Minimum spanning trees

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
A network construction problem: Minimum Spanning Tree

CLRS 23, KT 4.5, DPV 5.1

- We have a set of locations.
- For some pairs of locations it is possible to build a link connecting the two locations, but it has a cost.

- We want to build a network (if possible), connecting all the locations, with total minimum cost.
- So, the resulting network must be a tree.
Network construction: Minimum Spanning Tree

- We have a set of locations. Build a link connecting the locations $i$ and $j$ has a cost $w(v_i, v_j)$.
- We want to build tree spanning all the locations with total minimum cost.
Properties of trees

- A tree on \( n \) nodes has \( n - 1 \) edges.
- Any connected undirected graph with \( n \) vertices and \( n - 1 \) edges is a tree.
- An undirected graph is a tree iff there is a unique path between any pair of nodes.

Let \( G = (V, E) \) be a (undirected) graph.

- \( G' = (V', E') \) is a subgraph of \( G \) if \( V' \subseteq V \) and \( E' \subseteq E \).
- A subgraph \( G' = (V', E') \) of \( G \) is spanning if \( V' = V \).
- A spanning tree of \( G \) is a spanning subgraph that is a tree.

Any connected graph has a spanning tree
The problem

Minimum Spanning Tree problem (MST)

Given as input an edge weighted graph \( G = (V, E, w) \), where \( w : E \to \mathbb{R} \). Find a tree \( T = (V, E') \) with \( E' \subseteq E \), such that it minimizes \( w(T) = \sum_{e \in E(T)} w(e) \).
Some definitions

For a graph $G = (V, E)$:
A path is a sequence of consecutive edges.
A cycle is a path ending in an edge connecting to the initial vertex, with no other repeated vertex.
A cut is a partition of $V$ into two sets $S$ and $V - S$.
The cut-set of a cut is the set of edges with one end in $S$ and the other in $V - S$. $\text{cut}(S, V - S) = \{e = (u, v) \in E \mid u \in S \land v \notin S\}$
MST: Properties

Given a weighted graph $G = (V, E, w)$, assume that all edge weights are different.

A MST $T$ in $G$ has the following properties:

- **Cut property**
  $e \in T \iff e$ is the *lightest* edge across some cut in $G$.

- **Cycle property**
  $e \notin T \iff e$ is the *heaviest* edge on some cycle in $G$.

The MST algorithms use two rules for adding/discarding edges.
The problem

Properties

The cut and the cycle properties

A generic algorithm

Prim’s algorithm

Kruskal’s algorithm

Description

Union-Find implementation

Cost

An application: Clustering

MST: Properties

The $\leftrightarrow$ implication of the cut property yields the blue rule (include), which allow us to include safely in $T$ a min weight edge from some identified cut.

The $\Rightarrow$ implication of the cycle property will yield the red rule (exclude) which allow us to exclude from $T$ a max weight edge from some identified cycles.
The cut property

Let $G = (V, E, w)$, $w : E \to \mathbb{R}^+$, such that all weights are different. Let $T$ be a MST of $G$.

Removing an edge $e = (u, v)$ from $T$ yields two disjoint trees $T_u$ and $T_v$, so that $V(T_u) = V - V(T_v)$, $u \in T_u$ and $v \in T_v$. Let us call $S_u = V(T_u)$ and $S_v = V(T_v)$.

Claim

$e \in E(T)$ is the min-weight edge among those in cut($S_u, S_v$).

Proof.

Otherwise, we can replace $e$ by an edge in the cut with smaller weight. Thus, forming a new spanning tree with smaller weight.
The cut property

Claim (The cut rule)

For $S \subseteq V$, let $e = (u, v)$ be the min-weight edge in $\text{cut}(S, V - S)$, then $e \in T$.

Proof.

