DS Lab Assignment 11

Pranav Joshi
CS-B Batch 2
Roll no: 43

Title: WAP to Generate MST using Prims and Kruskal's algorithm on graph where graph is represented using:

a. Adjacency Matrix.

b. Adjacency Lists.

Code:

1. Prims algorithm on graph using Adjacency Matrix

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

#define V 5

int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;
    for (int 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[V][V]) {
    int totalWeight = 0;
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++) {
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
        totalWeight += graph[i][parent[i]];
    }
    printf("Total Weight: %d\n", totalWeight);
}
```

```c
void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    int mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V-1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;
        for (int 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];
    }
    printMST(parent, graph);
}

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}};
    primMST(graph);
    return 0;
}
```

Output:

```
C:\Code\C\Code\Sem4Assignment11.exe
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Total Weight: 16

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

2. Prims algorithm on graph using Adjacency List

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

struct Node {
    int vertex;
    int weight;
    struct Node* next;
};

struct Graph {
    int numVertices;
    struct Node** adjLists;
};

struct Node* createNode(int v, int w) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->weight = w;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));
    for (int i = 0; i < vertices; i++)
        graph->adjLists[i] = NULL;
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest, int weight) {
    struct Node* newNode = createNode(dest, weight);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    newNode = createNode(src, weight);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

void printMST(int parent[], int graph[][3], int edges) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < edges; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][2]);
```

```c
}

void primMST(struct Graph* graph) {
    int parent[graph->numVertices];
    int key[graph->numVertices];
    int mstSet[graph->numVertices];
    for (int i = 0; i < graph->numVertices; i++)
        key[i] = INT_MAX, mstSet[i] = 0;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < graph->numVertices - 1; count++) {
        int u, v, min = INT_MAX;
        for (int i = 0; i < graph->numVertices; i++)
            if (mstSet[i] == 0 && key[i] < min)
                min = key[i], u = i;
        mstSet[u] = 1;
        struct Node* temp = graph->adjLists[u];
        while (temp) {
            v = temp->vertex;
            if (mstSet[v] == 0 && temp->weight < key[v])
                parent[v] = u, key[v] = temp->weight;
            temp = temp->next;
        }
    }
    int graphArr[graph->numVertices - 1][3];
    int j = 0;
    for (int i = 1; i < graph->numVertices; i++) {
        graphArr[j][0] = parent[i];
        graphArr[j][1] = i;
        graphArr[j][2] = key[i];
        j++;
    }
    printMST(parent, graphArr, graph->numVertices);
    int totalWeight = 0;
    for (int i = 1; i < graph->numVertices; i++) {
        totalWeight += key[i];
    }
    printf("Total weight of minimum spanning tree: %d\n", totalWeight);
}


int main() {
    struct Graph* graph = createGraph(5);
    addEdge(graph, 0, 1, 2);
    addEdge(graph, 0, 3, 6);
    addEdge(graph, 1, 2, 3);
    addEdge(graph, 1, 3, 4);
    addEdge(graph, 1, 4, 5);
    addEdge(graph, 2, 4, 7);
```

```
    primMST(graph);
    return 0;
}
```

Output:

```
C:\Code\C\Code\Sem4Assignment11.exe
Edge    Weight
0 - 1   3
1 - 2   4
1 - 3   5
1 - 4   1
Total weight of minimum spanning tree: 14

Process returned 0 (0x0)   execution time : 0.027 s
Press any key to continue.
```

3. Kruskals algorithm on graph using Adjacency Matrix

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

#define V 5

int parent[V];

int find(int i) {
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void unionv(int i, int j) {
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}
```

```c
void kruskalMST(int graph[V][V]) {
    int mincost = 0;
    for (int i = 0; i < V; i++)
        parent[i] = i;
    int edge_count = 0;
    while (edge_count < V - 1) {
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (find(i) != find(j) && graph[i][j] < min) {
                    min = graph[i][j];
                    a = i;
                    b = j;
                }
            }
        }
        unionv(a, b);
        printf("Edge %d: (%d, %d) cost: %d\n", edge_count++, a, b, min);
        mincost += min;
    }
    printf("Minimum cost = %d\n", mincost);
}

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}};
    kruskalMST(graph);
    return 0;
}
```

