Prim's Algorithm (G, w, r)

1. $Q \leftarrow V$
2. **for** each $u \in Q$
3. **do** $Key[u] = \infty$
4. $Key[r] = 0$.
5. $\pi(r) = NIL$
6. **While** $Q \neq \emptyset$
7. **do** $\boxed{u \leftarrow EXTRACT\text{-}MIN(Q)}$ log n
8. **for** each $v \in Adj[u]$
9. **do** **if** $v \in Q$ & $w(u,v) < key[v]$
10. then $\pi(v) \leftarrow u$.
11. $\boxed{key[v] \leftarrow w(u,v)}$

1. $X = \{(v, \pi(v)) : v \in V - \{r\} - Q\}$

2. $V - Q$ are the vertices of the tree

3. If $v \in Q$ & $\pi(v) \neq NIL$

$Key[v] < \infty$.

$Key[v]$ gives the lightest edge connecting v to some vertex in X.

Correctness & Complexity.

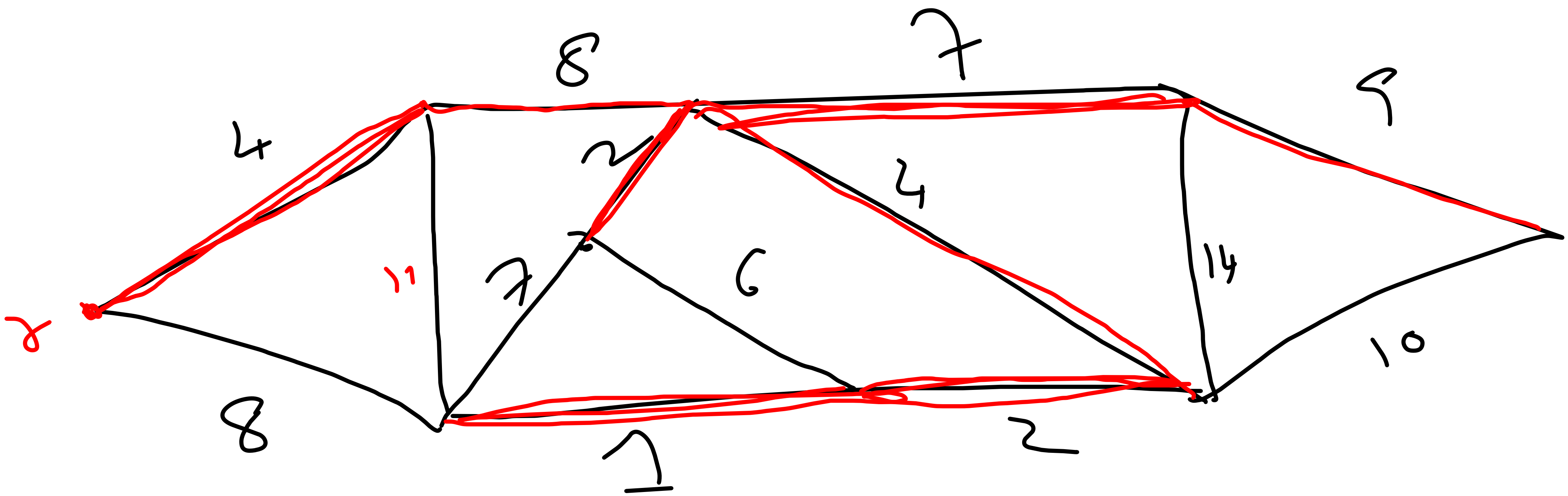
Correctness follows from the cut property.

Complexity of Prim's algorithm depends on how you implement the priority queue. If Q is implemented as a binary heap then line 7 takes $O(\log n)$ time

Hence the <u>while</u> loop can be executed in $O(n \log n)$ time.

