Lecture 11
Minimum Spanning Tree
and
Greedy Algorithms

Definitions
- Spanning tree: given a connected, undirected graph G=(V,E), a spanning tree is a subgraph of G that is an undirected tree and contains all the vertices of G.
- Minimum spanning tree (MST): is a spanning tree with minimum weight. The weight of a subgraph is the sum of the weights of the edges in the subgraph.

How to find MST?
- Using depth-first or breadth-first search to traverse the graph will yield a spanning tree, but the found spanning tree is not guaranteed to be a MST.
- Need a different scheme to traverse the graph

Optimization problem & Greedy approach
- Pick a starting point randomly
- "grow" step by step, each step is the best among all possible choices
- Stop when a stopping criterion is satisfied

Note: being greedy in a short term may not lead to the overall best solution

Growing a MST

Definitions
- Tree vertices: vertices that are in the tree constructed so far
- Fringe vertices: not in the tree, but adjacent to some tree vertices
- Unseen vertices: all others.
primMST(G,n)

1. Initialize all vertices as unseen
2. Select an arbitrary vertex s to start the tree, mark s as tree vertex
3. Mark all vertices adjacent to s as fringe
4. While there are fringe vertices
   1. Select an edge of minimum weight between a tree vertex t and a fringe vertex v
   2. Mark v as tree and add edge (tv) to the tree
5. Mark all unseen vertices adjacent to v as fringe

Definitions

- Minimum spanning tree property: Let T be any spanning tree of a connected, weighted graph G. For any edge uv of G that is not in T, if uv is added to T it creates a cycle and uv is a maximum-weight edge on that cycle, then T has the minimum spanning tree property.

Correctness of Prim’s MST algorithm

- Each time an edge from a tree vertex to a fringe vertex is added into the tree, so never is a cycle created.
- All vertices will be covered to the tree eventually.
- Therefore the final tree is a spanning tree.
- It is also a minimum spanning tree, because
  - At each step, the so far constructed tree has the MST property in its induced graph of G.

Theorem 8.2: In a connected, weighted graph G = (V,E,W), a spanning tree T is a MST if T has the MST property.

Proof:

- (If) A MST T must have the MST property. Otherwise there exits an edge uv not in T, and uv is not the maximum weight edge in the cycle formed by adding uv to T. The maximum weight edge, say xy, is in T. Removing xy creates a new spanning tree T’ whose weight is less than T’s. This contradicts that T is a MST.

- (Only if) A spanning tree T has the MST property, then T must be a MST. Let T_k be an arbitrary minimum spanning tree of G. If T_k and T differ by only one edge, say xy, then W(xy) < W(k). If T_k + xy is in T, a cycle C is created and uv must be on C. (why?) Let's call T + xy = T’. TC becomes T_k if uv is removed, i.e. the cycle C is broken.

Corollary: If all edge weights are distinct, the MST is unique.
It is impossible to have edge \( r t \) such that \( W(rt) > W(vi) \)

Suppose \( t \) is added in later than \( r \), then Prim algorithm will add edge \((uv)\) instead of \((rt)\), since \( v \) is in the fringe and has an edge of smaller weight.

**Kruskal MST Algorithm**

1. Initialize \( F \) // a forest of trees
2. Build a minimizing priority queue \( Q \) of edges of \( G \), prioritized by weight
3. Initialize a union-find structure, sets, in which each vertex of \( G \) is in its own set.
4. while \( Q \) is not empty
5. \( \text{vewEdge} = \text{Extract-Min}(Q) \)
6. \( \text{int vSet} = \text{find}(\text{sets, vewEdge.from}) \)
7. \( \text{int wSet} = \text{find}(\text{sets, vewEdge.to}) \)
8. do if \((vSet \neq wSet) \) // make sure \( v \) and \( w \) not in the same tree any?
9. then Add \( \text{vewEdge} \) to \( F \)
10. union (sets, vSet, wSet)
11. return \( F \)

**NOTE:** if there are degeneracy in edge weights, MST will not be unique, depending on how ties are resolved in the priority queue.

**Managing the Fringe with a priority queue**

1. \( \text{primMST}(G, r) \)
2. for each \( u \in Q \)
3. \( \text{key}[u] \leftarrow -\infty \)
4. \( \text{P}[u] \leftarrow \text{nil} \)
5. \( Q \leftarrow V(G) \)
6. while \( Q \) is not empty
7. \( u \leftarrow \text{Extract-Min}(Q) \)
8. for each \( v \in \text{Adj}[u] \)
9. if \( v \in Q \) and \( w(u,v) < \text{key}[v] \)
10. then \( \text{P}[v] \leftarrow u \)
11. \( \text{key}[v] \leftarrow w(u,v) \)

Total running time: \( O(V \lg V + E \lg V) \)

**Kruskal algorithm: example**

**Time analysis**

- Initialization: \( O(V) \)
- Deleting all edges from the queue: \( \Theta(E \log E) \)
- Find called \( 2|E| \) times, union called \( |V| \) times, cost is \( O(E \log^*(E)) \)

Total worst-case time: \( \Theta(E \log E) \)