- Assume $e \notin T$, $u \in S$ and $v \notin S$.
- $T$ is spanning, then a path $P(u, v)$ from $u$ to $v$ exists in $T$.
- $u \in S$ and $v \notin S$: there is $e' \in \text{cut}(S, V - S)$ in $P(u, v)$.
- Replacing $e'$ with $e$ produces another spanning tree.
- But then, as $w(e) > w(e')$, $T$ was not optimal.
The cycle property

For an edge $e \notin T$, adding it to $T$ creates a graph $T + e$ having a unique cycle involving $e$. Let's call this cycle $C_e$.

Claim

For $e \notin E(T)$, $e$ is the max-weight edge in $C_e$.

Proof.

Otherwise, removing any edge different from $e$ in $T + e$ produces a spanning tree with smaller total weight.
The cycle property

Claim (The cycle rule)

*For a cycle $C$ in $G$, the edge $e \in C$ with max-weight can not be part of $T$."

Proof.

Observe that, as $G$ is connected, $G' = (V, E - \{e\})$ is connected. Furthermore, a MST for $G'$ is a MST for $G$. 
Generic greedy for MST: Apply blue and/or red rules

- The two rules show the optimal substructure of the MST. So, we can design a greedy algorithm.
- Blue rule: Given a cut-set between $S$ and $V - S$ with no blue edges, select from the cut-set a non-colored edge with min weight and paint it blue
- Red rule: Given a cycle $C$ with no red edges, select a non-colored edge in $C$ with max weight and paint it red.
- Greedy scheme:
  Given $G$, apply the red and blue rules until having $n - 1$ blue edges, those form the MST.

Robert Tarjan: Data Structures and Network Algorithms, SIAM, 1984
Application of red/blue rules

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
Application of red/blue rules
Application of red/blue rules

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering
Application of red/blue rules
Application of red/blue rules
Application of red/blue rules
Application of red/blue rules

Graph with nodes and edges labeled with numbers.
Application of red/blue rules
Application of red/blue rules
Application of red/blue rules

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Kruskal's algorithm

Cost
Application of red/blue rules

A generic algorithm

Prim's algorithm

Kruskal's algorithm
Application of red/blue rules
Application of red/blue rules

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
Application of red/blue rules
Application of red/blue rules
Application of red/blue rules
Application of red/blue rules
Greedy for MST: Correctness

**Theorem**

The greedy scheme finishes in at most \( m \) steps and at the end of the execution the blue edges form a MST.

**Sketch.**

- As in each iteration an edge is added or discarded, the algorithm finishes after at most \( m \) applications of the rules.
- As the red edges cannot form part of any MST and the blue ones belong to some MST, the selections are correct.
- A set of \( n - 1 \) required edges form a spanning tree!

We need implementations for the algorithm!
A short history of MST implementation

There has been extensive work to obtain the most efficient algorithm to find a MST in a given graph:

- O. Borůvka gave the first greedy algorithm for the MST in 1926. V. Jarnik gave a different greedy for MST in 1930, which was re-discovered by R. Prim in 1957. In 1956 J. Kruskal gave a different greedy algorithms for the MST. All those algorithms run in $O(m \lg n)$.

- Fredman and Tarjan (1984) gave a $O(m \log^* n)$ algorithm, introducing a new data structure for priority queues, the Fibonacci heap. Recall $\log^* n$ is the number of times we have to apply iteratively the log operator to $n$ to get a value $\leq 1$, for ex. $\log^* 1000 = 2$.

- Gabow, Galil, Spencer and Tarjan (1986) improved Fredman-Tarjan to $O(m \log(\log^* n))$.


- In 1997 B. Chazelle gave an $O(m\alpha(n))$ algorithm, where $\alpha(n)$ is a very slowly growing function, the inverse of the Ackermann function.
Basic algorithms for MST

- **Jarník-Prim (Serial centralized)** Starting from a vertex \(v\), grows \(T\) adding each time the lighter edge already connected to a vertex in \(T\), using the blue rule.

  Uses a priority queue

- **Kruskal (Serial distributed)** Considers every edge, in order of increasing weight, to grow a forest by using the blue and red rules. The algorithm stops when the forest became a tree.

  Uses a union-find data structure.
Jarník–Prim greedy algorithm.

