Kruskalov algoritmus

```
Sort edges in order of increasing weight
so that w[f[1]] <= w[f[2]] <= ... <=w[f[m]]

T:=empty set
for i:=1 to m do
  let u,v be the endpoints of edge f[i]
  if there is no path between u and v in T then (**)
    add f[i] to T
return T
```

Štruktúra pre UNION/FIND-SET

```
function MAKE-SET(x):
  x.parent = x; x.rank = 0

function UNION(x,y):
  xs = FIND-SET(x); ys = FIND-SET(y);
  if xs.rank > ys.rank: ys.parent = xs
  else: xs.parent = ys
  if xs.rank == ys.rank:
     ys.rank = ys.rank + 1

function FIND-SET(x):
  if x != x.parent:
     return FIND-SET(x.parent)
  else: return x
```

Štruktúra pre UNION/FIND-SET s kompresiou cesty

```
function MAKE-SET(x):
  x.parent = x; x.rank = 0

function UNION(x,y):
  xs = FIND-SET(x); ys = FIND-SET(y);
  if xs.rank > ys.rank: ys.parent = xs
  else: xs.parent = ys
  if xs.rank == ys.rank:
     ys.rank = ys.rank + 1

function FIND-SET(x):
  if x != x.parent:
**   x.parent=FIND-SET(x.parent)
**return x.parent
```

Primov algoritmus

```
S := {s};
T := empty set;
while S<>V do
  e := (u,v) such that u is in S, v is not in S and      (*)
       w(e) is smallest possible;
  add v to S;
  add e to T;
return T;
```

```
S := {s};
T := empty set;
// initialize data structure
for each u not in S
  dist[u] := w(s,u);
  other[u] := s;
// main computation
while S<>V do
  v := vertex which is not in S and has the smallest dist[v];
  e := (v, other[v]);
  add v to S;
  add e to T;
  // update data structure
  for each x not in S
    if w(v,x)<dist[x] then
      dist[x] := w(v,x);
```

```
        other[x] := v;
return T;
```