Minimum Spanning Tree

- G=(V,E) is an undirected graph, where V is a set of nodes and E is a set of possible interconnections between pairs of nodes.
- For each edge (u,v) in E, we have a weight W(u,v).
- Find an acyclic subset T of E, that connects all the vertices and whose total weight is minimum.
A Spanning Trees

Quiz: Are minimum spanning trees unique?

Kruskal’s Algorithm

- Consider v isolated trees in the forest. Each initially with only one node.
- Pick the shortest path that connects two trees in the forest.
- In other words, select a least-cost edge that does not result in a cycle when added to a set of already selected edges.
Algorithm

**MST-Kruskal (G, w)**

1. \( A \leftarrow \emptyset \)
2. for each vertex \( v \in V[G] \)
   3. do \( \text{MAKE-SET}(v) \)
4. sort the edges of \( E \) by nondecreasing weight \( w \)
5. for each edge \((u, v) \in E\), in order by nondecreasing weight
   6. do if \( \text{FIND-SET}(u) \neq \text{FIND-SET}(v) \)
      7. then \( A \leftarrow A \cup \{(u, v)\} \)
     8. \( \text{UNION}(u, v) \)
9. return \( A \)

**Example:** Kruskal’s Algorithm

![Graph with edges and weights](image)

- **Initialize V forests.**
- **Sort edges.**
- **Make sure two ends are in two trees. No cycle.**
- **Merge the trees in the forest.**
- **If no cycle, add the edge in the spanning tree set.**
Example: Kruskal’s Algorithm

Example: Continued..
Proof of Correctness of an Algorithm

Correctness of Krsukal’s Algorithm

• Let T be the tree found by Kruskal’s algorithm.
• Let U be the actual minimum spanning tree.
• We will prove cost of T = cost of U.
• Do you agree?
  – T and U and all spanning trees must have exactly V-1 edges.
  – If, k (k>0) number of edges in U are not in T, then exactly k number of edges in T must not be in U.

• We will one by one substitute a unique edge of U by unique edge of T to prove that the cost does not change.
Correctness of Kruskal’s Algorithm (contd..)

- Let e be the least-cost edge in T that is not in U.
- Add e to U.
  - It must create a cycle.
  - There must be an edge f in this cycle which was not in T.
- Take it out. The new spanning tree has cost
  \[ V = U + \{e\} - \{f\} \]
- Can \(\{e\} < \{f\}\)?
  - No because, then U cannot be minimum spanning tree.
- Can \(\{e\} > \{f\}\)?
  - No because, then f will be included by Kruskal’s greedy scheme before e. That did not happen!
- Therefore \(\{e\} = \{f\}\)
- Therefore \(T = U\)

Correctness of Kruskal’s algorithm

Two solution’s with same cost.

\[ V = U + \{b,c\} - \{a,h\} \]
Complexity of Kruskal’s Algorithm

```
MST-KRUSKAL(G, w)
1. A ← ∅
2. for each vertex v ∈ V[G]
   do MAKE-SET(v)
4. sort the edges of E by nondecreasing weight w
5. for each edge (u, v) ∈ E, in order by nondecreasing weight
   do if FIND-SET(u) ≠ FIND-SET(v)
      then A ← A ∪ {(u, v)}
6. UNION(u, v)
9. return A
```

• 1-3: Initialization O(v)
• 4: sorting O(E log E)
• 5: E iterations.
• 6: Each FIND-SET is O(log E) total cost= 2E. O(log E)
• 7: 2E
• 8: UNION is at most V-1
• Overall complexity is O(V+E log E)

Prim’s Algorithm

• Like Kruskal’s, but, start with any node.
• Extend the tree to the closest node!
Prim’s Algorithm

**MST-Prim(G, w, r)**

1. \( Q \leftarrow V[G] \)
2. for each \( u \in Q \)
3. do \( \text{key}[u] \leftarrow \infty \)
4. \( \text{key}[r] \leftarrow 0 \)
5. \( \pi[r] \leftarrow \text{NIL} \)
6. while \( Q \neq \emptyset \)
7. do \( u \leftarrow \text{Extract-Min}(Q) \)
8. for each \( v \in \text{Adj}[u] \)
9. do if \( v \in Q \) and \( w(u, v) < \text{key}[v] \)
10. then \( \pi[v] \leftarrow u \)
11. \( \text{key}[v] \leftarrow w(u, v) \)

- Add the vertices in a Queue
- Key[\( u \)] is the cost of reaching vertex \( u \) from current tree set.
- Start from any node \( r \). Its cost is zero. \( P[\( u \)] \) is the root of \( u \).
- Take the vertex closest to the tree.
- For each node adj to \( u \), but not in spanning tree, update the reaching cost.

**Example: Prim’s Algorithm**

```
Example:
```

![Graph](image)

(a) \( a \)
Complexity of Prim’s Algorithm

1. Initialization $O(v)$
2. Loop executes $V$ times.
3. Each EXTRACT-MIN is $O(\log V)$. Total $O(V \log V)$.
4. Loop 8-11 executes $E$ times.
5. Membership can be tested in constant time.
6. $V$ have to be deleted from $Q$ (not shown): $O(\log V)$
7. Total: $O(V \log V + E \log V)$

Shortest Path
Shortest Path

Given a directed graph in which each edge has a nonnegative weight or cost, find a path of least total weight from a given vertex, called the source, to every other vertex in the graph.

- Other Variants:
  - Single Destination shortest-path problem.
  - Single-pair shortest path problem.
  - All pairs shortest-paths problem.

Greedy Method
(Dijkstra’s Algorithm)

- We keep a set $S$ of vertices whose closest distances to the source, Vertex 0, are known and add one vertex to $S$ at each stage.
- We maintain a table $D$ that gives, for each vertex $v$, the distance from 0 to $v$ along a path all of whose vertices are in $S$, except possibly the last one.
- To determine what vertex to add to $S$ at each step, we apply the greedy criterion of choosing the vertex $v$ with the smallest distance recorded in the table $D$, such that $v$ is not already in $S$. 
Example

Example (contd..)
Algorithm

- 1. INITIALIZE_SINGLE-SOURCE(G,s)
- 2. S = EMPTY.
- 3. Q = V[G]
- 4. While Q not EMPTY
- 5. u = EXTRACT-MIN(Q)
- 6. Add u in S
- 7. For each vertex v adjacent to u
- 8. Do Update cost
   - if D[v] > d[u] + w[u,v]
   - then D[v] = d[u] + w[u,v]
   - GoFrom[v] = u

Proof of Correctness

- Let us assume the path through another node x, which is not yet included in S to v is closer.
- Then D[x] must be smaller than D[v], but in that case x should already be included in S!
Complexity

- Each EXTRACT-MIN takes $O(V)$.
- Each time at least one vertex will be added.
- Therefore it can take at most $V$ iterations.
- Step 5 is $O(V^2)$
- On the other hand, in steps 4-8 each path will be processed only once.
- Thus the complexity is $O(V^2+E)$.

Bellman-Ford Algorithm

- It can solve the shortest-path problem, even if there are negative weighted links.
- What if there is a negative weighted cycle?
- Its complexity is $O(V.E)$.