V. Jarník, 1936, R. Prim, 1957

- The algorithm keeps a tree $T$ and adds one edge (and one node) to $T$ at each step until it became spanning.
- Initially the tree $T$ has one arbitrary node $r$, and no edges.
- At each step $T$ is enlarged adding a minimum weight edge in the set $\text{cut}(V(T), V – V(T))$.
- The algorithm is correct as it applies always the blue rule.
Jarník - Prim greedy algorithm.

\[
\text{MST} \ (G, w, r) \\
T = \{r\} \\
\text{for } i = 2 \text{ to } |V| \text{ do} \\
\quad \text{Let } e \text{ be a min weight edge in the cut}(V(T), V - V(T)) \\
\quad T = T \cup \{e\} \\
\text{end for}
\]
Example
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Example graph:

- Nodes: a, b, c, d, e, f, g, h
- Edges with weights:
  - a-b: 6
  - a-c: 5
  - a-e: 14
  - b-c: 4
  - b-f: 10
  - c-d: 9
  - c-f: 8
  - d-h: 15
  - e-f: 3
  - g is a red node.
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Kruskal's algorithm

Diagram:

- Nodes: a, b, c, d, e, f, g, h
- Edges with weights:
  - a-b: 6
  - b-c: 4
  - c-d: 9
  - a-e: 14
  - e-f: 3
  - f-g: 8
  - g-h: 15
  - g-a: 14

This diagram represents a graph with weighted edges.
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Example graph with weights:
- a to b: 6
- a to c: 5
- b to c: 4
- e to f: 3
- f to c: 8
- b to g: 2
- g to d: 9
- g to h: 15
- a to d: 10
- a to e: 14
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
Example
Example
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Example
Example
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
Example
Example

A generic algorithm

Example:

```
+---+---+---+---+
| a | b | c | d |
+---+---+---+---+
| e | f | g | h |
```

Graph:

- **Prim's algorithm**
  - Description
  - Union-Find implementation
  - Cost
  - An application: Clustering

- **Kruskal's algorithm**
  - Properties
  - The cut and the cycle properties
  - A generic algorithm
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Example

Diagram of a weighted graph with labeled edges:
- Edge labels: 2, 3, 4, 5, 6, 8, 9, 10, 14, 15
- Nodes: a, b, c, d, e, f, g, h
The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm

Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering
Jarník–Prim: Implementation

Use a priority queue to choose min weight $e$ in the cut set. In doing so we have to discard some edges

\[
\text{MST} \ (G, w, r)
\]
\[
T = (\{r\}, \emptyset); \ Q = \emptyset; \ s = 0
\]
Insert in $Q$ all edges $e = (r, v)$ with key $w(r, v)$

while $s < n - 1$ and $Q$ is not empty do

\[
(u, v, w) = Q.pop()
\]
if $u \notin V(T)$ or $v \notin V(T)$ then

Let $u'$ be the vertex from $(u, v)$ that is not in $T$
Insert in $Q$ all the edges $e = (u', v') \in E(G)$ for
$v' \notin V(T)$ with key $w(e)$
add $e$ to $T$; $++s$
end if
end while
Jarník - Prim greedy algorithm: Correctness

- The algorithm discards edge $e$:
  Such an edge $e = (u, v)$ has $u, v \in V(T)$, so it forms a cycle with the edges in $T$. But, $e$ is the edge with highest weight in this cycle. This is an application of the red rule.

- The algorithm adds to $T$ edge $e$:
  Then $e$ has minimum weight among all edges in $Q$, as $Q$ contains all edges in the cut-set($V(T), V - V(T)$). This is the blue rule

- Therefore the algorithm computes a MST.
Jarník–Prim greedy algorithm: Cost

**Time:** depends on the implementation of the priority queue $Q$. We have $\leq m$ insertions on the priority queue.

- $Q$ an unsorted array: $T(n) = O(|V|^2)$;
- $Q$ a heap: $T(n) = O(|E| \log |V|)$;
- $Q$ a Fibonacci heap: $T(n) = O(|E| + |V| \log |V|)$.
Kruskal’s algorithm.

