Problem

- A town has a set of houses and a set of roads.
- A road connects 2 and only 2 houses.
- A road connecting houses $u$ and $v$ has a repair cost $w(u, v)$.

**Goal:** Repair enough (and no more) roads such that:
1. everyone stays connected: can reach every house from all other houses, and
2. total repair cost is minimum.

Model as graph

- Undirected graph $G = (V, E)$.
- Weight $w(u, v)$ on each edge $(u, v) \in E$.
- Find $T \subseteq E$ such that:
  1. $T$ connects all vertices ($T$ is a spanning tree),
  2. $w(T) = \sum_{(u, v) \in T} w(u, v)$ is minimized.

**Generic Algorithm**

- Initially, $A$ has no edges.
- Add edges to $A$ and maintain the loop invariant: “$A$ is a subset of some MST”.

\[
\begin{align*}
A & \leftarrow \emptyset; \\
\text{while } A \text{ is not a spanning tree do} & \\
\text{find a edge } (u, v) \text{ that is safe for } A; & \\
A & \leftarrow A \cup \{(u, v)\} \\
\text{return } A
\end{align*}
\]

- **Initialization:** The empty set trivially satisfies the loop invariant.
- **Maintenance:** We add only safe edges, $A$ remains a subset of some MST.
- **Termination:** All edges added to $A$ are in an MST, so when we stop, $A$ is a spanning tree that is also an MST.
What is a safe edge?

Intuitively: Is (c,f) safe when A = ∅?

• Let S be any set of vertices including c but not f.
• There has to be one edge (at least) that connects S with V − S.
• Why not choosing the one with the minimum weight?

Theorem 1: Let (S, V-S) be any cut that respects A, and let (u, v) be a light edge crossing (S, V-S). Then, (u, v) is safe for A.

Proof:
Let T be a MST that includes A.

Case 1: (u, v) in T. We’re done.
Case 2: (u, v) not in T. We have the following:

(x, y) crosses cut. Let T’ = T − ((x, y)] ∪ [(u, v)). Because (u, v) is light for cut, w(u, v) ≤ w(x, y). Thus, w(T) = w(T) + w(u, v) ≤ w(T’).

Hence, T’ is also a MST. So, (u, v) is safe for A.

In general, A will consist of several connected components.

Corollary: If (u, v) is a light edge connecting one CC in (V, A) to another CC in (V, A), then (u, v) is safe for A.

Kruskal’s Algorithm

1. Starts with each vertex in its own component.
2. Repeatedly merges two components into one by choosing a light edge that connects them (i.e., a light edge crossing the cut between them).
3. Scans the set of edges in monotonically increasing order by weight.
4. Uses a disjoint-set data structure to determine whether an edge connects vertices in different components.
Kruskal’s complexity

- Initialize A: \( O(V) \)
- First for loop: \( |V| \) MAKE-SETS
- Sort \( E \): \( O(E \lg E) \)
- Second for loop: \( O(E) \) FIND-SETS and UNIONS

Assuming union by rank and path compression:
\( O((V+E)\alpha(V)+E\lg E) \)

- Since \( G \) is connected, \( |E| \geq |V| - 1 \) \( \Rightarrow O(E \alpha(V)) + O(E \lg E) \).
- \( \alpha(|V|) = O(\lg V) = O(\lg E) \).
- Therefore, total time is \( O(E \lg E) \).

\[ \Rightarrow O(E \lg V) \text{ time} \]

Prim’s Algorithm

1. Builds one tree, so \( A \) is always a tree.
2. Starts from an arbitrary "root" \( r \).
3. At each step, adds a light edge crossing \( (V_A, V - V_A) \) to \( A \).
- Where \( V_A \) = vertices that \( A \) is incident on.

Finding a light edge

1. Uses a priority queue \( Q \) to find a light edge quickly.
2. Each object in \( Q \) is a vertex in \( V - V_A \).
3. Key of \( v \) is minimum weight of any edge \( (u, v) \), where \( u \in V_A \).
4. Then the vertex returned by Extract-Min is \( v \) such that there exists \( u \in V_A \) and \( (u, v) \) is a light edge crossing \( (V_A, V - V_A) \).
5. Key of \( v \) is \( \infty \) if \( v \) is not adjacent to any vertex in \( V_A \).

Example of Prim’s Algorithm

Complexity:
Using binary heaps: \( O(E \lg V) \).
Initialization: \( O(V) \).
Building initial queue: \( O(V) \).
\( V \) Extract-Min: \( O(V \lg V) \).
\( E \) Decrease-Key: \( O(E \lg V) \).

Using Fibonacci heaps:
\( O(E + V \lg V) \).

\( \Downarrow \) decrease-key operation

Note: \( A = \{(v, \pi[v]) : v \in V - \{r\} \cap Q \} \).