Practical 3: Implementing Krushkal's Algorithm for minimum spanning tree with disjoint set data structure

Gahan Saraiya, 18MCEC10

18mcec10@nirmauni.ac.in

I. Introduction

Aim of this practical is to implement Krushkal's algorithm in *C Language* and using disjoint set data structure to detect cycle.

II. Kruskal's Algorithm

- Greedy Algorithm
- Finds out Minimum Spanning Tree (MST) for a connected weighted graph
- finds subset of vertex having total weight of all edges in tree is minimized.
- Directly based on MST property.

III. Program Logic

1. Consider input graph in adjacency matrix
2. Create list of edges for given graph, with their weights.
3. Initialize array representing Disjoint set Data Structure
4. Sort edge list in ascending order.
5. Pick Edge with minimum weight (Top item from edge list)
6. Remove Edge from edge list
7. Check for cycle formation if edge selected, if forms cycle then discard it otherwise add it to the list of edges for minimum spanning tree.
8. Repeat 5 to 7 until list of edges over or MST (minimum spanning tree) contains $total\_edges - 1$ edges.

IV. Implementation

```
//
// ------------------------------------------------------------------------------------------
// Author: Gahan Saraiya
// GiT: http://github.com/gahan9/
// StackOverflow: https://stackoverflow.com/users/story/7664524
// Website: http://gahan9.github.io/
//
// ------------------------------------------------------------------------------------------
// Implementing Krushkal's Algorithm for minimum spanning tree with disjoint set data structure

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX 10

typedef struct Edge {
    int node, another_node, weight;
} Edge;

typedef struct EdgeList {
    Edge data[MAX];
    int total_edges;
} EdgeList;

int Graph[MAX][MAX];


int find(int *array, int node1) {
    return *(array + node1);
}

//change all entries from arr[ A ] to arr[ B ].
void _union(int array[], int number_of_nodes, int node1, int node2) {
    for (int i = 0; i < number_of_nodes; i++) {
        if (array[i] == node2)
            array[i] = node1;
    }
}

void sort(EdgeList edge_list) {
    Edge temp;
    for (int i = 1; i < edge_list.total_edges; i++) {
        for (int j = 0; j < edge_list.total_edges - 1; j++)
```

```
            if (edge_list.data[j].weight > edge_list.data[j + 1].weight) {
                temp = edge_list.data[j];
                edge_list.data[j] = edge_list.data[j + 1];
                edge_list.data[j + 1] = temp;
            }
    }
}

EdgeList Krushkal(int n) {
    /*
     * n: total number of vertices
     * returns edge list of minimum spanning tree
     *
     * */
    int disjoint_set[MAX], i, j;
    EdgeList edge_list;
    edge_list.total_edges = 0;

    // store adjacency matrix in to EdgeList structure
    for (i = 1; i < n; i++) {
        for (j = 0; j < i; j++) {
            if (i != j){  // skip self loop (if accidental entry)
                if (Graph[i][j] != 0) {  // consider 0 weight as same
                 ↪  node/vertices
                    edge_list.data[edge_list.total_edges].node = i;
                    edge_list.data[edge_list.total_edges].another_node = j;
                    edge_list.data[edge_list.total_edges].weight = Graph[i][j];
                    edge_list.total_edges++;  // increase count of total edges
                }
            }
        }
    }

    // sort edges by weight to pick minimum edge
    sort(edge_list);

    for (i = 0; i < n; i++)
        // initialize disjoint set to point at self
        disjoint_set[i] = i;

    EdgeList span_list;
    register int node1, node2;
    span_list.total_edges = 0;

    for (i = 0; i < edge_list.total_edges; i++) {
        node1 = find(disjoint_set, edge_list.data[i].node);
        node2 = find(disjoint_set, edge_list.data[i].another_node);
        if (node1 != node2) {
```

```
90              span_list.data[span_list.total_edges] = edge_list.data[i];
91              span_list.total_edges++;
92              _union(disjoint_set, n, node1, node2);
93          }
94      }
95      return span_list;
96  }
97
98  void pretty_print(EdgeList span_list) {
99      int cost = 0;
100     printf("\nNode\tNode\tWeight");
101     for (int i = 0; i < span_list.total_edges; i++) {
102         printf("\n%d\t   %d\t   %d", span_list.data[i].node,
            ↪  span_list.data[i].another_node, span_list.data[i].weight);
103         cost = cost + span_list.data[i].weight;
104     }
105     printf("\n\nCost of minimum spanning tree : %d", cost);
106 }
107
108 int main(int argc, char *argv[]) {
109     int number_of_vertices = 0;
110     int auto_mode = atoi(argv[1]);
111     if (auto_mode)
112         number_of_vertices = atoi(argv[1]);
113     int num;
114     int i, j, total_cost;
115     printf("\n########################################################################"
116            "\n### Krushkal's Algorithm for MST(Minimum Spanning Tree).... ###"
117
               ↪  "\n########################################################################\n");
118     printf("\nEnter number of vertices: ");
119     if (!auto_mode)
120         scanf("%d", &number_of_vertices);
121     else
122         printf("%d", number_of_vertices);
123     printf("\nEnter the adjacency matrix:\n");
124     for (i = 0; i < number_of_vertices; i++) {
125         for (j = 0; j < number_of_vertices; j++) {
126             if (!auto_mode)
127                 scanf("%d", &Graph[i][j]);
128             else {
129                 if (i == j) {
130                     printf("\t0");
131                 } else {
132                     num = 1 + (rand() % 10);
133                     printf("\t%d", num);
134                     Graph[i][j] = num;
135                 }
```

```
136             }
137         }
138         if (auto_mode)
139             printf("\n");
140     }
141     printf("\n");
142 
143     EdgeList mst_edges;
144     mst_edges = Krushkal(number_of_vertices);
145     pretty_print(mst_edges);  // print edges picked for MST
146     return 1;
147 }
```

Output

```
######################################################################
### Krushkal's Algorithm for MST(Minimum Spanning Tree).... ###
######################################################################

Enter number of vertices: 6
Enter the adjacency matrix:
        0       2       8       5       1       10
        5       0       9       9       3       5
        6       6       0       2       8       2
        2       6       3       0       8       7
        2       5       3       4       0       3
        3       2       7       9       6       0


Node        Node        Weight
1           0           5
2           0           6
3           0           2
5           1           2
5           4           6

Cost of minimum spanning tree : 21
```