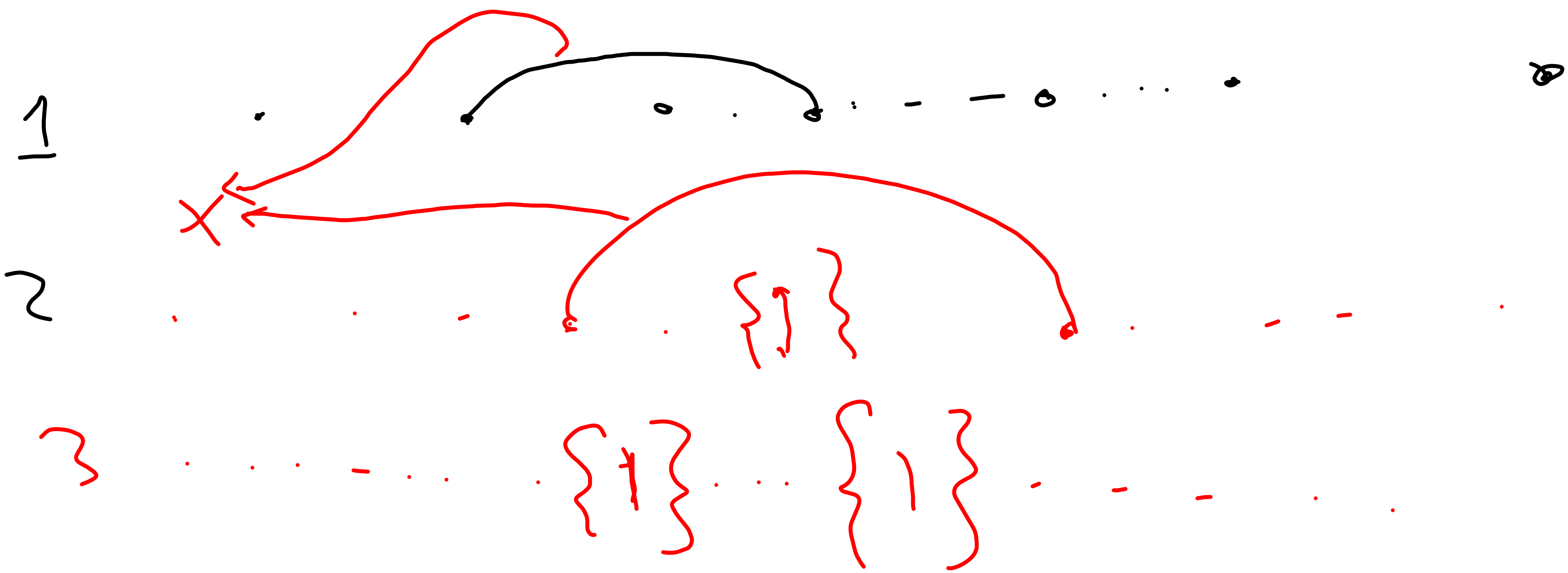
Updating the key can be implemented in $O(\log n)$ time & hence the <u>for</u> loop can be executed in $O(e \log n)$ time.

Running time $O(n\log n + e \log n) = O(e \log n)$



4
8
7
9
2
4
11
7
6
14
r
8
1
2
10

Kruskal's Algorithm.



Kruskal's alg. adopts a different
strategy. Instead of growing a
single tree Kruskal's alg. adds
a lightest possible edge. To start
we sort the edges in order of
increasing wt. Initially $X = \varphi$
& each vertex in V is regarded
as a trivial tree. We ~~to~~ examine

each edge in order & if end pts of this edge belong to the same tree we discard. Otherwise, this edge is added to X. This causes the two trees containing the end pts to merge into a single tree.

To implement, given a forest of trees, we need to decide given a pair of vertices if they belong to the same tree. For the purpose of implementation, each tree is regarded as a set of vertices of that tree. Our data structure should support merging of two trees. So the data structure should maintain a collection of disjoint set & should support the following

1. MAKESET(x): Create a set consisting of the single element x.

2. FIND(u): Given u, it finds the set containing u.

3. UNION(u, v): Replace the set containing u & the set containing v by their Union.

Kruskal's Algorithm.

$X \leftarrow \phi$

Sort E by weight, $\longrightarrow O(e \lg e)$ time.

for $u \in V$

MAKESET(u)

end for

for $(u, v) \in E$ in increasing order **do**

if FIND(u) $\neq$ FIND(v)

$X \leftarrow X \cup \{(u,v)\}$.

