Date : 15.04.2025
Lakshana Baskaran
24011103026

LAB - 12
Graph ADT (Adjacency Matrix & List), Greedy Algorithms

QUESTION 1:

A. Write a separate C++ menu-driven program to implement Graph ADT with an adjacency matrix. Maintain proper boundary conditions and follow good coding practices. The Graph ADT has the following operations,

1. Insert
2. Delete
3. Search
4. Display
5. Exit

SOURCE CODE:

```
//Implementation of graphs using adjacency matrix
#include <iostream>
#include <vector>
using namespace std;

class graph {
    private:
        int n;
        vector<vector<int>> adjmat;

    public:
        graph(int nodes) : n(nodes) {
            adjmat.resize(n+1, vector<int>(n+1, 0));
        }
        void insert_edge(int, int, int);
        void print_graph();
};
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
int main() {
    int nodes, choice, u, v, cost;
    cout << "Enter the number of nodes: ";
    cin >> nodes;
    graph G(nodes);
    cout << "\nMENU\nInsert - 1\nExit - 2\n";
    while (true) {
        cout << "\nAdjacency Matrix:\n";
        G.print_graph();
        cout << "\nEnter Choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "node1 node2 cost: ";
                cin >> u >> v >> cost;
                G.insert_edge(u, v, cost);
                break;

            case 2:
                cout << "\nExiting...\n";
                return 0;

            default:
                cout << "\nInvalid choice. Try again.\n";
                break;
        }
    }
}

void graph::insert_edge(int u, int v, int cost) {
    if (u > 0 && u < n+1 && v > 0 && v < n+1) {
        adjmat[u][v] = cost;
        // adjmat[v][u] = cost; //if undirected graph
    }
}

void graph::print_graph() {
    for (int i = 1; i < n+1; ++i) {
        for (int j = 1; j < n+1; ++j) {
            cout << adjmat[i][j] << " ";
        }
```

```
        cout << endl;
    }
}
```

OUTPUT:

```
● lemon@jupiter:~/workspace/college/DSA/Lab-12$ g++ -o out adj_matrix.cpp
● lemon@jupiter:~/workspace/college/DSA/Lab-12$ ./out
  Enter the number of nodes: 3

  MENU
  Insert - 1
  Exit - 2

  Adjacency Matrix:
  0 0 0
  0 0 0
  0 0 0

  Enter Choice: 1
  node1 node2 cost: 1 2 7

  Adjacency Matrix:
  0 7 0
  0 0 0
  0 0 0

  Enter Choice: 1
  node1 node2 cost: 2 3 5

  Adjacency Matrix:
  0 7 0
  0 0 5
  0 0 0

  Enter Choice: 1
  node1 node2 cost: 3 2 9

  Adjacency Matrix:
  0 7 0
  0 0 5
  0 9 0

  Enter Choice: 2

  Exiting...
○ lemon@jupiter:~/workspace/college/DSA/Lab-12$
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

QUESTION 2:

B. Write a separate C++ menu-driven program to implement Graph ADT with an adjacency list. Maintain proper boundary conditions and follow good coding practices. The Graph ADT has the following operations,

1. Insert
2. Delete
3. Search
4. Display
5. Exit

SOURCE CODE:

```
//Implementation of graphs using adjacency list
#include "sll.h"
#include <vector>
#include <iostream>

using namespace std;

class graph {
private:
    int n;
    vector<list> adjlist;

public:
    graph(int nodes) : n(nodes) {
        adjlist.resize(n+1);
    }

    void insert_edge(int, int, int);
    void print_graph();
};

int main() {
    int nodes, choice, u, v, cost;
    cout << "Enter the number of nodes: ";
    cin >> nodes;
    graph G(nodes);
    cout << "\nMENU\nInsert - 1\nExit - 2\n";
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
    while (true) {
        cout << "\nAdjacency List:\n";
        G.print_graph();
        cout << "\nEnter Choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "node1 node2 cost: ";
                cin >> u >> v >> cost;
                G.insert_edge(u, v, cost);
                break;

