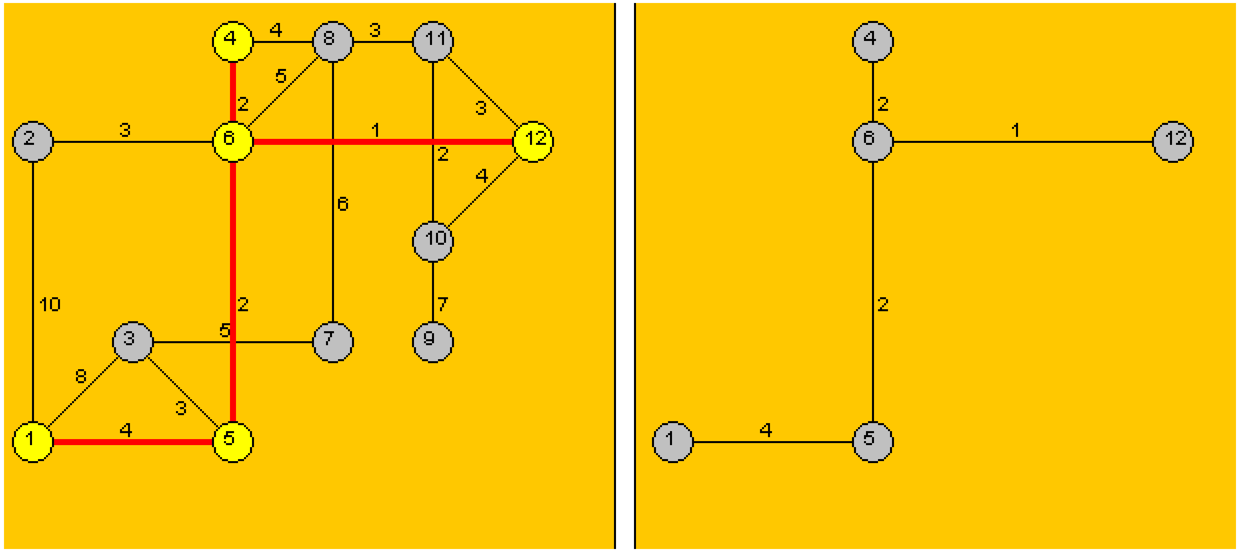


Primer

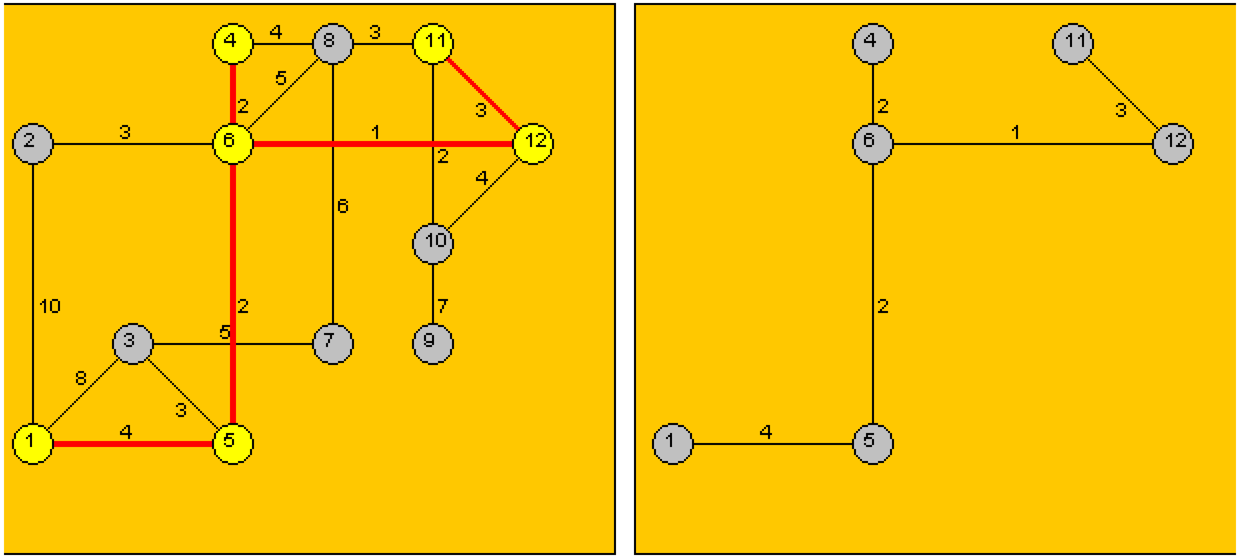




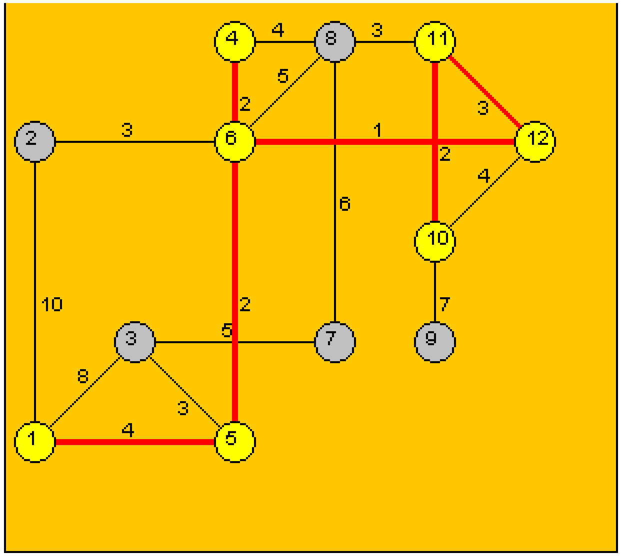
Primer