UNION(u, v)

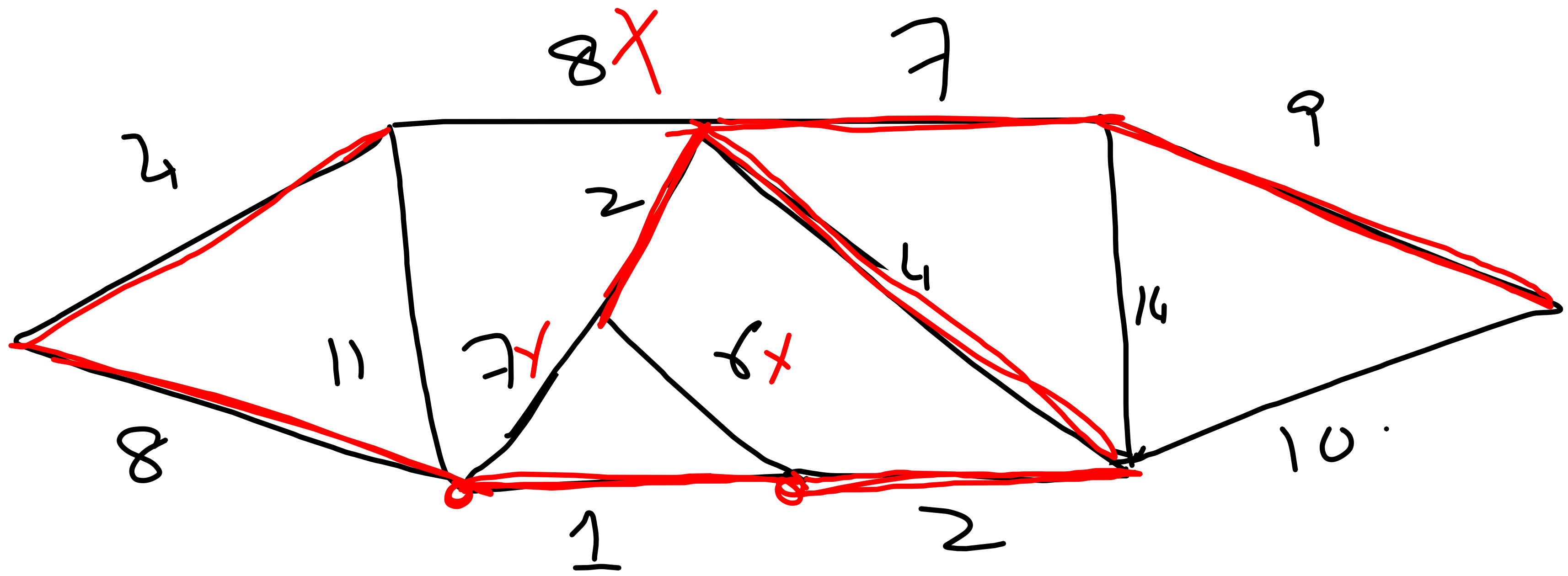
end for

return X.

Complexity

If we are given a sorted list of edges, then the running time is determined by $n-1$ UNION of $2e$ FIND operations. Using a fast UNION-FIND, this can be achieved in time $O(e \log n)$

