Kruskal's Algorithm for Minimum Spanning Tree Generation

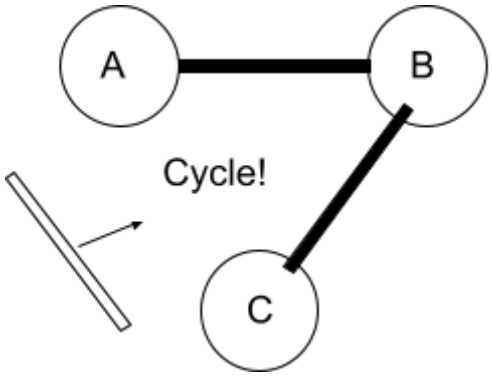
Loren Holl and Kanae Iwata

Graph Theory, Spring 2024

In this report, we would like to present Kruskal's algorithm as an alternative for minimum spanning generation. It is a fairly simple algorithm best used for simpler graphs that are not too dense. We will start by giving an intuitive overview of the algorithm, followed by the algorithm itself along with an example.

Kruskal's algorithm is based on the simple fact that a minimum spanning tree (MST) is mostly made of the edges with lowest weights. It starts by taking all the weights of the graph and sorting them in ascending order. Each time one wants to find the next edge to insert into the minimum spanning tree, one simply looks at the sorted list of edges and picks the next lowest value. However, the trick is to only add edges that will not make a cycle, infringing on the tree property.

In order to efficiently check for cycles, we will use disjoint sets. Disjoint sets allow for very quick find operations (see report on path detection with disjoint sets), which we will use to check if a cycle exists. They also contain the union operation, which unions two of the sets together by adding the smaller set to the larger one. We start by setting each node of the graph to its own set, marking it as the representative of the set. Each time an edge is added that connects vertex v to vertex u, the smaller of the sets will be added to the larger one, and the representative of the smaller one will be set to point to the representative of the larger one. This allows us to quickly determine if two elements are in the same set by checking if they have the same representative. This means that if we have vertex v and u in the same set, then they are connected in the MST. For example, if we had vertices a, b, and c all in the same set (say through connections a-b, b-c), then attempting to add an edge between any of the vertices will create a cycle (see figure 1.1 below).



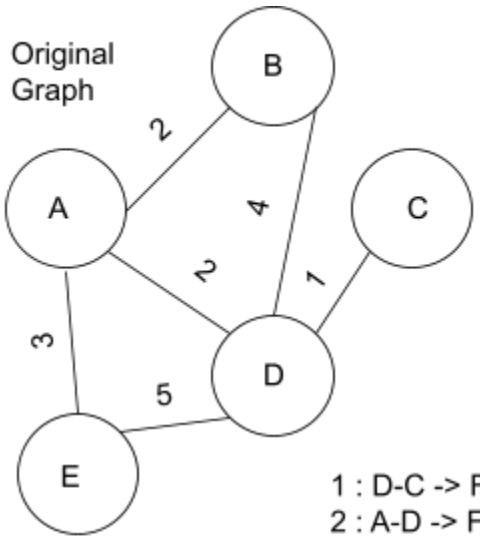
Once we have successfully added all the edges in ascending order to the graph that do not create cycles, then our MST will be complete.

We now present the algorithm below:

Algorithm Kruskal-MST(G)
Input A graph, G
Output A list of the edges that make the MST, M

1. M = { } // Set M to be an empty list
2. For each vertex v in G do
    a. MakeSet(v) // Make a new set for each vertex
3. Sort the edges of G based based off weight in ascending order
4. For each edge (u, v) in G do // For each newly sorted edge in the graph
    a. If Find(u) != Find(v) then // If new edge will not create a cycle
        i. M = M ∪ (u, v) // Add new edge to MST edge list
        ii. Union(u ,v) // Union the vertices together for cycle detection
5. Return M

An example of how this algorithm runs is given below.



Sort edges by order:

Edges = { {D-C, 1}, M = { }
{A-D, 2},
{A-B, 2},
{A-E, 3},
{D-B, 4},
{D-E, 5} }

1 : D-C -> Find(D) != Find(C) -> M.add((D,C)), Union(D, C)
2 : A-D -> Find(A) != Find(C) -> M.add((A, D)), Union(A, D)
3 : A-B -> Find(A) != Find(D) -> M.add((A, B)), Union(A, B)
4 : A-E -> Find(A) != Find(E) -> M.add((A, E)), Union(A, E)
5 : D-B -> Find(D) = Find(B) -> Move to next
6 : D-E -> Find(D) = Find(E) -> Move to next
7 : No more vertices, end algorithm

M = {(D,C), (A,D), (A,B), (A,E)}