DATA STRUCTURE: LAB 14

F24-0767

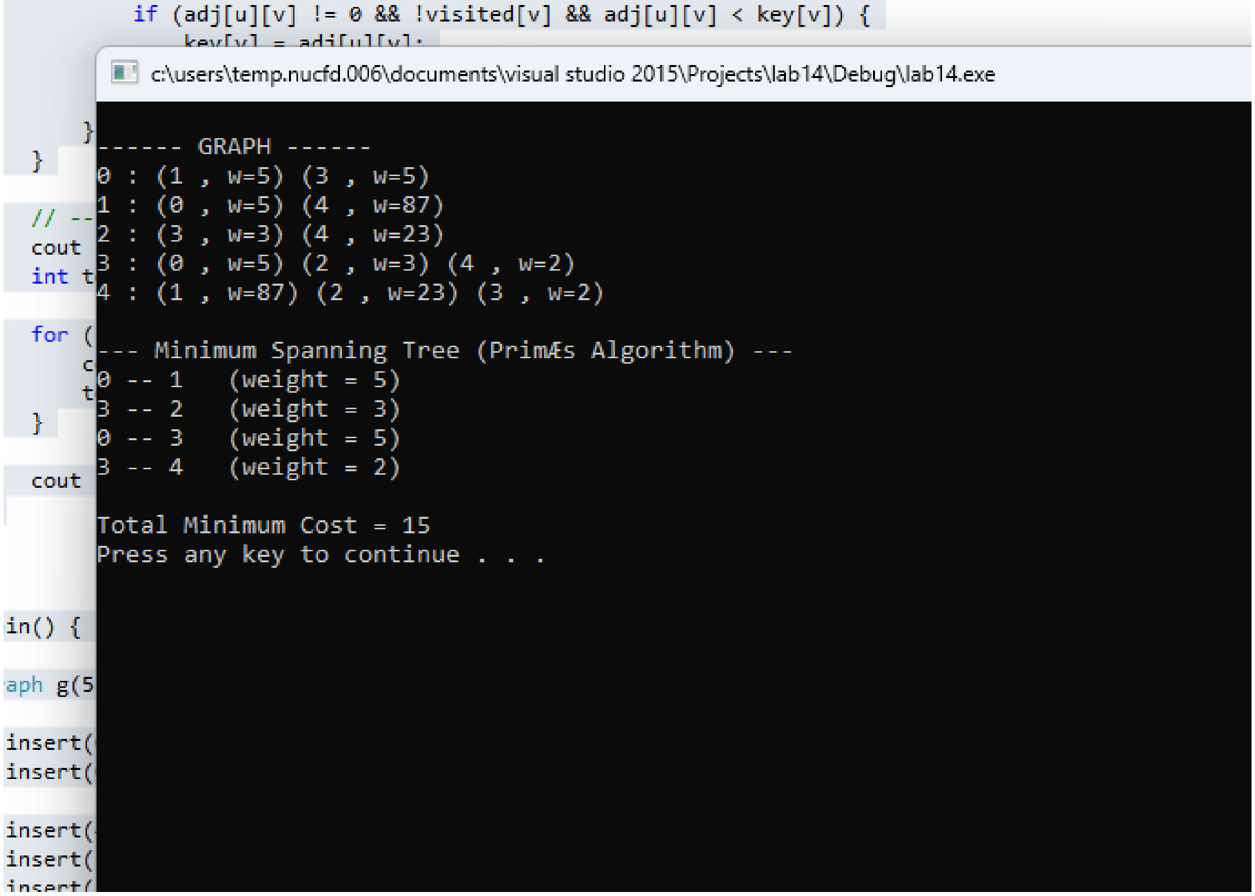
Muhammad Ayaan Amir

TASK#1:

```
#include<iostream>
#include<climits>
using namespace std;

class graph {
private:
	int **adj;
	int size;

public:
	graph(int n) {
		size = n;
		adj = new int*[size];
		for (int i = 0; i < size; i++) {
			adj[i] = new int[size];
			for (int j = 0; j < size; j++)
				adj[i][j] = 0;
		}
	}

	void insert(int u, int v, int weight) {
		adj[u][v] = weight;
		adj[v][u] = weight;
	}

	void display() {
		cout << "\n------ GRAPH ------\n";
		for (int i = 0; i < size; i++) {
			cout << i << " : ";
			for (int j = 0; j < size; j++) {
				if (adj[i][j] != 0)
					cout << "(" << j << " , w=" << adj[i][j] << ") ";
			}
			cout << endl;
		}
	}
```

```
    void prim() {
        int parent[100];
        int key[100];
        bool visited[100];

        for (int i = 0; i < size; i++) {
            parent[i] = -1;
            key[i] = INT_MAX;
            visited[i] = false;
        }

        key[0] = 0; // start with node 0

        for (int c = 0; c < size - 1; c++) {

            // --------- Find Minimum Key Vertex ----------
            int u = -1;
            int minVal = INT_MAX;

            for (int i = 0; i < size; i++) {
                if (!visited[i] && key[i] < minVal) {
                    minVal = key[i];
                    u = i;
                }
            }

            visited[u] = true;

            // --------- Update Adjacent Vertices ----------
