CSCI 115 Lab

Week 15 - Prim's and Kruskal's MST algorithm

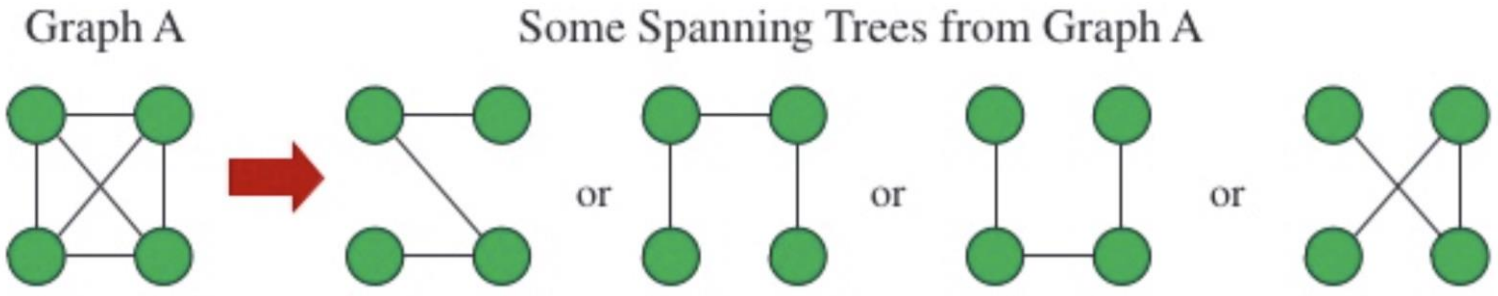
- Professor: Dr. Matin Pirouz
  Email: mpirouz@csufresno.edu

- TA: Shreeja Miyyar
  Email:shreejarao12@mail.fresnostate.edu

Table of Contents

Spanning Tree

- A spanning tree of a graph is a subgraph which contains all the vertices.
- A graph may have many spanning trees.



- A minimum spanning tree is a spanning tree with the lowest cost.
- MST can be obtained using:
  - Prim's algorithm
  - Kruskal's algorithm

Prim's MST

- It is a greedy algorithm which finds the minimum spanning tree for weighted undirected graphs.

- Approach:
  - Create a list that keeps track of vertices already included in MST.
  - Create a key list which keeps track of the weights of the vertex.
  - Create a parent list which keeps track of the parent of a vertex.
  - Initialize all key values as INFINITE in the key list. Assign key value as 0 for the starting vertex.
  - While all vertices have not been traversed
    - Pick a vertex u which is not there in the tracking list and has minimum key value.
    - Include u to the tracking list.
    - Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v
    - Update the parent list of u with the vertex which has minimum cost
  - Create a adjacency matrix from the parent list and output it.

Prim's Algorithm

```
primsMST(G, start_node) {
  // Let the parent list be denoted as parent_list
  // Let the key list be denoted as key_list and set all values to Infinity except start_node. Set start_node to 0.
  // Let the vertex list be denoted as vertex_list and set all values to False. This indicates that none of the vertices have been traversed.
  // Let the output adjacency matrix be denoted as output_G. Initially all values of the matrix is 0.
  For each vertex {
    u = vertex with minimum key value from the key_list
    make the vertex_list[u] = true
    for each adjacent vertices (v) of u {
        if (G[u][v]!=0 && vertex_list[v] == false && G[u][v] < key_list[v]) {
            parent_list[v] = u;
            key_list[v] = G[u][v];
        }
    }
  }
  for each vertex v of the vertex count {
      output_G[v][parent_list[v]] = G[v][parent_list[v]]
      output_G[parent_list[v]][v] = G[parent_list[v]][v]
  }
  Print(output_G)
}
main () {
  // Let the input adjacency matrix be denoted as G
 primsMST(G, start_node)
}
```

Kruskal's MST algorithm

- It is a greedy algorithm which finds the minimum spanning tree for weighted undirected graphs.

- Approach:
  - Create a set with each vertex as its only member.
  - Sort all the edges in non-decreasing order of their weight.
  - For each edge in the sorted list
    - If the vertices of the edge do not belong in the same set
      - Add both the vertices in the same set.

Kruskal's Algorithm

```
kruskalMST(G, start_node) {
  // Let the parent list be denoted as parent_list
  // Let the output adjacency matrix be denoted as output_G. Initially all values of the
  // matrix is 0.
  For each vertex u in vertexes {
      parent_list[u] = u
  }
  while iterating till number of vertices {
      // Iterate the adjacency matrix and find the edge (u, v) with the minimum cost
      // Use the union-find algorithm to check if the edge forms a cycle.
      // If there is no cycle then do union(u, v)
  }
  for each vertex v of the vertex count {
      output_G[v][parent_list[v]] = G[v][parent_list[v]]
      output_G[parent_list[v]][v] = G[parent_list[v]][v]
  }
  Print(output_G)
}
main () {
  // Let the input adjacency matrix be denoted as G
 kruskalMST(G, start_node)
}
```

```
Union(u, v) {
   v1 = find(u)
   v2 = find(v)
   if v1 is not equal to v2{
         parent_list[v1] = v2
    }
}
Find(u) {
   while (parent_list[u] is not equal u)
         u = parent_list[u];
    return u
}
```

Lab Assignment

Hints and Coding Guidelines:

Prim's MST:

- Create a main function which accepts adjacency matrix and start node as inputs.
- Create a function primsMST which accepts these two inputs as arguments. Create the necessary temporary variables as shown in the algorithm.
- To output the adjacency matrix (2D array), here is code snippet:

```
for (int i = 0; i < V; ++i) {
    for (int j = 0; j < V; ++j) {
        std::cout << output[i][j] << ' ';
    }
    std::cout << std::endl;
}
```

Kruskal's MST:

- Create a main function which accepts adjacency matrix and start node as inputs.
- Create a function kruskalMST which accepts these two inputs as arguments. Create the necessary temporary variables as shown in the algorithm.
- Create functions find() and union() which performs the union-find operation as shown in the algorithm.
- To output the adjacency matrix (2D array), refer the code snippet above.

Questions?