Chapter 4

New approach to find minimum spanning tree for undirected graphs

1. INTRODUCTION:

In this chapter we find out a new approach to finding the minimum spanning tree for simple undirected graphs and undirected multi-graphs. The algorithm involves choosing the minimum edge that connects each disjoint component of the graph, until a single component is formed. This single component is the minimum spanning tree of the given graph. The approach we take is a slight modification to Boruvka's algorithm.

One may use minimum spanning trees or MSTs to set up communication links between cities with minimum cost or minimum length. Similarly, MSTs may be used to set up communication networks, telephone line networks, computer networks or piping in a flow network. Due to the various uses of MSTs in everyday problems, efficient algorithms that can solve graphs with a large number of vertices are required. Traditionally, Prim's and Kruskal's algorithms have been used to create minimum spanning trees, and do so efficiently. Their predecessor is Boruvka's (or Sollin's) algorithm, which is where the MST algorithm in this chapter is derived from and discussed detail.

2. BASIC DEFINITIONS:

2.1 Minimum Spanning Tree:

A minimum spanning tree (MST) of a weighted graph $G$ is a spanning tree of $G$ whose edges sum to minimum weight. In other words, a minimum spanning tree is a tree formed from a subset of the edges in a given undirected graph, with two properties: (1) it spans the graph, i.e., it includes every vertex in the graph, and (2) it is a minimum, i.e., the total weight of all the edges is as low as possible.
Figure 4.2 gives four minimum spanning trees, where each of them is of total weight 14. These trees can be derived by growing the spanning tree in a greedy way.
Lemma 1: Any two vertices in a tree are connected by a unique path.

Proof: Since a tree is connected, any two vertices in a tree are connected by at least one simple path. Let $T$ be a tree, and assume there are two distinct paths $P_1$ and $P_2$ from vertex $u$ to vertex $v$. There exists an edge $e = (x,y)$ of $P_1$ that is not an edge of $P_2$. It can be seen that $(P_1 \cup P_2) - e$ is connected, and it contains a path $P_{xy}$ from vertex $x$ to vertex $y$. But then $P_{xy} + e$ is a cycle. This is a contradiction. Thus, there can be at most one path between two vertices in a tree.\[1\]

Lemma 2: Let $T$ be a spanning tree of a graph $G$, and let $e$ be an edge of $G$ not in $T$. Then $T + e$ contains a unique cycle.

Proof: Let $e = (u,v)$. Since $T$ is acyclic, each cycle of $T + e$ contains $e$. Moreover, $X$ is a cycle of $T + e$ if and only if $X - e$ is a path from $u$ to $v$ in $T$. By Lemma 1, such a path is unique in $T$. Thus $T + e$ contains a unique cycle.

In this chapter, we shall examine three well-known algorithms for solving the minimum spanning tree problem: Borůvka’s algorithm, Prim’s algorithm, and Kruskal’s algorithm. They all exploit the following fact in one way or another.\[1\]
3. BORUVKA’S ALGORITHM:

In 1926 Otokar Boruvka attacked the problem of finding the most efficient electricity network for the now-nonexistent nation of Moravia. [1] Distilled into a mathematical form, this is the problem of finding the subgraph of least cost that is still connected. Since then, the task of finding a minimum spanning tree has become a staple of the algorithms repertoire. A few others proposed better solutions, the methods of Kruskal and Prim’s algorithm being the most intuitive and popular[1]

In this MST algorithm, each disjoint component of the spanning tree is connected by adding the minimum edge between any two components to the tree. The algorithm starts with taking all the vertices of the graph as disjoint components and examines each component to determine the minimum edge incident on that component. Effectively, this algorithm continues until only one final component remains, and in the worst case, each iteration halves the number of disjoint components remaining. This selection process also applies to multi-graphs, as only one minimum edge will be chosen between two components that have multiple edges connecting them. The final single component is the required minimum spanning tree.

Algorithm: Boruvka

Input: A weighted, undirected graph $G = (V, E, w)$.
Output: A minimum spanning tree $T$

\begin{align*}
T & \leftarrow \emptyset \\
\text{While } |T| < n - 1 \text{ do} \\
F & \leftarrow \text{a forest consisting of the smallest edge incident to each vertex in } G \\
G & \leftarrow G \setminus F \\
T & \leftarrow T \cup F
\end{align*}
4 PRIM'S ALGORITHM:

Prim's algorithm involves dividing the vertices of a graph into two sets - visited and unvisited, with the initial visited vertex being arbitrarily chosen as the starting vertex. On each iteration, the minimum edge connecting an unvisited vertex to a visited vertex is added to the tree. The final tree $T$ formed, that spans the vertices of the graph is a minimum spanning tree [1].
**Algorithm: Prim**

Input: A weighted, undirected graph $G = (V, E, w)$.