J. Kruskal, 1956

Similar to Jarník–Prim, but chooses minimum weight edge, in some cut. The selected edges form a forest until the last step.

\[
\text{MST-K} \ (G, w, r) \\
T = \emptyset \\
\text{for } i = 1 \text{ to } |V| \text{ do} \\
\quad \text{Let } e \in E : \text{with minimum weight among those that do not form a cycle with } T \\
\quad T = T \cup \{e\} \\
\text{end for}
\]
Example
Example

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering

Example

- a
- b
- c
- d
- e
- f
- g
- h

- Edge weights:
  - a-b: 6
  - a-c: 5
  - a-d: 14
  - b-c: 4
  - b-e: 10
  - c-e: 3
  - c-f: 15
  - d-e: 2
  - d-g: 15
  - e-g: 8
  - f-g: 14
  - g-h: 3

Diagram:
- Vertices: a, b, c, d, e, f, g, h
- Edges and weights:
  - a to b: 6
  - a to c: 5
  - a to d: 14
  - b to c: 4
  - b to e: 10
  - c to e: 3
  - c to f: 15
  - d to e: 2
  - d to g: 15
  - e to g: 8
  - f to g: 14
  - g to h: 3

Diagram shows a network with weighted edges.
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering

\[
\begin{array}{cccccc}
 a & b & c & d & e & f \\
 \hline
10 & 6 & 9 & 15 & 3 & 8 \\
5 & 4 & 14 & 2 & 14 & 3 \\
15 & 3 & 8 & 9 & 14 & 10 \\
\end{array}
\]
The problem
Properties
The cut and the cycle properties
A generic algorithm
Prim's algorithm
Kruskal's algorithm
Description
Union-Find implementation
Cost
An application: Clustering

Example

```
 a  b  c
   
 d

 a  b  c  d
   
 f

 e  g

 g  h
```

Cost:

- ab: 6
- ac: 5
- ad: 9
- bc: 4
- bd: 2
- cd: 10
- eg: 3
- fg: 8
- gh: 15
- ge: 14
The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering

Example
Example

The problem

Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm

Kruskal's algorithm

Description

Union-Find implementation

Cost

An application: Clustering
The problem

Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm

Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering

Example

```
\begin{tabular}{cccc}
  \text{a} & \text{b} & \text{c} & \text{d} \\
  14 & 4 & 5 & 9 \\
  \text{e} & \text{f} & \text{g} & \text{h} \\
  3 & 10 & 8 & 15 \\
\end{tabular}
```
Example
Example

The problem
Properties
The cut and the cycle properties
A generic algorithm

Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation
Cost
An application: Clustering

Example

- a
- b
- c
- d
- e
- f
- g
- h

Weights:
- ab: 4
- cd: 2
- ef: 3
- gh: 15
- bc: 5
- fc: 8
- cg: 9
Kruskal’s algorithm: Efficient Implementation

\[ \text{MST-K2} \ (G, w, r) \]
Sort \( E \) by increasing weight
\( T = \emptyset \)
for \( e \in E \) in sorted order do
    if \( e \) does not form a cycle with \( T \) then
        \( T = T \cup \{ e \} \)
    end if
end for
The output is the same as for \textbf{MST-K} but we do not need to examine the remaining edges at intermediate steps.
Kruskal’s algorithm: Implementation

- We have a cost of $O(m \log m)$ to sort the edges. But as $m \leq n^2$, $O(m \log m) = O(m \log n)$.
- We need an efficient implementation of the algorithm selecting an adequate data structure.
- Let us look to some properties of the objects constructed along the execution of the algorithm.
Kruskal evolves by building spanning forests, merging two trees (blue rule) or discarding an edge (red rule) so as to do not create a cycle.

The connectivity relation is an equivalence relation: $u \mathcal{R}_F v$ iff there is a path between $u$ and $v$.