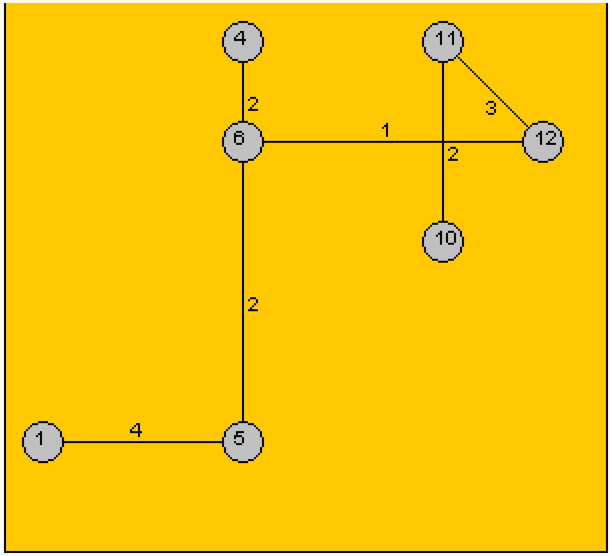


Primer

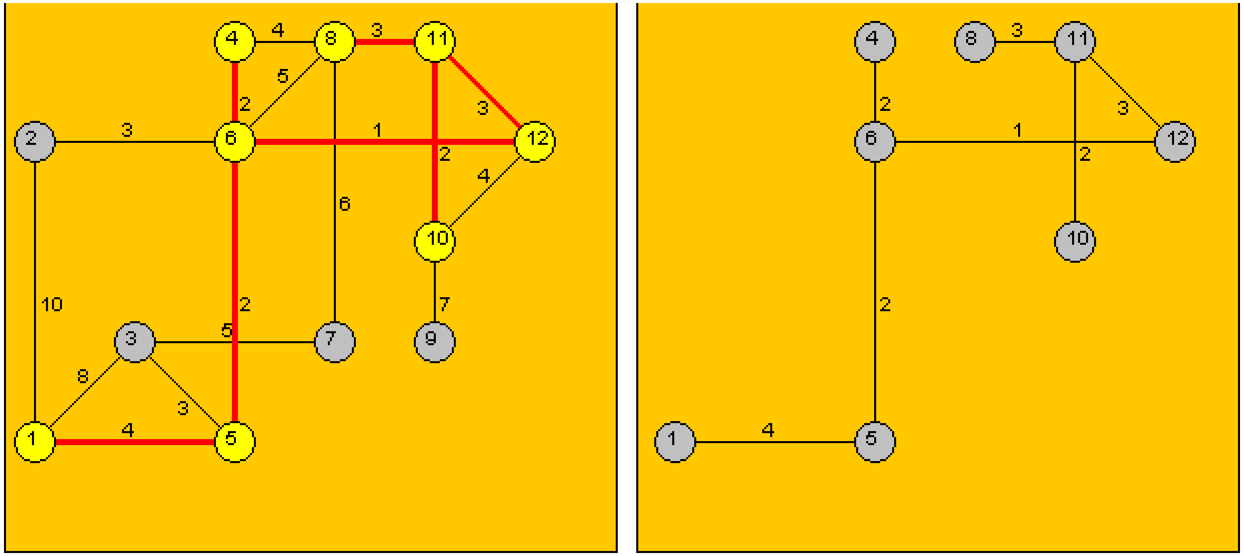


Primer



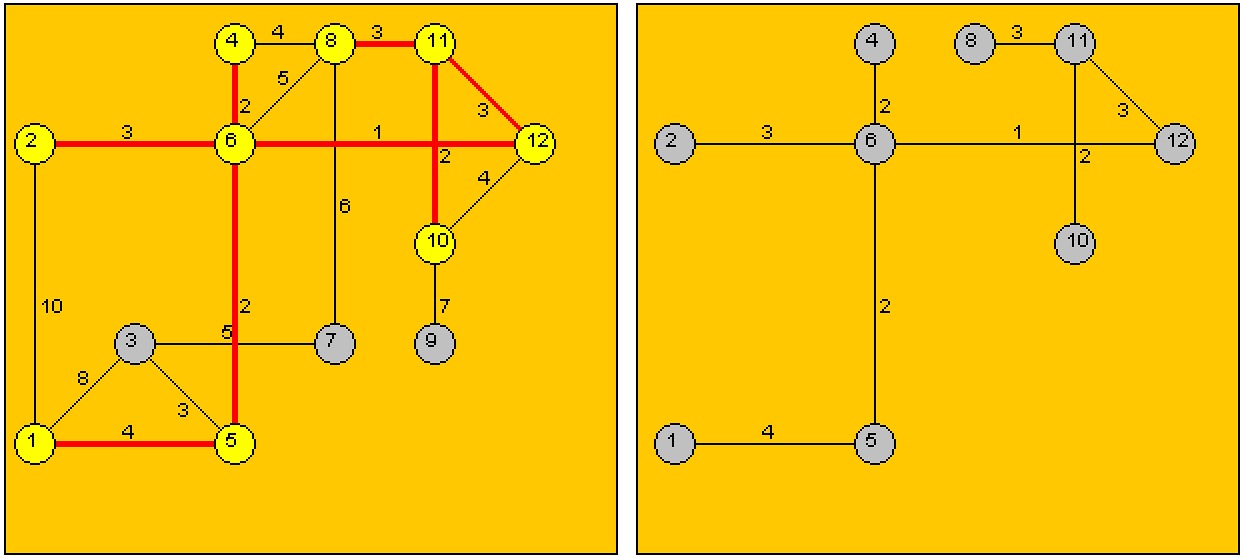


Primer

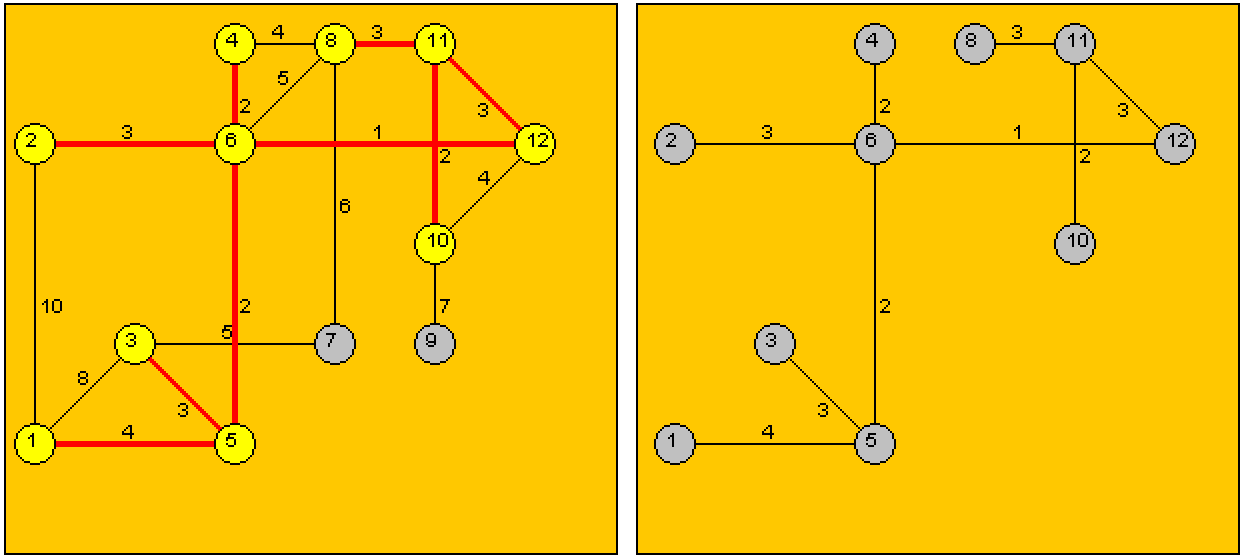