Output:

```
C:\Code\C\Code\Sem4Assignment11.exe
Edge    Weight
0 - 1   3
1 - 2   4
1 - 3   5
1 - 4   1
Total weight of minimum spanning tree: 14

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

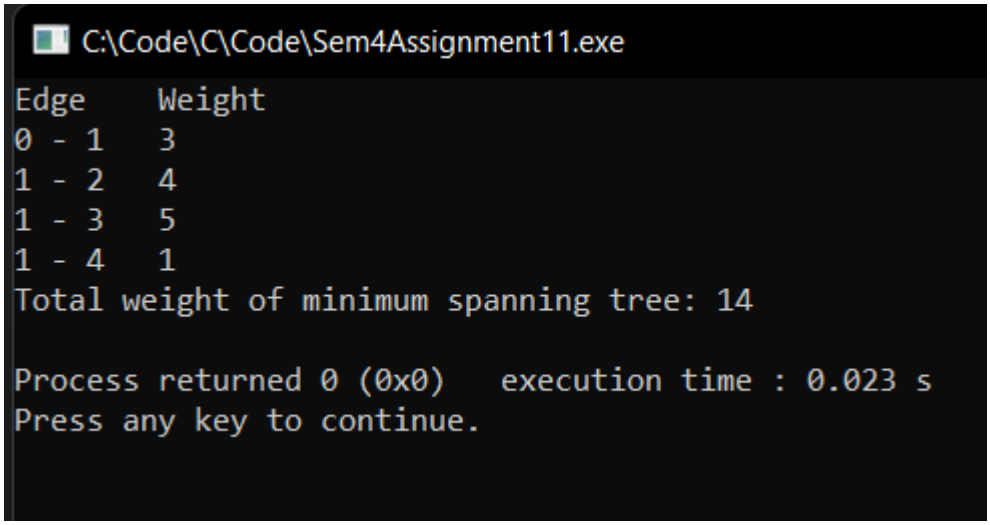
4. Kruskals algorithm on graph using Adjacency List

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

struct Edge {
    int src, dest, weight;
};

struct Subset {
    int parent;
    int rank;
};

struct Graph {
    int V;
    struct Node** adjList;
};

struct Node {
    int dest, weight;
    struct Node* next;
};

struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;
    graph->adjList = (struct Node**) malloc(V * sizeof(struct Node*));
    for (int i = 0; i < V; i++) {
        graph->adjList[i] = NULL;
    }
    return graph;
}

void addEdge(struct Graph* graph, int src, int dest, int weight) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;
    newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->dest = src;
    newNode->weight = weight;
    newNode->next = graph->adjList[dest];
    graph->adjList[dest] = newNode;
}
```

```c
int find(struct Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}

void unionv(struct Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

int compare(const void* a, const void* b) {
    struct Edge* aEdge = (struct Edge*) a;
    struct Edge* bEdge = (struct Edge*) b;
    return aEdge->weight - bEdge->weight;
}

void kruskalMST(struct Graph* graph) {
    int V = graph->V;
    struct Edge result[V - 1];
    int i = 0;

    struct Edge edges[V * V];
    int edgeCount = 0;
    for (int u = 0; u < V; u++) {
        struct Node* temp = graph->adjList[u];
        while (temp != NULL) {
            if (u < temp->dest) {
                edges[edgeCount].src = u;
                edges[edgeCount].dest = temp->dest;
                edges[edgeCount].weight = temp->weight;
                edgeCount++;
            }
            temp = temp->next;
        }
    }
    qsort(edges, edgeCount, sizeof(struct Edge), compare);
```

```
    struct Subset subsets[V];
    for (int v = 0; v < V; v++) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    int e = 0;
    int totalWeight = 0;
    while (i < V - 1 && e < edgeCount) {

        struct Edge nextEdge = edges[e++];
        int x = find(subsets, nextEdge.src);
        int y = find(subsets, nextEdge.dest);

        if (x != y) {
            result[i++] = nextEdge;
            unionv(subsets, x, y);
            totalWeight += nextEdge.weight;
        }
    }

    printf("Edges of the minimum spanning tree:\n");
    for (i = 0; i < V - 1; i++) {
        printf("%d - %d with weight %d\n", result[i].src, result[i].dest,
result[i].weight);
    }
    printf("Total weight of the minimum spanning tree: %d\n", totalWeight);
}


int main() {
    int V = 4;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 10);
    addEdge(graph, 0, 2, 6);
    addEdge(graph, 0, 3, 5);
    addEdge(graph, 1, 3, 15);
    addEdge(graph, 2, 3, 4);
    kruskalMST(graph);
    return 0;
}
```

Output:


C:\Code\C\Code\Sem4Assignment11.exe
Edge Weight
0 - 1 3
1 - 2 4
1 - 3 5
1 - 4 1
Total weight of minimum spanning tree: 14
Process returned 0 (0x0) execution time : 0.023 s
Press any key to continue.