

Minimum Spanning Trees

Input: connected undirected graph $G = (V, E)$ with nonnegative edge weights $w(u, v)$ for (u, v) in E

Output: Minimum Spanning Tree (MST)
= set A of edges forming a tree, containing all nodes, of minimum total weight

For set of edges A : safe edge e for A
= if A is subset of a MST, so is $A \cup \{e\}$

Generic-MST (V, E, w) :

```
{  
    A = emptyset;  
    while (A not a MST)  
    {  
        find edge e in E that is safe for A;  
        A = A  $\cup$  {e};  
    }  
    return A;  
}
```

Theorem:

Let $A \subseteq E$ be subset of MST for $G = (V, E, w)$.

Let $(S, V-S)$ be **cut** that **respects** A

(i.e. $S \subseteq V$, **no edge in A has only one endpoint in S**).

Let (u, v) be a **light** edge (of **minimum weight**)
crossing the cut, i.e. $u \in S$, $v \in V-S$.

Then (u, v) is safe for A .

MST-Kruskal (V,E,w)

```
{
  A = emptyset;
  for (each u in V)
    make_set(u);

  sort E in order of nondecreasing weight;

  for (each (u,v) in E, in that sort order)
    if ( find_set(u) != find_set(v) )
      // u and v are in different components of A
      {
        A = A  $\cup$  { (u,v) };
        union (u,v); // u and v in same component
      }

  return A;
}
```

Methods for union/find problem:

make_set(u): put u in set of its own

find_set(u): find out which set u belongs to

union(u,v): merge the two sets containing u and v.

MST-Prim (V,E,w)

```
{
    Q = V;    // priority queue sorted by key[u] for u in Q

    for (each u in V)
        key[u] = infinity;
    key[r] = 0;    // grow spanning tree from root r
    p[r] = null; // root has no parent in MST

    while (Q not empty)
    {
        u = Q.extract_min; // extract u with smallest key[u]
        for (each v in Adj(u) )
            if (v in Q and w(u,v) < key[v])
            {
                p[v] = u ;
                key[v] = w(u,v);
            }
    }
}
```

Here: $key[v]$ = weight of shortest edge $(p[v],v)$ joining v to A , with $p[v] \in A$, A to become MST

Dijkstra (V,E,w,r)

// output: shortest-path tree from r to any other node

```
{
    Q = V;    // priority queue sorted by key[u] for u in Q

    for (each u in V)
        key[u] = infinity; // distance of u from r
    key[r] = 0;    // grow shortest path tree from root r
    p[r] = null;

    while (Q not empty)
    {
        u = Q.extract_min; // extract u with smallest key[u]
        for (each v in Adj(u) )
            if (v in Q and key[u] + w(u,v) < key[v])
            {
                p[v] = u ;
                key[v] = key[u] + w(u,v);
            }
    }
}
```

Here: $key[v]$ = shortest distance from r to v using
(except for v) only nodes in $V-Q$,
via edge $(p[v],v)$, with $p[v] \in V-Q$, $v \in Q$.

