COMP9312 Advanced Graph Traversal

Outline

- Minimum Spanning Tree
- Prim’s algorithm
- Kruskal’s algorithm

Minimum Spanning Tree

Definition

- Given a connected graph with n vertices, a spanning tree is defined a collection of n – 1 edges which connect all n vertices.
- The spanning tree which minimizes the weight is a minimum spanning tree.

Prim's Algorithm

Main idea

- Starting with an arbitrary vertex to form a minimum spanning tree on one vertex.
- At each step, add that vertex v not yet in minimum spanning tree that has an edge with least weight that connects v to the existing minimum spanning sub-tree
- Continue until we have n-1 edges and n vertices.

Prim's Algorithm

Implementation

- Initialization:
  - Select a root node and set its distance as 0
  - Set the distance to all other vertices as ∞
  - Set all vertices to being unvisited
  - Set the parent pointer of all vertices to 0
    - Iterate while there exists an unvisited vertex with distance < ∞
      - Select that unvisited vertex with minimum distance
      - Mark that vertex as having been visited
      - For each adjacent vertex, if the weight of the connecting edge is less than the current distance to that vertex:
        - Update the distance to equal the weight of the edge
        - Set the current vertex as the parent of the adjacent vertex

Prim's Algorithm

Complexity

- Initialization: $O(|V|)$ memory and run time
- Iterate |V|-1 times, each time finding the closest vertex
  - Iterating through table takes $O(|V|)$ time
  - Each time we find a vertex, we must check its neighbors
  - With an adjacency matrix, the run time is $O(|V|\ (|V|\ +\ |V|))\ =\ O(|V|^2)$
  - With an adjacency list, the run time is $O(|V|^2 + |E|) = O(|V|^2)$ as $E = O(|V|^2)$

Prim's Algorithm

Min-heap optimization

- Iterate |V|-1 times, each time finding the closest vertex
  - Obtain the shortest distance from the min heap takes $O(1)$, maintain the heap takes $O(\log(|V|))$

Thus, the total runtime is $O(|V|\log(|V|) + |E|\log(|V|)) = O(|E|\log(|V|))$

Kruskal's Algorithm

Main idea

- Sorts the edges by weight and goes through the edges from least weights to greatest weight adding the edges to an empty graph as long as the addition does not create a cycle.

Kruskal's Algorithm

Implementation and analysis

- Sort the edges and their weights in an array (using quicksort or distribution sort)
- To determine if a cycle is created, we can perform BFS (a run time of $O(|V|)$)
- Consequently, the runtime would be $O(|E|\log(|E|) + |E| \cdot |V|)$, which should be $O(|E| \cdot |V|)$
- The critical operation is determining if two vertices are connected.

Kruskal's Algorithm

Implementation and analysis

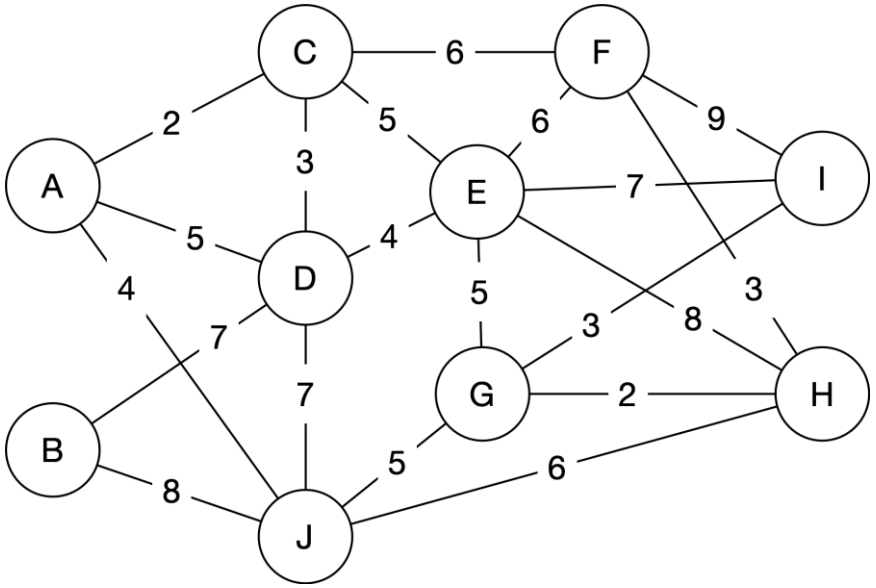
- Using disjoint sets:
  - Consider edges in the same connected sub-graph as forming a set
  - If the vertices of the next edge are in different sets, take the union of the two sets
  - Do not add an edge if both vertices are in the same set.

- Checking if two vertices are in the same set is almost linear
- Taking the union of two disjoint sets is almost linear

Thus, checking and building the minimum spanning tree is now $O(|E|)$
The dominant time is the time for sorting. (e.g. $O(|E|\log(|E|))$ if using quick sort)

Exercise 1: Minimum Spanning Tree

- Implement finding Minimum Spanning Tree
- You can choose to implement one of the algorithms mentioned in lecture



Now, you have 15 minutes to implement ex1.

Q & A