Kruskal, starts with a partition of $V$ into $n$ sets and ends with a partition of $V$ into one set.

$\mathcal{R}$ partition the elements of $V$ in equivalence classes, which are the connected components of the forest.
The Union-Find data structure


- Is a data structure to maintain a dynamic partition of a set.
- One of the most elegant in the algorithmic toolkit.
- It makes possible to design almost linear time algorithms for problems that otherwise would be unfeasible.
- Union-Find is a first introduction to an active research field in algorithmics: Self organizing data structures used in data stream computation.
Union-Find

- Union-Find maintains a partition of a set i.e. a collection of pairwise disjoint sets.
- A set is represented by a rooted tree with labels. The root of the tree the representative of the tree (set).
- Internally a partition is a spanning forest.
Union-Find supports three operations on partitions of a set:

- **MAKESET**(x): creates a new set containing the single element x.
  Creates a tree with only one node, the root, associated with x.

- **UNION**(x, y): Merge the sets containing x and y, by using their union.
  Define how to merge the trees and choose the root of the merged trees.

- **FIND**(x): Return the representative of the set containing x.
  Find the root of the tree containing x and return the associated element.
Warning about UNION operation

- **Warning**: For any $x, y \in S$, we might need to do $\text{UNION}(x, y)$, for $x, y$ that are not representatives. Depending on the implementation this might or might not be allowed.

- To determine the complexity under different implementations, we consider that

  $$\text{UNION}(x, y) = \text{UNION}(\text{FIND}(x), \text{FIND}(y)).$$
Union-Find implementation for Kruskal

\[ \text{MST} \ (G(V, E), w, r), \ |V| = n, |E| = m \]

Sort \( E \) by increasing weight: \( \{e_1, \ldots, e_m\} \)

\( T = \emptyset \)

for all \( v \in V \) do

\hspace{1cm} \text{MAKESET}(v)

end for

for \( i = 1 \) to \( m \) do

\hspace{1cm} Assume that \( e_i = (u, v) \)

\hspace{1.2cm} if \( \text{FIND}(u) \neq \text{Find}(v) \) then

\hspace{2.2cm} \( T = T \cup \{e_i\} \)

\hspace{2.2cm} \text{UNION}(u, v)

\hspace{1.2cm} end if

end for

- Sorting takes time \( O(m \log n) \).
- The remaining part of the algorithm is a sequence of \( n \) MAKESET and \( O(m) \) operations of type FIND/UNION.
An amortized analysis is any strategy for analyzing a sequence of operations on a Data Structure, to provide the "average" cost per operation, even though a single operation within the sequence might be expensive.

An amortized analysis guarantees the average performance of each operation is the worst case on the sequence.

The easier way to think about amortized analysis is to consider total cost of the steps, for a sequence of operations, divided by its size.
(4.6 KT)
For a set with $n$ elements.

- **Using an array holding the representative of each element.**
  - MAKESET and FIND takes $O(1)$
  - UNION takes $O(n)$.

- **Using an array holding the representative, a list by set, and in a UNION keeping the representative of the larger set.**
  - MAKESET and FIND takes $O(1)$
  - any sequence of $k$ UNION takes $O(k \log k)$. 
Complexity of Union Find implementations: Amortized cost

For a set with $n$ elements.

- Using a rooted tree by set, in a UNION keeping the representative of the larger set.
  - MAKESET and UNION takes $O(1)$
  - FIND takes $O(\log n)$.

- Using a rooted tree by set, in a UNION keeping the representative of the larger set, and doing path compression during a FIND.
  - MAKESET takes $O(1)$
  - any intermixed sequence of $k$ FIND and UNION takes $O(k\alpha(n))$.