Output: A minimum spanning tree $T$.

$T \leftarrow \emptyset$

Let $r$ be an arbitrarily chosen vertex from $V$.

$U \leftarrow \{r\}$

While $|U| < n$ do

Find $u \in U$ and $v \in V - U$ such that the edge $(u, v)$ is a smallest edge between $U$ and $U - V$

$T \leftarrow T \cup \{(u, v)\}$

$U \leftarrow U \cup \{v\}$

---

(a) $(a, b)$ is added

(b) $(b, d)$ is added
(g) (b, c) is added

(h) A Minimum Spanning Tree
5. KRUSKAL’S ALGORITHM:

In Kruskal’s algorithm, each edge of the graph \( G \) is examined in ascending order of weight, and if the chosen edge does not form a cycle in the tree \( T \), it is added to \( T \). The process continues until \( n-1 \) edges have been added.

**Algorithm:** Kruskal

Input: A weighted, undirected graph \( G = (V, E, w) \).

Output: A minimum spanning tree \( T \).

Sort the edges in \( E \) in nondecreasing order by weight.

\( T \leftarrow \emptyset \)

Create one set for each vertex.

for each edge \((u, v)\) in sorted order do

\( x = \text{Find}(u) \).

\( x = \text{Find}(v) \).

if \( x \neq y \) then

\( T \leftarrow T \cup \{(u, v)\} \)

Union \((x, y)\)

Figure 5 illustrates the execution of the Kruskal algorithm on the graph from Figure 1. Initially, every vertex is a tree in the forest. Let the sorted order of the edges be \( \{(d, e), (g, h), (e, f), (d, f), (b, d), (e, g), (f, h), (b, c), (a, b), (c, h), (a, g)\} \). Since \((d, e)\) joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees (see Figure 5(a)). Next we consider \((g, h)\). Vertex \( g \) and vertex \( h \) belong to two different trees, thus \((g, h)\) is added to the forest as shown in Figure 5(b). In Figure 5(d), when \((d, f)\) is processed, both \( d \) and \( f \) belong to the same tree, therefore we do nothing for this edge. Sorting the edges in nondecreasing order takes \( O(m \log m) \) time.
(a) (d, e) is added
(b) (g, h) is added
(c) (e, f) is added
(d) (b, d) is added
6. FINDING A SPANNING TREE FOR UNDIRECTED GRAPH:

We consider all the vertices of the graph as disjoint components. First to each vertex we identify the edge with minimum weight on it. Ties are broken arbitrarily. These selected edges become the required edges of our minimum spanning tree. In the next stage we are left with several connected components. We then select the edge with the minimum weight between two components. The process is continued till we get a connected component. This selection process also applies to multi-graphs, as only one minimum edge will be chosen between two components. We claim the tree $S$ thus obtained is a minimal spanning tree.

6.1 Proof

$S$ is a spanning graph because the algorithm ensures that all the vertices of $G$ are present. Since the algorithm stops when there is one connected component, the graph $G$ is also connected. Further since we either do not add edges whose vertices are already in $S$, or only add edges from the vertices of one component to the vertices
of the other component, cycles are not formed. Hence S is a spanning tree. We now show that the spanning tree formed by this algorithm and the minimum spanning tree (MST) $T$ obtained by any other standard algorithm like Prim’s or Kruskal’s have the same weight.

Let $e_1, e_2, e_n$ be the edges of $S$ as added by the algorithm. Suppose the edge $e_k = (u, v)$ differs from that in $T$. Let $p$ be a path between $u$ and $v$ in $T$ and let $e$ be the edge by which $u$ is connected to $T$. Consider $T \cup e_k$. This will create cycle in $T$ since the algorithm has picked up $e_k$, it means that $e_k$, is the edge with the least weight incident on either $u$ or $v$. Without loss of generality let us assume $e_k$ is the edge of least weight incident on $u$. Then $wt(e) > wt(e_k)$. Therefore, delete from the tree and replace it by $e_k$. Continuing this process we can replace edge that is present in $T$ and not present in $S$ by an edge present in $S$ which is of less than equal to weight. Hence $wt(S) \leq wt(T)$. Since $T$ is minimum spanning tree. Then $wt(S) = wt(T)$, as $T$ is minimum spanning tree weight of $S$ cannot be less than less than the weight of $T$. Hence $S$ is minimum spanning tree.

