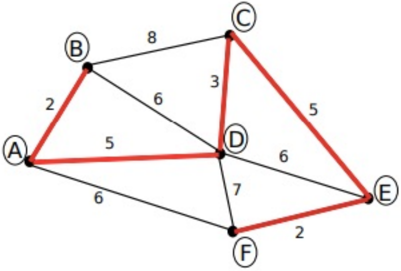
CS660: Algorithms - Lecture 9

Instructor: Hoa Vu

San Diego State University

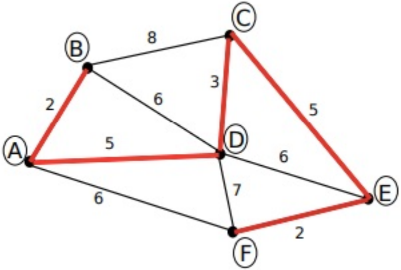
Minimum Spanning Tree - PRIM's algorithm

- Given a weighted connected graph. Find a tree that covers every vertex such that the total edge weights is minimized.



Minimum Spanning Tree

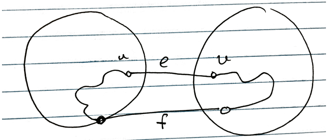
- Greedy algorithm: pick an arbitrary node as the starting tree. Add the minimum weight edge with one end point to the tree until the tree



covers all vertices.

Minimum Spanning Tree - PRIM's algorithm

- For simplicity, assume that the weights are distinct (note that the proof still goes through if they are not just a bit more wordy).
- Let $e = uv$ be the first edge added by the algorithm that is not in the minimum spanning tree (let the tree at that point be T and the remaining graph be $G \setminus T$). In particular, T is not part of any MST, but $T \setminus (uv)$ is part of some MST.
- Since u and v must be connected in the MST, there must be a path from u to v. Hence, there must be an edge f between a vertex in T and a vertex in $G \setminus T$. If we remove f from the MST, and add e to the MST, we get a spanning tree with a smaller weight (since the algorithm picked e over f because $w(e) < w(f)$) which is a contradiction.

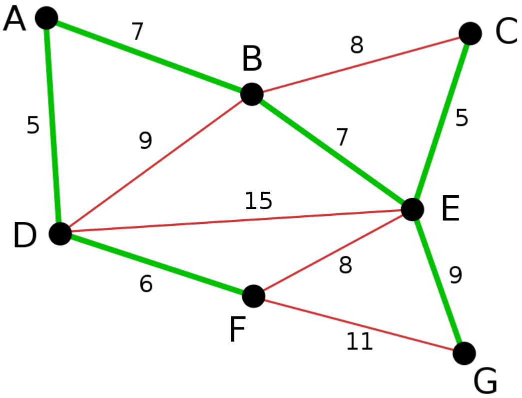


Minimum Spanning Tree

- Prim's algorithm can be implemented to run in $O(|E| + |V| \log |V|)$ time using Fibonacci heap (read 7.4).

Minimum Spanning Tree - Kruskal's algorithm

- Scan through the edges. Among the safe edges (not creating a cycle), add the one with the smallest weight.



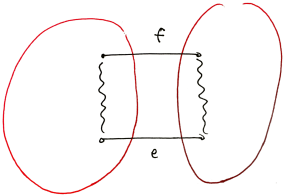
-

Minimum Spanning Tree - Kruskal's algorithm

- Proof of correctness? For simplicity, assume distinct weights.
- Consider the connected components (a single vertex is also a connected component). An edge is safe if and only if it connects two vertices from different connected components.
- Let's prove via induction. We will prove that at any step, the forest we constructed so far is a subgraph of an MST.
- The base case is clear: the empty graph is a subgraph of a MST.
- Suppose the forest F we construct so far is a subgraph of some MST T. Consider the next edge e that the algorithm adds. Suppose $F + e$ isn't a subgraph of any MST. Then, $T + e$ must have a cycle.
- In the cycle, there must be an edge f that is not picked by the algorithm. We know that since the algorithm picks e over f, $w(e) < w(f)$.

Minimum Spanning Tree - Kruskal's algorithm

- Suppose the forest F we construct so far is a subgraph of some MST T. Consider the next edge e that the algorithm adds. Suppose $F + e$ isn't a subgraph of any MST. Then, $T + e$ must have a cycle.
- In the cycle, there must be an edge f that is not picked by the algorithm. We know that since the algorithm picks e over f, $w(e) < w(f)$.



-
- $T - f + e$ is a spanning tree with smaller weight which is a contradiction. Hence, $F + e$ must be a subgraph of some MST.

Minimum Spanning Tree - Kruskal's algorithm

- How to implement Kruskal's algorithm?

```
KRUSKAL(V, E):
    sort E by increasing weight
    F ← (V, ∅)
    for each vertex v ∈ V
        MAKESET(v)
    for i ← 1 to |E|
        uv ← ith lightest edge in E
        if FIND(u) ≠ FIND(v)
            UNION(u, v)
            add uv to F
    return F
```

-

Minimum Spanning Tree - Kruskal's algorithm

- How to implement Kruskal's algorithm?
- Sort edges by weights. $O(|E|\log|E|)$ time.
- We use the following data structure.
- $\mathit{MakeSet}(v)$: create a set containing only vertex v.
- $\mathit{Find}(v)$: Return an identifier of the set containing v.
- $\mathit{Union}(u, v)$: Replace the sets containing u and v with their union.

Minimum Spanning Tree - Kruskal's algorithm

- *MakeSet*(v): create a set containing only vertex v. This take constant $O(1)$ time. For each vertex v, let $S[v]$ be the identifier of the set containing v. We can use the current largest identifier X plus 1 as the identifier of a new set containing v. Particularly, $S[v] \leftarrow X + 1$ and $X \leftarrow X + 1$.
- *Find*(v): Return an identifier of the set containing v. Return $S[v]$.

Minimum Spanning Tree - Kruskal's algorithm

- For each set (component), we have a spanning tree that touches every vertex in that component.
- $Union(u, v)$: Replace the sets containing u and v with their union. If the identifier of the set containing u is larger than the identifier of the set containing v, i.e., $|set(S[u])| > |set(S[v])|$, then we put every element in $S[v]$ to $S[u]$. This requires a depth-first-search through each vertex in $S[v]$ in the spanning tree $T[v]$ of $S[v]$ and reassign the set identifier of those nodes to $S[u]$. We need to traverse through each vertex in $T[v]$ using DFS and update the set containing it to $S[u]$.
- Consider a node v, each time it is assigned to a new set, the component containing v doubles in size. Hence, we traverse and update the set containing v at most $O(\log |V|)$ times during the algorithm.
- Finally, add uv and $T[v]$ to the spanning tree $T[u]$.