If the edges are unsorted, the running time is $O(e \log e)$

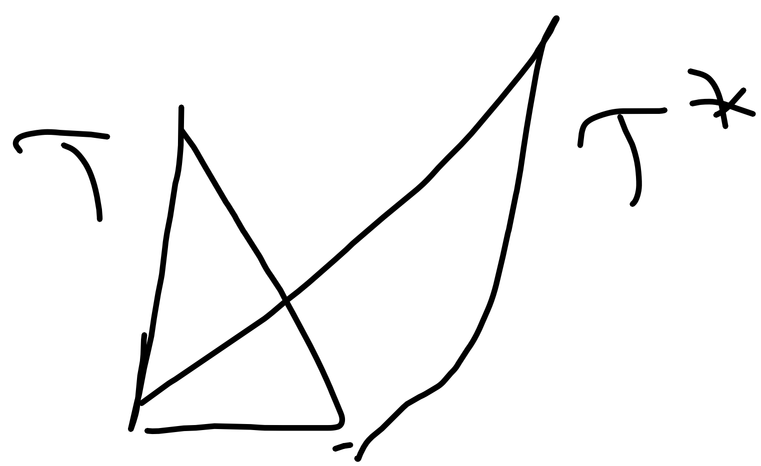

8
7
4
9
2
11
7
4
14
6
8
10
1
2

Ex 1 Show that if the wts. are distinct, then there is a unique MST

Ex 2 Prove the following exchange property for spanning trees.

- Let T & T' be spanning trees in $G = (V, E)$. Given an edge $e' \in T' - T$, $\exists$ an edge $e \in T - T'$ s.t. $(T - \{e\}) \cup \{e'\}$ is also a spanning tree.

Use this to show that one can "walk" from any spanning tree to a minimum spanning tree.



3. <u>Clustering</u>.

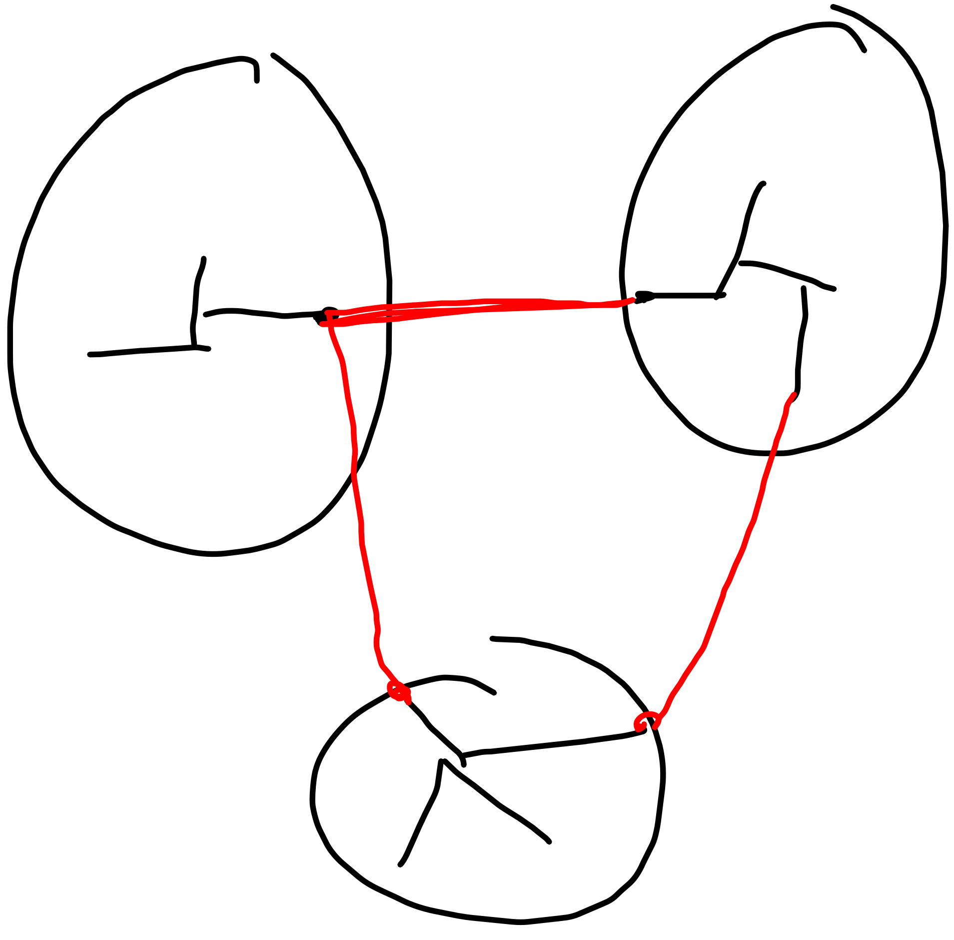
Given a set of objects e.g texts, photos, micro-organism etc we want to organize or classify these objects into "coherent" groups.