4
4
8
3
11
5
2
3
2
3
6
1
12
2
4
6
10
10
2
7
3
5
7
9
8
3
1
4
5
4
3
8
3
11
2
3
6
1
12
2
10
2
1
4
5

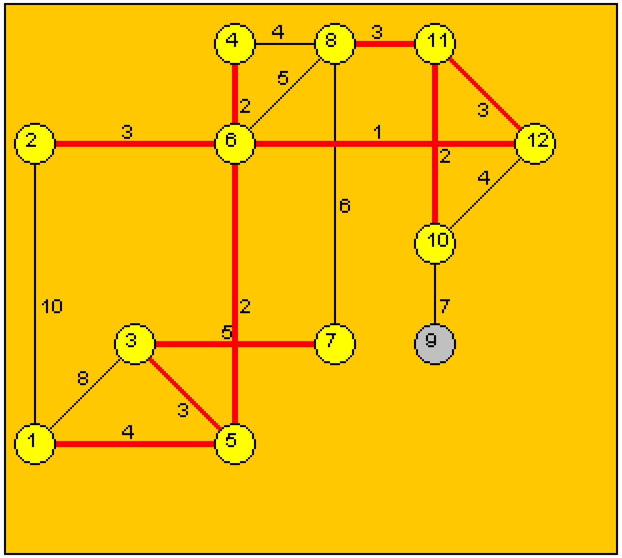
PRIMER

