Data Structures

Minimum Spanning Trees
Minimum Spanning Trees

• 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
    • Everyone stays connected.
      – Can reach every house from all other houses.
    • Total repair cost is minimum.
Model as a 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**), and
  2. $w(T) = \sum_{(u, v) \in T} w(u, v)$ is minimized.

A spanning tree whose weight is minimum over all spanning trees is called a **minimum spanning tree**, or MST.

Example of such a graph *edges in MST are shaded*:

In this example, there is more than one MST. Replace edge $(e, f)$ by $(c, e)$. Get a different spanning tree with the same weight.
Some Properties of an MST

- It has $|V| - 1$ edges.
- It has no cycles.
- It might not be unique.
Growing a Minimum Spanning Tree

• Building up the solution
  – Start with an initially empty set of edges $A$, subset of any MST.
  – Repeatedly add safe edges to $A$, until it becomes an MST, where
    • An edge $(u, v)$ is safe for $A$ if and only if $A \cup \{(u, v)\}$ is also a subset of some MST.

```plaintext
genericMST (G, w)
A = ∅
while (A is not a spanning tree) do
  find an edge (u, v) that is safe for A
  A = A \cup \{(u, v)\}
return A
```
Finding a Safe Edge

• Let $S \subseteq V$ and $A \subseteq E$.

• A cut $(S, V - S)$ is
  – a partition of vertices into disjoint sets $S$ and $V - S$.

• Edge $(u, v) \in E$ crosses cut $(S, V - S)$
  – If one endpoint is in $S$ and the other is in $V - S$.

• A cut respects $A$
  – Iff no edge in $A$ crosses the cut.

• An edge is a light edge crossing a cut
  – If and only if its weight is minimum over all edges crossing the cut.
  – For a given cut, there can be > 1 light edge crossing it.
Finding a Safe Edge

• Theorem:

  Let

  • $A$ be a subset of edges of some MST,
  • $(S, V - S)$ be a cut that respects $A$.
  • $(u, v)$ be a light edge crossing $(S, V - S)$.

  Then

  – $(u, v)$ is safe for $A$. 

Finding a Safe Edge

• Proof:
  – Let T be an MST that includes A.
  – If T contains \((u, v)\), done.
  – So now assume that T does not contain \((u, v)\).
  – Since T is a tree, it contains a unique path \(p\) between \(u\) and \(v\).
  – Path \(p\) must cross the cut \((S, V - S)\) at least once.
  – Let \((x, y)\) be an edge of \(p\) that crosses the cut.
  – From how we chose \((u, v)\), must have \(w(u, v) \leq w(x, y)\).
  – Thus, \(T' = T - \{(x, y)\} \cup \{(u, v)\}\) is MST, and \(A \cup \{(u, v)\} \subseteq T'\).
  – Hence \((u, v)\) is safe for A.
Kruskal’s Algorithm

- A is a forest of sub-trees of the MST.
- Initially, each sub-tree is a single vertex (no edges).
- Any safe edge merges two of these sub-trees into one.
- Since an MST has exactly |V| – 1 edges, the for loop iterates |V| – 1 times.
- How to find safe edges?
  - Sort the edges in E by their weights in ascending order.
  - Run over them in that order.
  - If an edge (v, u) merge two sub-trees, it safe, since it is a light edge that cross the cut (S, V – S) that respects A, where S consists of the vertices of one of the two sub-trees in hand.

```
genericMST (G,w)
A = ∅
while (A is not a spanning tree) do
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```
Kruskal’s Algorithm

\[ \text{mst} \ (V, E, w) \]
\[ A = \emptyset \]
\[ \text{for each vertex } v \in V \ \text{do} \]
\[ \text{makeSet}(v) \]
\[ \text{sort } E \text{ into nondecreasing order by weight } w \]
\[ \text{for each } (u, v) \text{ taken from the sorted list do} \]
\[ \text{if } \text{findSet}(u) \neq \text{findSet}(v) \text{ then} \]
\[ A = A \cup \{(u, v)\} \]
\[ \text{union}(u, v) \]
\[ \text{return } A \]
Complexity

- **Initialize A**: $O(1)$
- **First for loop**: $V$ makeSets + $V$
- **Sort E**: $O(E \lg E)$
- **Second for loop**: $O(E)$ findSets and $O(V)$ unions + $E$
- **Total**: $O((V + E) \alpha(V)) + O(E \lg E)$.

- Since $G$ is connected, $E \geq V - 1 \implies$ total = $O(E \alpha(V)) + O(E \lg E))$.
- Since $\alpha(V) = O(\lg V) = O(\lg E) \implies$ total = $O(E \lg E)$.
- Since $E \leq V^2 \implies$ $\lg E = O(2 \lg V) = O(\lg V) \implies$ total = $O(E \lg V)$.

- If edges are already sorted, total = $O(E \alpha(V))$, which is almost linear.
Prim’s Algorithm

• Builds one tree.
  – So A is always a tree.
• Starts from an arbitrary “root” r.
• At each step, find a light edge crossing cut \((V_A, V - V_A)\), where \(V_A = \text{vertices that of A is incident on}\)
• Add this edge to A.
How to Find the Light Edge Quickly?

• Use a priority queue for the vertices in $V - V_A$.
• Key of $v$ is minimum weight of any edge $(u, v)$, where $u \in V_A$.
• Then the vertex returned by extractMin is $v$ such that there exists $u \in V_A$ and $(u, v)$ is light edge crossing $(V_A, V - V_A)$.
• Key of $v$ is $\infty$ if $v$ is not adjacent to any vertices in $V_A$. 
Prim’s Algorithm

```plaintext
prim (V, E, w, r )
  for each u ∈ V do
    u.key = ∞
    u.π = null
  r.key = 0
  build a priority queue Q on V
while Q ≠ ∅ do
  u = extractMin(Q)
  for each v ∈ Adj[u] do
    if v ∈ Q & w(u, v) < v.key then
      v.π = u
      decreaseKey(Q, v , w(u, v))
```

Complexity

• Suppose Q is a **binary heap**.
  - **Initialize Q and first for loop**: \( O(V) \)
  - **Decrease key of r**: \( O(\log V) \)
  - **while loop**: \(|V|\) extractMin calls \( \Rightarrow O(V \log V) \)
    \(|E|\) decreaseKey calls \( \Rightarrow O(E \log V) \)
  - **Total**: \( O(E \log V) \)

• Suppose Q is a **Fibonacci heap**:
  - We could do decreaseKey in \( O(1) \) amortized time.
  - \(|E|\) decreaseKey calls take \( O(E) \) time altogether \( \Rightarrow \)
  **total time** becomes \( O(V \log V + E) \).