            for (int v = 0; v < size; v++) {
                if (adj[u][v] != 0 && !visited[v] && adj[u][v] < key[v]) {
                    key[v] = adj[u][v];
                    parent[v] = u;
                }
            }
        }

        // -------- PRINT MST --------
        cout << "\n--- Minimum Spanning Tree (Prim’s Algorithm) ---\n";
        int totalCost = 0;

        for (int i = 1; i < size; i++) {
            cout << parent[i] << " -- " << i << "   (weight = " <<
adj[i][parent[i]] << ")\n";
            totalCost += adj[i][parent[i]];
        }

        cout << "\nTotal Minimum Cost = " << totalCost << endl;
    }
};


int main() {

    graph g(5);

    g.insert(0, 1, 5);
```

```
        g.insert(0, 3, 5);

        g.insert(4, 3, 4);
        g.insert(3, 2, 3);
        g.insert(2, 4, 23);
        g.insert(1, 4, 87);
        g.insert(3, 4, 2);

        g.display();

        g.prim();
        system("pause");
        return 0;
}
```



```
            if (adj[u][v] != 0 && !visited[v] && adj[u][v] < key[v]) {
```

c:\users\temp.nucfd.006\documents\visual studio 2015\Projects\lab14\Debug\lab14.exe

```
------ GRAPH ------
0 : (1 , w=5) (3 , w=5)
1 : (0 , w=5) (4 , w=87)
2 : (3 , w=3) (4 , w=23)
3 : (0 , w=5) (2 , w=3) (4 , w=2)
4 : (1 , w=87) (2 , w=23) (3 , w=2)

--- Minimum Spanning Tree (PrimÆs Algorithm) ---
0 -- 1    (weight = 5)
3 -- 2    (weight = 3)
0 -- 3    (weight = 5)
3 -- 4    (weight = 2)

Total Minimum Cost = 15
Press any key to continue . . .
```

