CS16 Section 8

Monday April 6 - Wednesday April 8
Agenda

1. Icebreakers
2. Mini-Assignment
3. Topsort
4. Graph algorithms: Shortest Path and MST
Mini Assignment - Shortest Path
Mini Assignment - Topsort

1. Topological sort is an algorithm that only works on directed acyclic graphs and provides a valid ordering of vertices in the DAG, ensuring that for each vertex, all of its prerequisites come before it in the ordering.
   a. Can be used to model situations in which completing certain things depend on completing other things, such as course prerequisites.

2. Topsort iterates over vertices in a DAG. If a vertex has no prerequisites (if it is a source), we visit it. After visiting a vertex, we remove all of its outgoing edges because the prerequisite they represent has been satisfied. This will create new source nodes that we then visit.
   a. NOTE: What options do you have if you can’t alter the graph?

3. Limitations - topsort can only be used on DAGs. Cycles will mess up the algorithm.
Topsort Review
Topsort Graph A
Topsort Graph B
Shortest Distance and MSTs
Shortest Distance vs. MST

1. Shortest Distance Graph:
   a. Shortest path from a specified start vertex to every other vertex
   b. Minimizes this path weight
   c. Have to specify a starting vertex

2. Minimum Spanning Tree:
   a. A spanning tree with minimum total edge weight
   b. Every vertex is reached in the tree
   c. Minimizes the sum of all edges in the graph
   d. Do not specify a starting vertex
Example Graph C
function dijkstra(G, s):
    // Input: A graph G with vertices V, and a start vertex s
    // Output: Nothing
    // Purpose: Decorate nodes with shortest distance from s
    for v in V:
        v.dist = infinity
        v.prev = null
    s.dist = 0

    PQ = PriorityQueue(V)  // Depends on PQ implementation!
    while PQ not empty:
        u = PQ.removeMin()  // Depends on PQ implementation!
        for all edges (u, v):
            if v.dist > u.dist + cost(u, v):
                v.dist = u.dist + cost(u,v)
                v.prev = u
                PQ.replaceKey(v, v.dist)  // Depends on PQ implementation!
Dijkstra: Review

- **General Idea:** Given a starting node, finds the shortest distance from this node to each other node (Shortest Distance Graph)
- **Runtime depends on the implementation**
  - Binary Heap: $O( |V| + |E| \log |V| )$
  - Array and LinkedList: $O(|V|^2)$
- **NOTE:** Dijkstra will not work with negative edge weights
function prim(G):
    // Input: weighted, undirected graph G with vertices V
    // Output: list of edges in MST
    for all v in V:
        v.cost = ∞
        v.prev = null
    source = a random v in V
    source.cost = 0
    MST = []
    PQ = PriorityQueue(V) // priorities will be v.cost values
    while PQ is not empty:
        v = PQ.removeMin()
        if v.prev != null:
            MST.append((v, v.prev))
        for all incident edges \((v,u)\) of v:
            if u.cost > (v,u).weight:
                u.cost = (v,u).weight
                u.prev = v
                PQ.replaceKey(u, u.cost)
    return MST
Prim-Jarnik

- **General Idea:** Finds the Minimum Spanning Tree given a priority queue
- **Runtime using a Heap implementation:** $O( (|V| + |E|) \log|V| )$
- **NOTE:** Dijkstra will not work with negative edge weights