            case 2:
                cout << "\nExiting...\n";
                return 0;

            default:
                cout << "\nInvalid choice. Try again.\n";
                break;
        }
    }
    return 0;
}

void graph::insert_edge(int u, int v, int cost) {
    if (u > 0 && u < n+1 && v > 0 && v < n+1) {
        adjlist[u].insert_beginning(v, cost);
    }
}

void graph::print_graph() {
    for (int i = 1; i < n+1; ++i) {
        cout << i << ": ";
        adjlist[i].print();
        cout << endl;
    }
}
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

OUTPUT:

```
lemon@jupiter:~/workspace/college/DSA/Lab-12$ g++ -o out adj_list_sll.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-12$ ./out
Enter the number of nodes: 3

MENU
Insert - 1
Exit - 2

Adjacency List:
1: head -> NULL

2: head -> NULL

3: head -> NULL


Enter Choice: 1
node1 node2 cost: 1 2 8

Adjacency List:
1: (2, 8) -> NULL

2: head -> NULL

3: head -> NULL


Enter Choice: 1
node1 node2 cost: 2 3 6

Adjacency List:
1: (2, 8) -> NULL

2: (3, 6) -> NULL

3: head -> NULL


Enter Choice: 1
node1 node2 cost: 3 2 9
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
Enter Choice: 1
node1 node2 cost: 3 2 9

Adjacency List:
1: (2, 8) -> NULL

2: (3, 6) -> NULL

3: (2, 9) -> NULL


Enter Choice: 1
node1 node2 cost: 1 3 4

Adjacency List:
1: (3, 4) -> (2, 8) -> NULL

2: (3, 6) -> NULL

3: (2, 9) -> NULL


Enter Choice: 2

Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-12$
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

QUESTION 3:

C. Write a separate C++ menu-driven program to implement Graph ADT with the implementation for Prim's algorithm, Kruskal's algorithm, and Dijkstra's algorithm. Maintain proper boundary conditions and follow good coding practices. **(K3)**

SOURCE CODE:

```
//implementation of prim's algorithm to find the shortest path
#include <bits/stdc++.h>
using namespace std;

int minKey(vector<int> &key, vector<bool> &mstSet) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < mstSet.size(); v++)
        if (!mstSet[v] && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(vector<int> &parent, vector<vector<int>> &graph) {
    int totalCost = 0;
    cout << "Edge \tWeight\n";
    for (int i = 1; i < graph.size(); i++) {
        cout << parent[i] << " - " << i << " \t" <<
graph[parent[i]][i] << " \n";
        totalCost += graph[parent[i]][i];
    }
    cout << "Weight of MST is " << totalCost << endl;
}

void primMST(vector<vector<int>> &graph) {
    int V = graph.size();
    vector<int> parent(V);
    vector<int> key(V, INT_MAX);
    vector<bool> mstSet(V, false);

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
        mstSet[u] = true;

        for (int v = 0; v < V; v++)
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph);
}

int main() {
    vector<vector<int>> graph = {
        { 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;
}
```

```
//implementation of kruskal's algorithm to find the shortest path
#include <bits/stdc++.h>
using namespace std;

typedef pair<int, int> iPair;

struct Graph {
      int V, E;
      vector< pair<int, iPair> > edges;

      Graph(int V, int E) {
            this->V = V;
            this->E = E;
      }

      void addEdge(int u, int v, int w) {
            edges.push_back({w, {u, v}});
      }

      int kruskalMST();
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
};

struct DisjointSets {
	int *parent, *rnk;
	int n;

