Madan Bhandari Memorial College

Lab No: 15 **Date:** 2081/12/29

Write a program in C implementing Kruskal's Algorithm to create MST by greedy approach.

Theory:

Kruskal's Algorithm is a greedy algorithm used to find the Minimum Spanning Tree (MST) of a connected, undirected, and weighted graph. An MST is a subset of the edges that connects all the vertices with the minimum total edge weight and no cycles.

Algorithm:

1. Start
2. Sort all edges in non-decreasing order of their weight.
3. Initialize an empty set for MST.
4. Use Disjoint Set Union (DSU) or Union-Find to detect cycles.
5. Iterate through sorted edges:
   If adding the edge doesn't form a cycle (i.e., the vertices are in different sets), include it in MST.
6. Repeat until MST has (V - 1) edges (where V is the number of vertices).

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Edge {
   int src, dest, weight;
};
struct DisjointSet {
   int* parent;
   int* rank;
   int n;
};
struct DisjointSet* createDisjointSet(int n) {
   struct DisjointSet* ds = (struct DisjointSet*)malloc(sizeof(struct DisjointSet));
   ds->n = n;
   ds->parent = (int*)malloc(n * sizeof(int));
   ds->rank = (int*)malloc(n * sizeof(int));
         int i;
   for (i = 0; i < n; i++) {
      ds->parent[i] = i;
      ds->rank[i] = 0;
   }
   return ds;
}
// Find function with path compression
int find(struct DisjointSet* ds, int u) {
   if (ds->parent[u] != u)
      ds->parent[u] = find(ds, ds->parent[u]);
   return ds->parent[u];
}
// Union by rank
void unionSet(struct DisjointSet* ds, int u, int v) {
   int pu = find(ds, u);
   int pv = find(ds, v);
```

```
    if (pu == pv) return;
    if (ds->rank[pu] < ds->rank[pv]) {
        ds->parent[pu] = pv;
    } else if (ds->rank[pu] > ds->rank[pv]) {
        ds->parent[pv] = pu;
    } else {
        ds->parent[pv] = pu;
        ds->rank[pu]++;
    }
}
int compareEdges(const void* a, const void* b) {
    struct Edge* e1 = (struct Edge*)a;
    struct Edge* e2 = (struct Edge*)b;
    return e1->weight - e2->weight;
}
// Kruskal's Algorithm
void kruskalMST(struct Edge edges[], int V, int E) {
    int step = 1,i,j;
    // Step 1: Sort edges by weight
    printf("Step %d: Sorting edges by weight\n", step++);
    qsort(edges, E, sizeof(edges[0]), compareEdges);
    for (i = 0; i < E; i++) {
        printf("Edge %d - %d: %d\n", edges[i].src, edges[i].dest, edges[i].weight);
    }
    // Step 2: Initialize disjoint set and MST
    struct DisjointSet* ds = createDisjointSet(V);
    struct Edge* mst = (struct Edge*)malloc((V - 1) * sizeof(struct Edge));
    int mst_weight = 0, mst_index = 0;
    // Step 3: Process edges
    printf("Step %d: Building MST\n", step++);
    for (i = 0; i < E && mst_index < V - 1; i++) {
        int u = edges[i].src;
        int v = edges[i].dest;
        int w = edges[i].weight;
        printf("Considering edge %d - %d (weight %d): ", u, v, w);
        if (find(ds, u) != find(ds, v)) {
            mst[mst_index++] = edges[i];
            mst_weight += w;
            unionSet(ds, u, v);
            printf("Added to MST\n");

            printf("MST after step %d: ", step++);
            for (j = 0; j < mst_index; j++) {
                printf("%d-%d ", mst[j].src, mst[j].dest);
            }
            printf("(Total weight = %d)\n", mst_weight);
        } else {
            printf("Skipped (forms cycle)\n");
        }
    }
    // Final MST output
    printf("Final MST: ");
    for (i = 0; i < V - 1; i++) {
        printf("%d-%d ", mst[i].src, mst[i].dest);
    }
    printf("\nTotal weight: %d\n", mst_weight);
    free(ds->parent);
    free(ds->rank);
```

```c
    free(ds);
    free(mst);
}
int main() {
    int V, E, i;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);
    struct Edge* edges = (struct Edge*)malloc(E * sizeof(struct Edge));
    printf("Enter %d edges (source destination weight):\n", E);
    for (i = 0; i < E; i++) {
        printf("Edge %d: ", i + 1);
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }
    printf("Graph edges:\n");
    for (i = 0; i < E; i++) {
        printf("%d - %d: %d\n", edges[i].src, edges[i].dest, edges[i].weight);
    }
    kruskalMST(edges, V, E);
    free(edges);
    return 0;
}
```

OUTPUT:

```
C:\Users\ASUS TUF\OneDrive\
*****************************
        Anupam Nepal
*****************************
Enter the number of vertices: 4
Enter the number of edges: 5
Enter 5 edges (source destination weight):
Edge 1: 0 1 10
Edge 2: 0 2 6
Edge 3: 0 3 5
Edge 4: 1 3 15
Edge 5: 2 3 4
Graph edges:
0 - 1: 10
0 - 2: 6
0 - 3: 5
1 - 3: 15
2 - 3: 4
Step 1: Sorting edges by weight
Edge 2 - 3: 4
Edge 0 - 3: 5
Edge 0 - 2: 6
Edge 0 - 1: 10
Edge 1 - 3: 15
Step 2: Building MST
Considering edge 2 - 3 (weight 4): Added to MST
MST after step 3: 2-3 (Total weight = 4)
Considering edge 0 - 3 (weight 5): Added to MST
MST after step 4: 2-3 0-3 (Total weight = 9)
Considering edge 0 - 2 (weight 6): Skipped (forms cycle)
Considering edge 0 - 1 (weight 10): Added to MST
MST after step 5: 2-3 0-3 0-1 (Total weight = 19)
Final MST: 2-3 0-3 0-1
Total weight: 19

--------------------------------
Process exited after 38.92 seconds with return value 0
Press any key to continue . . .
```

Lab No: 16 **Date:** 2081/12/29

Write a program in C implementing Prim's Algorithm.

Theory:

Prim's Algorithm is a greedy algorithm that constructs the Minimum Spanning Tree (MST) of a connected, undirected graph with weighted edges. Starting from an arbitrary vertex, it incrementally adds the edge with the smallest weight that connects a visited vertex to an unvisited one, ensuring no cycles are formed. The result is a tree that spans all vertices with the minimum total edge weight.

Algorithm:

Input: A graph with V vertices represented as an adjacency matrix
Output: The Minimum Spanning Tree edges and total weight

1. Initialize key[V] with infinity (minimum weight to include each vertex), parent[V] with -1 (to store MST edges), and visited[V] with false.
2. Set key[0] = 0 (start from vertex 0).
3. For count = 0 to V−1:
   - o Find the unvisited vertex u with the minimum key[u].
   - o Mark visited[u] = true.
   - o For each vertex v adjacent to u:
     - ▪ If v is unvisited and graph[u][v] < key[v]:
     - ▪ Update key[v] = graph[u][v].
     - ▪ Set parent[v] = u.
4. Return parent array (MST edges) and compute total weight from key.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define MAX 100
// Function to find the vertex with the minimum key value, from the set of
vertices not yet included in MST
int minKey(int key[], int mstSet[], int V) {
    int min = INT_MAX, min_index = -1, v;
    for (v = 0; v < V; v++)
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    return min_index;
}
void printMST(int parent[], int graph[MAX][MAX], int V) {
    int totalWeight = 0, i;
    printf("Final MST:\n");
    for (i = 1; i < V; i++) {
        printf("%d - %d: %d\n", parent[i], i, graph[i][parent[i]]);
        totalWeight += graph[i][parent[i]];
    }
    printf("Total weight: %d\n", totalWeight);
}
```

```
// Function to perform Prim's algorithm
void primMST(int graph[MAX][MAX], int V) {
    int parent[V];     // Stores MST
    int key[V];        // Minimum weight edge to a vertex
    int mstSet[V];     // True if vertex is included in MST
    int step = 1, i, count, v;
    // Initialize all keys to infinity and mstSet[] to false
    for (i = 0; i < V; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST
    printf("Step %d: Start from vertex 0\n", step++);
    for (count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet, V);
        mstSet[u] = 1;
        printf("Step %d: Pick vertex %d and add to MST\n", step++, u);
        for (v = 0; v < V; v++) {
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
                printf("    Update: parent[%d] = %d, key[%d] = %d\n", v,
u, v, graph[u][v]);
            }
        }
    }
    printMST(parent, graph, V);
}
int main() {
    int V, i, j;
    int graph[MAX][MAX];
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the adjacency matrix (use 0 if no edge exists):\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            printf("Edge weight [%d][%d]: ", i, j);
            scanf("%d", &graph[i][j]);
        }
    }
    printf("\nGraph adjacency matrix:\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            printf("%4d", graph[i][j]);
        }
        printf("\n");
    }
    primMST(graph, V);
    return 0;
}
```

OUTPUT:

```
C:\Users\ASUS TUF\OneDrive'

*****************************
        Anupam Nepal
*****************************
Enter the number of vertices: 4
Enter the adjacency matrix (use 0 if no edge exists):
Edge weight [0][0]: 0
Edge weight [0][1]: 2
Edge weight [0][2]: 3
Edge weight [0][3]: 0
Edge weight [1][0]: 2
Edge weight [1][1]: 0
Edge weight [1][2]: 1
Edge weight [1][3]: 3
Edge weight [2][0]: 3
Edge weight [2][1]: 1
Edge weight [2][2]: 0
Edge weight [2][3]: 4
Edge weight [3][0]: 0
Edge weight [3][1]: 3
Edge weight [3][2]: 4
Edge weight [3][3]: 0

Graph adjacency matrix:
   0   2   3   0
   2   0   1   3
   3   1   0   4
   0   3   4   0
Step 1: Start from vertex 0
Step 2: Pick vertex 0 and add to MST
    Update: parent[1] = 0, key[1] = 2
    Update: parent[2] = 0, key[2] = 3
Step 3: Pick vertex 1 and add to MST
    Update: parent[2] = 1, key[2] = 1
    Update: parent[3] = 1, key[3] = 3
Step 4: Pick vertex 2 and add to MST
Final MST:
0 - 1: 2
1 - 2: 1
1 - 3: 3
Total weight: 6
```