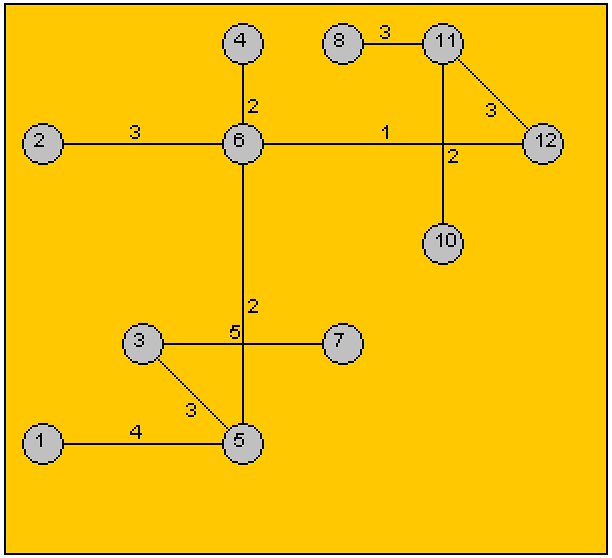


Primer


4
4
8
3
11
5
2
3
2
3
6
1
12
2
4
6
10
10
2
5
3
7
9
7
8
3
1
4
5
4
2
8
3
11
2
3
2
3
6
1
12
2
10
2
3
3
1
4
5

PRIMER





PRIMER