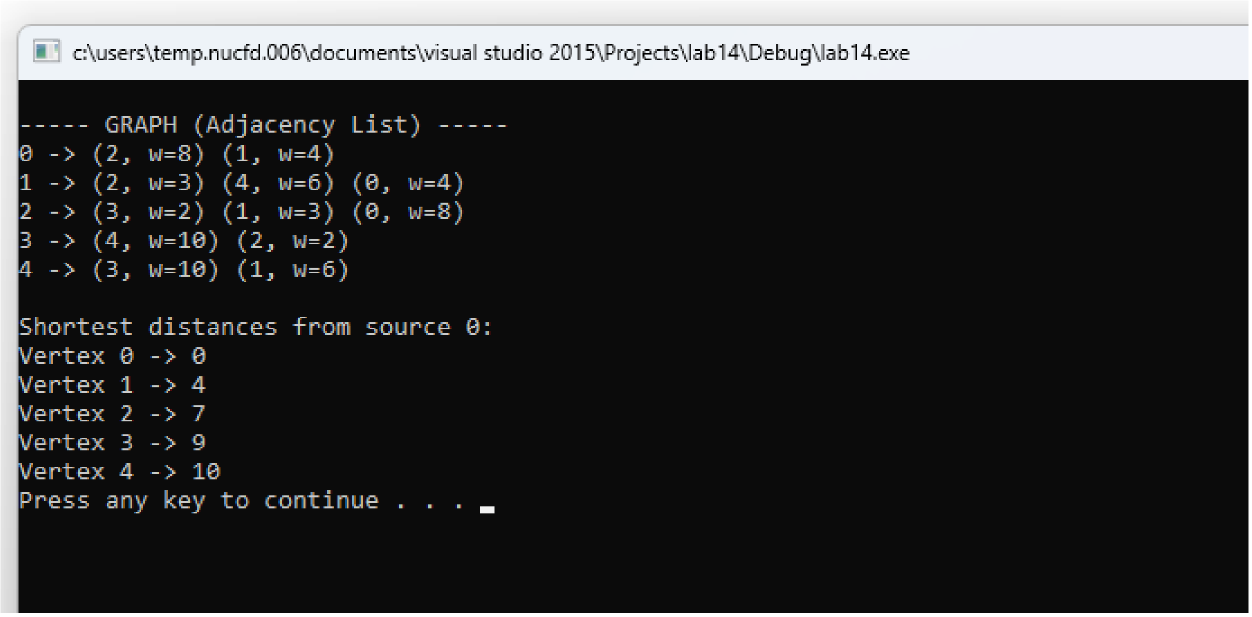
TASK#2:

```
#include <iostream>
#include <climits>
using namespace std;

// ---------------- NODE CLASS ----------------
class Node {
public:
        int v;          // neighbor vertex
        int w;          // weight
        Node* next;
```

```
	Node(int vertex, int weight) {
		v = vertex;
		w = weight;
		next = nullptr;
	}
};

// ---------------- GRAPH CLASS ----------------
class Graph {
private:
	int size;
	Node* adj[100];    // adjacency list

public:
	Graph(int n) {
		size = n;
		for (int i = 0; i < size; i++)
			adj[i] = nullptr;
	}

	// Add edge u -- v with weight w (undirected)
	void addEdge(int u, int v, int w) {
		Node* temp1 = new Node(v, w);
		temp1->next = adj[u];
		adj[u] = temp1;

		Node* temp2 = new Node(u, w);
		temp2->next = adj[v];
		adj[v] = temp2;
	}

	// Display adjacency list
	void display() {
		cout << "\n----- GRAPH (Adjacency List) -----\n";
		for (int i = 0; i < size; i++) {
			cout << i << " -> ";
			Node* temp = adj[i];
			while (temp != nullptr) {
				cout << "(" << temp->v << ", w=" << temp->w << ") ";
				temp = temp->next;
			}
			cout << endl;
		}
	}

	// ---------------- DIJKSTRA'S ALGORITHM ----------------
	void dijkstra(int src) {
		int dist[100];
		bool visited[100];

		for (int i = 0; i < size; i++) {
			dist[i] = INT_MAX;
			visited[i] = false;
		}

		dist[src] = 0;
```

```cpp
        for (int count = 0; count < size - 1; count++) {

            // ---- Pick minimum distance unvisited vertex ----
            int u = -1;
            int minVal = INT_MAX;

            for (int i = 0; i < size; i++) {
                if (!visited[i] && dist[i] < minVal) {
                    minVal = dist[i];
                    u = i;
                }
            }

            visited[u] = true;

            // ---- Relax all neighbors of u ----
            Node* temp = adj[u];
            while (temp != nullptr) {
                int v = temp->v;
                int w = temp->w;

                if (!visited[v] && dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                }

                temp = temp->next;
            }
        }

        // ---- Print result ----
        cout << "\nShortest distances from source " << src << ":\n";
        for (int i = 0; i < size; i++) {
            cout << "Vertex " << i << " -> " << dist[i] << endl;
        }
    }
};

// ---------------- MAIN FUNCTION ----------------
int main() {

    Graph g(5);

    // Given Example
    g.addEdge(0, 1, 4);
    g.addEdge(0, 2, 8);

    g.addEdge(1, 4, 6);
    g.addEdge(1, 2, 3);

    g.addEdge(2, 3, 2);

    g.addEdge(3, 4, 10);

    g.display();

    g.dijkstra(0);
    system("pause");
    return 0;
```

```
}
```



c:\users\temp.nucfd.006\documents\visual studio 2015\Projects\lab14\Debug\lab14.exe