$\alpha(n)$ is the inverse Ackerman’s function which grows extremely slowly. For practical applications it behaves as a constant.
Union-Find implementation for Kruskal

\[
\text{MST} \ (G(V, E), w, r), \ |V| = n, |E| = m
\]
Sort \( E \) by increasing weight: \( \{e_1, \ldots, e_m\} \)
\( T = \emptyset \)
for all \( v \in V \) do
    MAKESET(\( v \))
end for
for \( i = 1 \) to \( m \) do
    Assume that \( e_i = (u, v) \)
    if \( \text{FIND}(u) \neq \text{Find}(v) \) then
        \( T = T \cup \{e_i\} \)
        UNION(\( u, v \))
    end if
end for
- Sorting take time \( O(m \log n) \).
- The remaining part of the algorithm has cost \( n + O(m\alpha(n)) = O(n + m) \).

But due to the sorting instruction, cost is \( O(n + m \log n) \).
Some applications of Union-Find

- Kruskal’s algorithm for MST.
- Dynamic graph connectivity in networks with a large number of edges.
- Cycle detection in undirected graphs.
- Random maze generation and exploration.
- Strategies for games: Hex and Go.
- Least common ancestor.
- Compiling equivalence statements.
- Equivalence of finite state automata.
Clustering: process of finding interesting structure in a set of data.

Given a collection of objects, organize them into similar coherent groups with respect to some distance function $d(\cdot, \cdot)$.

The distance function not necessarily has to be the physical (Euclidean) distance. The interpretation of $d(\cdot, \cdot)$ is that for any two objects $x, y$, the larger that $d(x, y)$ is, the less similar that $x$ and $y$ are.

If $x, y$ are two species, we can define $d(x, y)$ as the years since they diverged in the course of evolution.
Given a set of data points $\mathcal{U} = \{x_1, x_2, \ldots, x_n\}$ together with a distance function $d$ on $X$, and given a $k > 0$, a $k$-clustering is a partition of $X$ into $k$ disjoint subsets.
The single-link clustering problem

Let $\mathcal{U}$ be a set of $n$ data points, assume $\{C_1, \ldots, C_k\}$ is a $k$-clustering for $\mathcal{U}$. Define the spacing $s$ in the $k$-clustering as the minimum distance between any pair of points in different clusters.

The single-link clustering problem: Given $\mathcal{U} = \{x_1, x_2, \ldots, x_n\}$, a distance function $d$, and $k > 0$, find a $k$-clustering of $\mathcal{U}$ maximizing the spacing $s$.

Notice there are exponentially many different $k$-clusterings of $\mathcal{U}$. 
TrKruskal: An algorithm for the single-link clustering problem

- Represent $\mathcal{U}$ as vertices of an undirected graph where the edge $(x, y)$ has weight $d(x, y)$.
- Apply Kruskal’s algorithm until the forest has $k$ trees.
The problem
Properties
The cut and the cycle properties
A generic algorithm
Prim's algorithm
Kruskal's algorithm

Description
Union-Find implementation

Cost
An application:
Clustering

Complexity and correctness

**Theorem**

*TrKruskal solves the single-link clustering problem in* $O(n^2 \lg n)$

**Proof.**

We have to create a complete graph and sort the $n^2$ edges. Thus TrKruskal has cost $O(n^2 \lg n)$

**Correctness**

Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be the $k$-clustering produced by TrKruskal, and let $s$ be its spacing.

Assume there is another $k$-clustering $\mathcal{C}' = \{C'_1, \ldots, C'_k\}$ with spacing $s'$ and s.t. $\mathcal{C} \neq \mathcal{C}'$. We must show that $s' \leq s$. 
Complexity and correctness

- If $C \neq C'$, then $\exists C_r \in C$ s.t. $\forall C_t' \in C', C_r \not\subseteq C_t'$.
- Then $\exists x, y \in C_r$ s.t. $x \in C_a', y \in C_b'$ and $a \neq b$.
- $\exists$ a path $x \sim y$ in $C_r$ contained in the spanning tree $T_r$ obtained by TrKruskal for $C_r$.
- Then, $\exists (x', y') \in E(T_r)$ with $x' \in C_a'$ and $y' \in C_b'$, so $s' \leq d(x', y')$.
- As $(x', y') \in E(T_r)$, $d(x', y') \leq s$ and $s' \leq s$.

End Proof