Minimum Spanning Tree

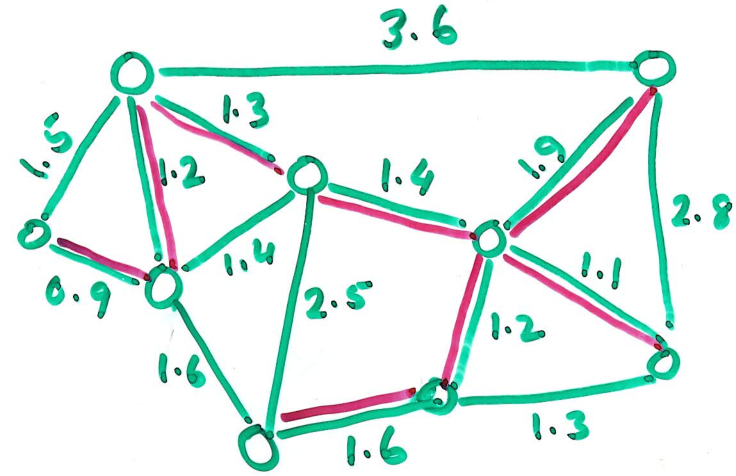
(V,E) is a connected, undirected, weighted graph. A <u>spanning tree</u> is a subgraph (V,T), $T \subseteq E$ that is connected and has no cycle.

- Recall that a tree with n vertices has $n-1$ edges.

- A <u>minimum spanning tree</u> (MST) is a spanning tree (V,T) that minimizes

$$w(T) = \sum_{\{u,v\} \in T} w(\{u,v\}).$$

Ex.



MST is denoted with red-green edges

We will study two algorithms Prim's and Kruskal's algorithms for computing MST. Both of these algorithm can be viewed as a special case of a generic process which we study first.

Growing an MST

Invariant $A \subseteq E$ is always a subset of some MST of (V,E).

An edge $uv \in E$ is **safe** for A if $uv \notin A$ and $A \cup \{uv\}$ also satisfies the invariant.

Generic Method

```
A := ∅
while A is not a spanning tree
        of V yet do

    find a safe edge uv;
    A := A ∪ {uv}

endwhile
```

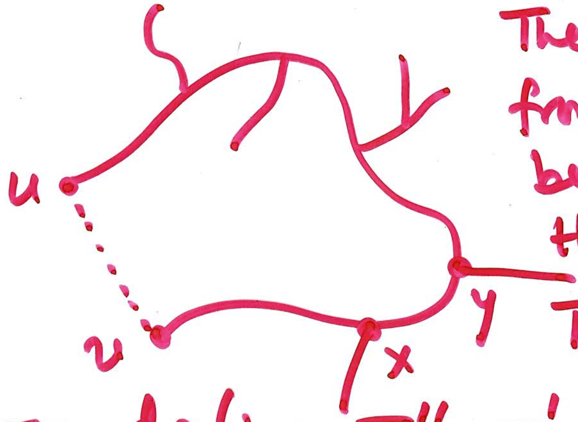
So far the method is trivial. The main part is how to choose safe edges.

A *cut* is a partition $V = W \,\dot{\cup}\, (V-W)$;
it *respects* $A \subseteq E$ if $A \subseteq \binom{W}{2} \cup \binom{V-W}{2}$,
i.e. all edges of A are either connecting two vertices in W or in $(V-W)$.

An edge uv *crosses* the cut if one vertex belongs to W and the other in $V-W$.

Claim. Let A be a subset of some MST of (V,E). Let $(W, V-W)$ be a cut that respects A. Let uv be a crossing edge that minimizes $w(uv)$. Then uv is safe for A.

Proof Consider an MST $=(V,T)$ with $A \subseteq T$.
If $uv \in T$ we are done.
So, assume $uv \notin T$ and $T' = T \cup \{uv\}$.



There is a unique path from u to v in T. Let xy be an edge on this path that crosses $(W, V-W)$.
Thus, $w(uv) \leq w(xy)$.

Now define $T'' = T' - \{xy\}$. T'' is again a spanning tree of V and $W(T'') \leq W(T)$. So, (V, T'') is an MST.