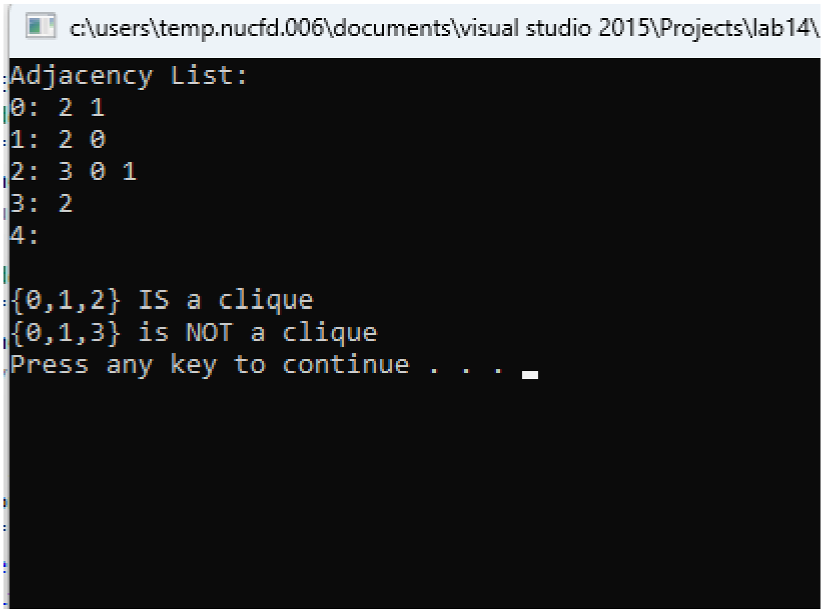
```
----- GRAPH (Adjacency List) -----
0 -> (2, w=8) (1, w=4)
1 -> (2, w=3) (4, w=6) (0, w=4)
2 -> (3, w=2) (1, w=3) (0, w=8)
3 -> (4, w=10) (2, w=2)
4 -> (3, w=10) (1, w=6)

Shortest distances from source 0:
Vertex 0 -> 0
Vertex 1 -> 4
Vertex 2 -> 7
Vertex 3 -> 9
Vertex 4 -> 10
Press any key to continue . . .
```

TASK#3:

```
#include <iostream>
using namespace std;

class Node {
public:
	int v;        // neighbor vertex
	Node* next;   // pointer to next node

	Node(int x) {
		v = x;
		next = nullptr;
	}
};

class Graph {
private:
	int size;
	Node** adj;   // adjacency list array

public:
	Graph(int n) {
		size = n;
		adj = new Node*[size];

		for (int i = 0; i < size; i++)
			adj[i] = nullptr;  // start with empty list
```

```
        }

        void addEdge(int u, int v) {
                // add v in u's list
                Node* n1 = new Node(v);
                n1->next = adj[u];
                adj[u] = n1;

                // add u in v's list
                Node* n2 = new Node(u);
                n2->next = adj[v];
                adj[v] = n2;
        }

        // Check if edge u — v exists
        bool isConnected(int u, int v) {
                Node* temp = adj[u];
                while (temp != nullptr) {
                        if (temp->v == v)
                                return true;
                        temp = temp->next;
                }
                return false;
        }

        // Check if the given list of nodes forms a clique
        bool is_clique(int nodes[], int n) {
                // Check all pairs
                for (int i = 0; i < n; i++) {
                        for (int j = i + 1; j < n; j++) {
                                int a = nodes[i];
                                int b = nodes[j];

                                // If ANY pair is not connected → NOT a clique
                                if (!isConnected(a, b))
                                        return false;
                        }
                }
                return true;
        }

        // Display adjacency list (for testing)
        void display() {
                for (int i = 0; i < size; i++) {
                        cout << i << ": ";
                        Node* temp = adj[i];
                        while (temp != nullptr) {
                                cout << temp->v << " ";
                                temp = temp->next;
                        }
                        cout << endl;
                }
        }
};

int main() {
        Graph g(5);
```

```
    g.addEdge(0, 1);
    g.addEdge(1, 2);
    g.addEdge(0, 2);
    g.addEdge(2, 3);

    cout << "Adjacency List:\n";
    g.display();

    int group1[] = { 0, 1, 2 };
    int group2[] = { 0, 1, 3 };

    if (g.is_clique(group1, 3))
        cout << "\n{0,1,2} IS a clique\n";
    else
        cout << "\n{0,1,2} is NOT a clique\n";

    if (g.is_clique(group2, 3))
        cout << "{0,1,3} IS a clique\n";
    else
        cout << "{0,1,3} is NOT a clique\n";
    system("pause");
    return 0;
}
```


c:\users\temp.nucfd.006\documents\visual studio 2015\Projects\lab14\

```
Adjacency List:
0: 2 1
1: 2 0
2: 3 0 1
3: 2
4:

{0,1,2} IS a clique
{0,1,3} is NOT a clique
Press any key to continue . . .
```