CSE 100
Minimum Spanning Trees
Spanning trees

• We will consider spanning trees for *undirected* graphs
• A spanning tree of an undirected graph $G$ is an undirected graph that...
  • contains all the vertices of $G$
  • contains only some edges of $G$
  • has no cycles
  • is connected

• A spanning tree is called “spanning” because it connects all the graph’s vertices

• A spanning tree is called a “tree” because it has no cycles (recall the definition of *cycle* for undirected graphs)

• What is the root of the spanning tree?
  • you could pick any vertex as the root; the vertices adjacent to that one are then the children of the root; etc.
Spanning trees: examples

• Consider this undirected graph $G$: 

```
V0 ─ V1
  |   |
V2 ─ V3 ─ V4
  |
V5 ─ V6
```
Is this graph a spanning tree of G?

A. Yes
B. No
Spanning tree? Ex. 2

Is this graph a spanning tree of G?

A. Yes
B. No
Spanning tree? Ex. 3

Is this graph a spanning tree of G?

A. Yes
B. No
Minimum Spanning tree: Spanning tree with minimum total cost

Is this graph a minimum spanning tree of G?

A. Yes
B. No
Minimum spanning trees in a weighted graph

• A single graph can have many different spanning trees

• They all must have the same number of edges, but if it is a weighted graph, they may differ in the total weight of their edges

• Of all spanning trees in a weighted graph, one with the least total weight is a minimum spanning tree (MST)

• It can be useful to find a minimum spanning tree for a graph: this is the least-cost version of the graph that is still connected, i.e. that has a path between every pair of vertices

• How to do it?
Prim’s MST Algorithm

- Start with any vertex and grow like a mold, one edge at a time
- Each iteration choose the cheapest crossing edge
Running time of Naïve Implementation of Prim’s MST Algorithm

- Denote set of all vertices in MST as $X$
- Denote set of all edges in MST as $T$

- Initialize: Pick a random vertex $s$,
  $X = \{s\}$, $T = \{\}$

- While $X$ not equal to $V$
  - For all edges crossing $X$ and $V-X$, choose the one with the minimum edge weight i.e.
    $$(u^*, v^*) = \arg \min_{uv} l_{uv}$$

    $$X = X \cup v^*$$, $$T = T \cup (u^*, v^*)$$

Ref: Tim Roughgarden (stanford)
Fast Implementation of Prim’s MST Algorithm

1. Create an empty graph $T$. Initialize the vertex vector for the graph. Set all “done” fields to false. Pick an arbitrary start vertex $s$. Set its “done” field to true. Iterate through the adjacency list of $s$, and put those edges in the priority queue.

2. While the priority queue is not empty:
   - Remove from the priority queue the edge $(v, w, \text{cost})$ with the smallest cost.
   - Is the “done” field of the vertex $w$ marked true?
     - If Yes: this edge connects two vertices already connected in the spanning tree, and we cannot use it. Go to 2.
     - Else accept the edge:
       - Mark the “done” field of vertex $w$ true, and add the edge $(v, w)$ to the spanning tree $T$.
       - Iterate through $w$’s adjacency list, putting each edge in the priority queue.

   - The resulting spanning tree $T$ is the minimum spanning tree.
Finding a minimum spanning tree: Prim’s algorithm

• As we know, minimum weight paths from a start vertex can be found using Djikstra’s algorithm

• At each stage, Djikstra’s algorithm extends the best path from the start vertex (priority queue ordered by total path cost) by adding edges to it

• To build a minimum spanning tree, you can modify Djikstra’s algorithm slightly to get Prim’s algorithm

• At each stage, Prim’s algorithm adds the edge that has the least cost from any vertex in the spanning tree being built so far (priority queue ordered by single edge cost)

• Like Djikstra’s algorithm, Prim’s algorithm has worst-case time cost $O(|E| \log |V|)$

• We will look at another algorithm: Kruskal’s algorithm, which also is a simple greedy algorithm

• Kruskal’s has the same big-O worst case time cost as Prim’s, but in practice it can be made to run faster than Prim’s, if efficient supporting data structures are used
Weighted minimum spanning tree: Kruskal’s algorithm

• Prim’s algorithm starts with a single vertex, and grows it by adding edges until the MST is built: it builds the MST ‘top-down’

• Kruskal’s algorithm starts with a forest of single-node trees (one for each vertex in the graph) and joins them together by adding edges until the MST is built; it builds the MST ‘bottom-up’

Why Kruskal?
Kruskal’s algorithm:
Naïve Implementation of Kruskal’s algorithm:

1. Sort edges in increasing order of cost

2. Set of edges in MST, T=

3. For $i = 1$ to $|E|

   If $T \cup \{ e_i \}$ has no cycles

   Add $e_i$ to $T$

Ref: Tim Roughgarden (stanford)
Running Time of Naïve Implementation of Kruskal’s algorithm:

1. Sort edges in increasing order of cost
2. Set of edges in MST, $T = \{\}$
3. For $i = 1$ to $|E|$
   
   If $T \cup \{e_i\}$ has no cycles
   
   Add $e_i$ to $T$

Ref: Tim Roughgarden (Stanford)
Towards a fast implementation for Kruskal’s Algorithm

• What is the work that we are repeatedly doing with Kruskal’s Algo?
Towards a fast implementation for Kruskal’s Algorithm

• What is the work that we are repeatedly doing with Kruskal’s Algo?
  – Checking for cycles: Linear with BFS, DFS
  – Union-Find Data structure allows us to do this in near constant time! (Next time)
Graph algorithm time costs: a summary

- The graph algorithms we have studied have fast algorithms:
  - Find shortest path in unweighted graphs
    - Solved by basic breadth-first search: $O(|V|+|E|)$ worst case
  - Find shortest path in weighted graphs
    - Solved by Dijkstra’s algorithm: $O(|E| \log|V|)$ worst case
  - Find minimum-cost spanning tree in weighted graphs
    - Solved by Prim’s or Kruskal’s algorithm: $O(|E| \log|V|)$ worst case

- The “greedy” algorithms used for solving these problems have polynomial time cost functions in the worst case
  - since $|E| \leq |V|^2$, Dijkstra’s, Prim’s and Kruskal’s algorithms are $O(|V|^3)$

- As a result, these problems can be solved in a reasonable amount of time, even for large graphs; they are considered to be ‘tractable’ problems

- However, many graph problems do not have any known polynomial time solutions...!