	DisjointSets(int n) {
		this->n = n;
		parent = new int[n+1];
		rnk = new int[n+1];

		for (int i = 0; i <= n; i++) {
			rnk[i] = 0;
			parent[i] = i;
		}
	}

	int find(int u) {
		if (u != parent[u])
			parent[u] = find(parent[u]);
		return parent[u];
	}

	void merge(int x, int y) {
		x = find(x), y = find(y);
		if (rnk[x] > rnk[y])
			parent[y] = x;
		else
			parent[x] = y;
		if (rnk[x] == rnk[y])
			rnk[y]++;
	}
};

int Graph::kruskalMST() {
	int mst_wt = 0;
	sort(edges.begin(), edges.end());
	DisjointSets ds(V);

	for (auto it = edges.begin(); it != edges.end(); it++) {
		int u = it->second.first;
		int v = it->second.second;
		int set_u = ds.find(u);
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
            int set_v = ds.find(v);

            if (set_u != set_v) {
                  cout << u << " - " << v << endl;
                  mst_wt += it->first;
                  ds.merge(set_u, set_v);
            }
      }
      return mst_wt;
}

int main() {
      int V = 9, E = 14;
      Graph g(V, E);

      g.addEdge(0, 1, 4);
      g.addEdge(0, 7, 8);
      g.addEdge(1, 2, 8);
      g.addEdge(1, 7, 11);
      g.addEdge(2, 3, 7);
      g.addEdge(2, 8, 2);
      g.addEdge(2, 5, 4);
      g.addEdge(3, 4, 9);
      g.addEdge(3, 5, 14);
      g.addEdge(4, 5, 10);
      g.addEdge(5, 6, 2);
      g.addEdge(6, 7, 1);
      g.addEdge(6, 8, 6);
      g.addEdge(7, 8, 7);

      cout << "Edges of MST are \n";
      int mst_wt = g.kruskalMST();
      cout << "\nWeight of MST is " << mst_wt << endl;

      return 0;
}
```

//implementation of dijkstra's algorithm to find the shortest path

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

#define V 9
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

```
int minDistance(int dist[], bool sptSet[]) {
     int min = INT_MAX, min_index;
     for (int v = 0; v < V; v++)
          if (sptSet[v] == false && dist[v] <= min)
               min = dist[v], min_index = v;
     return min_index;
}

void printSolution(int dist[], int n) {
     printf("Vertex Distance from Source\n");
     for (int i = 0; i < V; i++)
          printf("\t%d \t\t\t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src) {
     int dist[V];
     bool sptSet[V];

     for (int i = 0; i < V; i++)
          dist[i] = INT_MAX, sptSet[i] = false;

     dist[src] = 0;

     for (int count = 0; count < V - 1; count++) {
          int u = minDistance(dist, sptSet);
          sptSet[u] = true;

          for (int v = 0; v < V; v++)
               if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                    && dist[u] + graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
     }

     printSolution(dist, V);
}

int main() {
     int graph[V][V] = {
          { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
          { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
          { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
          { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
          { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
```

```
            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
            { 0, 0, 2, 0, 0, 0, 6, 7, 0 }
        };

        dijkstra(graph, 0);
        return 0;
}
```

OUTPUT:

PRIM'S ALGORITHM:

```
lemon@jupiter:~/workspace/college/DSA/Lab-12$ g++ -o out prims.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-12$ ./out
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Weight of MST is 16
lemon@jupiter:~/workspace/college/DSA/Lab-12$
```

KRUSKAL'S ALGORITHM:

```
lemon@jupiter:~/workspace/college/DSA/Lab-12$ g++ -o out kruskals.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-12$ ./out
Edges of MST are
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4

Weight of MST is 37
lemon@jupiter:~/workspace/college/DSA/Lab-12$
```

Date : 15.04.2025
Lakshana Baskaran
24011103026

DIJKSTRA'S ALGORITHM:

```
lemon@jupiter:~/workspace/college/DSA/Lab-12$ g++ -o out dijktras.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-12$ ./out
Vertex Distance from Source
        0                                0
        1                                4
        2                                12
        3                                19
        4                                21
        5                                11
        6                                9
        7                                8
        8                                14
lemon@jupiter:~/workspace/college/DSA/Lab-12$
```

—