We assume that we are given a distance f^n on the objects with the understanding that objects with larger distance are less similar.

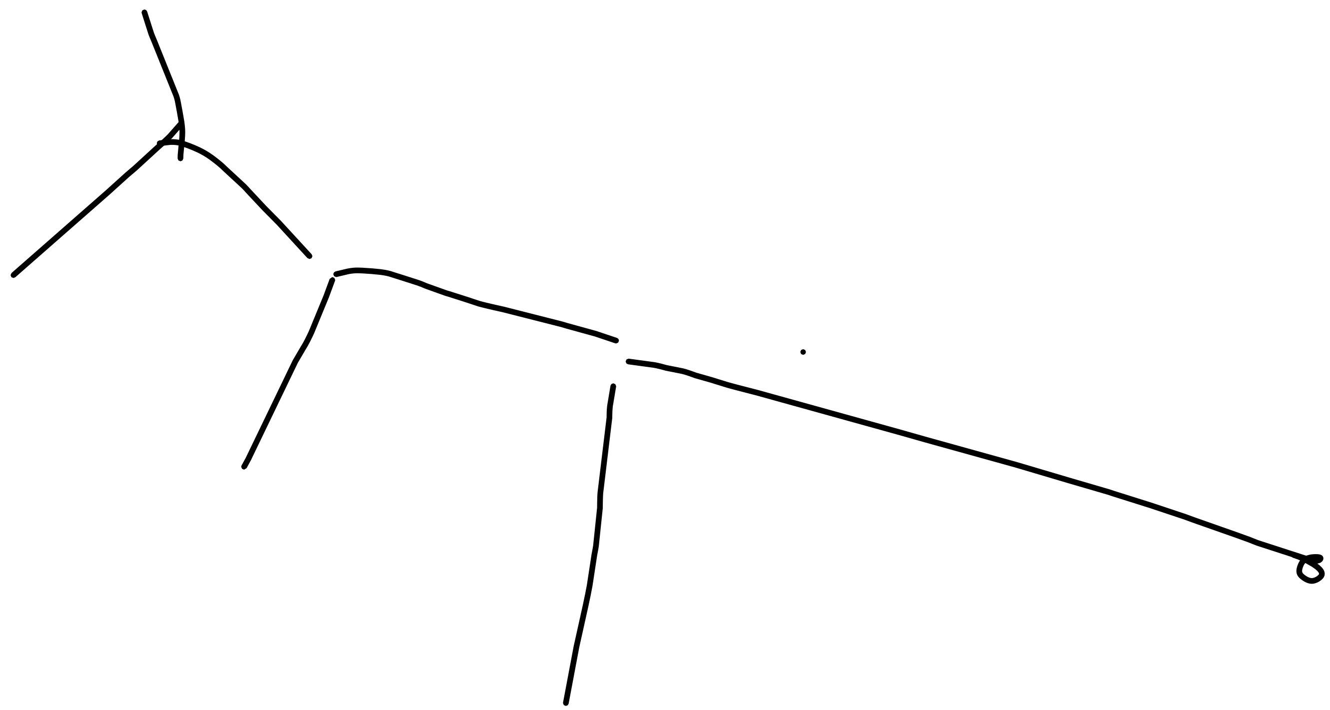
Thus given a distance fn on the set of objects we seek to divide them into groups so that objects in the same group are "near" & objects in different groups are "far apart".

Clustering with maximum spacing

We are given a set U of n objects with labels $p_1, p_2, \ldots p_n$. We wish to partition U into k non-empty sets $C_1, C_2, \ldots C_k$. Such a partion is called a k-clustering. We define the spacing of a k-clustering to be the min. of any pair of pts in different clusters



Our clustering problem is to find a k-clustering with maximum spacing



The Algorithm

To get a k-clustering with maximum spacing we run Kruskal's algorithm for $n-k$ steps

The remaining $k-1$ $\big(= (n-1)-(n-k)\big)$ edges are the $k-1$ most expensive edges of the MST

This is equivalent to running the full Kruskal's alg. & then discarding the $k-1$ most expensive edges. The first of these gives the spacing of the k-clustering obtain.