6.2 Algorithm & Pseudo Code

The algorithm is as follows:
1 Start
2 For a component $u$ in the graph add the minimum edge $e$ incident on it to it's minimum spanning tree, $S$.
3 If the number of components in the graph is greater than 1, repeat step 1. Otherwise, if there is only one component, it is required minimum spanning tree, $S$, of the graph $G$.
4 End

Pseudo Code main()
1 Start
2 Input the number of vertices
3 Initialize variables
   int i,j=0,min,u=0,v=0, components=n;
4 Find the edges incident on each vertex that are of minimum weight
   for(i=0;i<n;i++)
      findMin(G,i);
5 Build an array of all the vertices connected to vertex 1.
   buildVisited(0);
If the number of components is greater than 1, it means that the minimum spanning tree has not been formed, and the least edge between each component must be added to the tree. If the number of components is 1, go to step

```java
while (components>1)
{
    min = 99; for(i=0;i<n;i++)
    {
        if (visited[i] == 1)
        {

            for(j=0;j<n;j++)
            {
                if (visited[j]==0 && min>G[i][j])
                {
                    min = G[i][j];
                    u = i;
                    v = j;
                }
            }

            connect[v][u] = 1;
            buildVisited(u);
        }
    
    System.out.println(weight);
}
```

FindMin()

1 Start
2 Initialize the variables
3 int j,min=99,pos=0;
4 Find the minimum edge incident on each vertex and add those edges to the tree
   for(j=0;j<n;j++)
\{
    if(min>X[i][j])
    {
        min = X[i][j]; pos = j;
    }
}
connect[i][pos] = 1;
4 End

BuildVisited(u)
1 Start
2 Declare local variables int i;
3 Set the source vertex u as visited visited[u] = 1
4 For each unvisited vertex connected to the source vertex, connect the least edge, and recursively find the least edge connected to this vertex as the source
for(i=0; i<n; i++)
{ 
    if (connect[i][u] == 1 && visited[i] == 0)
    {
        components--;
        System.out.println(i+" - "+u);
        weight -= G[i][u];
        buildVisited(i);
    }
}
5 End

6.3 Illustrate

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{graph.png}
\end{figure}
6.4 A C program for Boruvk's Minimum Spanning Tree (MST) algorithm.

```c
#include <stdio.h>
#include <limits.h>

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with minimum key value, from
// the set of vertices not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed MST stored in parent[]
int printMST(int parent[], int n, int graph[V][V])
{
    printf("Edge Weight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d  %d \n", parent[i], i, graph[i][parent[i]]);
}

// Function to construct and print MST for a graph represented using adjacency
// matrix representation
void BoruvkaMST(int graph[V][V])
```

![Graph](image.png)
Department of Computer Science, Karnatak University, Dharwad

{  
    int parent[V]; // Array to store constructed MST
    int key[V]; // Key values used to pick minimum weight edge in cut
    bool mstSet[V]; // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V-1; count++)
    {
        // Pick the minimum key vertex from the set of vertices
        // not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of the adjacent vertices of
        // the picked vertex. Consider only those vertices which are not yet
        // included in MST
        for (int v = 0; v < V; v++)
        {
            // graph[u][v] is non zero only for adjacent vertices of m
            // mstSet[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
        }
    }

    // print the constructed MST
    printMST(parent, V, graph);
}

// driver program to test above function
int main()
{

int graph[V][V] = {{0, 2, 0, 6, 0},
{2, 0, 3, 8,5},
{0, 3, 0, 0, 7},
{6, 8, 0, 0, 9},
{0, 5, 7, 9, 0},
};

// Print the solution
BoruvkaMST(graph);

return 0;
}

7. CONCLUSION:

The Minimum spanning tree algorithm has been designed and tested to have a complexity of $O(n^2)$, which is comparable to Prim’s algorithm in its adjacency matrix implementation. The applications of MSTs vary over numerous fields; from circuit design to minimize the number of wires used to connect pins.
REFERENCES:

[2]. Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest “Introduction to Algorithms”, PHI,
[5]. Otokar Boruvka. Wikipedia.