Prim's Algorithm

For each vertex i we assume a field p ($V[i].p$) that can be used to store a real number which is the priority of i.

We first add all vertices to a priority queue PQ, and the tree growing process starts. Here, the vertices which are extracted from PQ forms the cut with the rest of the vertices in PQ. Then we update the priorities of vertices with respect to new cross edges each time we include a vertex from PQ to our current set.

Initialization

```
PQ := ∅;
for i := 1 to n do
  if i ≠ k then V[i].p := ∞
    else V[i].p := 0;
         V[i].π := nil;
  endif
  add i to PQ with
         priority V[i].p
endfor
```

Main algorithm.

```
While PQ ≠ ∅ do
   i := Extract_min (PQ);  t := V[i].adj;
   while t ≠ nil do  j := t.v;
         if j ∈ PQ and w(ij) < V[j].p then
               V[j].p := w(ij);  V[j].π := i
         endif
         t := t.next
   endwhile
endwhile
```

After running the algorithm MST can be recovered from the π field. The vertices in PQ needs to be marked and they are unmarked once they removeed from PQ.

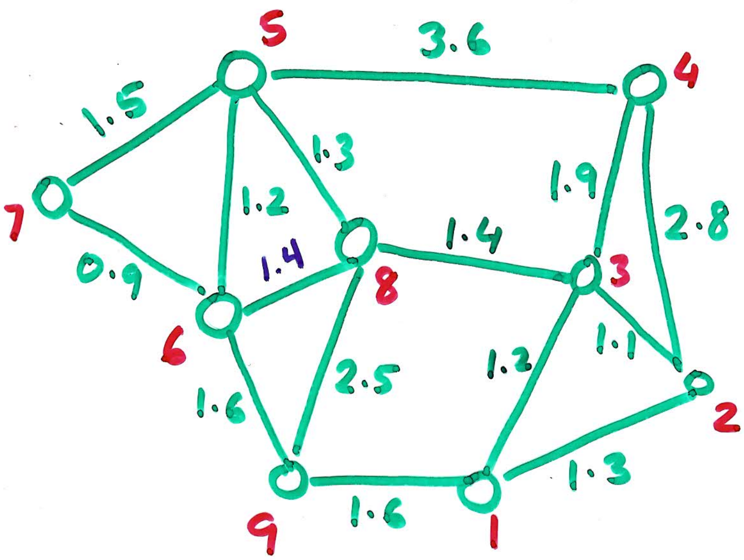
We have n insertions to PQ : $n\log n$

n minimum deletions : $n\log n$

At most m decrease-key : $O(m)$ if we use Fibonacci heap.

Total: $O(n\log n + m)$ if we use Fibonacci heap.

Example



1	2	3	4	5	6	7	8	9	
0	∞	∞	∞	∞	∞	∞	∞	∞	remove 1
	1.3	1.2	∞	∞	∞	∞	∞	1.6	remove 3
	1.1		1.9	∞	∞	∞	1.4	1.6	remove 2
			1.9	∞	∞	∞	1.4	1.6	remove 8
			1.9	1.3	1.4	∞		1.6	remove 5
			1.9		1.2	1.5		1.6	remove 6
			1.9			0.9		1.6	remove 7 9 4

Kruskal's algorithm.

This algorithm considers the globally shortest yet edge not considered. If this edge crosses a cut then it is safe and is added.

Algorithm uses two data structures:
  a priority queue PQ for the edges.
  a set-system C for the vertices

Initialization:

$PQ := \emptyset$;
for each edge $uv \in E$, insert uv with
    its weight as priority in PQ;

$C = \{ \{v\} \mid v \in V \}$ * initialize a set *

Main.

```
A = ∅;
while |A| < n-1 do
    uv := Extractmin(PQ);
    find U, V ∈ C s.t. u ∈ U and v ∈ V;
                        * find oprn *
    if U ≠ V then
        A := A ∪ {uv}
        U := U ∪ V      * set union *
    endif
endwhile
```

Complexity: $O(n\log n + m\log n)$: ordinary Union-find