

LAB REPORT

**CSE2011 – DATA STRUCTURES AND ALGORITHMS LAB**



**(B.Tech. CSE Specialisation in Bioinformatics)**

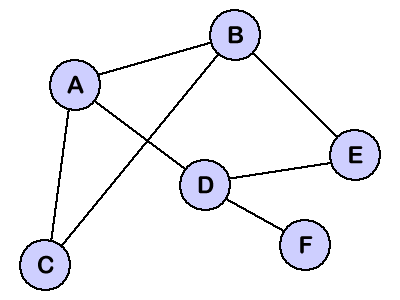
**WINTER SEMESTER 2020-2021**

|  |  |
| --- | --- |
| **Name:** | ALOK MATHUR |
| **Reg. No:** | 20BCB0086 |
| **Slot:** | L51+L52 |
| **Faculty Name:** | SRIVANI A Ma’am |

**VIT – A Place to Learn; A Chance to Grow**

**ASSIGNMENT 6**

1. Construct the given graph using Adjacency Matrix and Perform breadth first search on it.



**CODE**

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <stdbool.h>

*#define* MAX 6

struct Vertex

{

    char label;

    bool visited;

};

int queue[MAX];

int rear = -1;

int front = 0;

int queueItemCount = 0;

struct Vertex \*lstVertices[MAX];

int adjMatrix[MAX][MAX];

int vertexCount = 0;

void insert(int data)

{

    queue[++rear] = data;

    queueItemCount++;

}

int removeData()

{

    queueItemCount--;

*return* queue[front++];

}

bool isQueueEmpty()

{

*return* queueItemCount == 0;

}

void addVertex(char label)

{

    struct Vertex \*vertex = (struct Vertex \*)malloc(sizeof(struct Vertex));

    vertex->label = label;

    vertex->visited = false;

    lstVertices[vertexCount++] = vertex;

}

void addEdge(int start, int end)

{

    adjMatrix[start][end] = 1;

    adjMatrix[end][start] = 1;

}

void displayVertex(int vertexIndex)

{

    printf("%c ", lstVertices[vertexIndex]->label);

}

int getAdjUnvisitedVertex(int vertexIndex)

{

    int i;

*for* (i = 0; i < vertexCount; i++)

    {

*if* (adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false)

*return* i;

    }

*return* -1;

}

void breadthFirstSearch()

{

    int i;

    lstVertices[0]->visited = true;

    displayVertex(0);

    insert(0);

    int unvisitedVertex;

*while* (!isQueueEmpty())

    {

        int tempVertex = removeData();

*while* ((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1)

        {

            lstVertices[unvisitedVertex]->visited = true;

            displayVertex(unvisitedVertex);

            insert(unvisitedVertex);

        }

    }

*for* (i = 0; i < vertexCount; i++)

    {

        lstVertices[i]->visited = false;

    }

}

int main()

{

    int i, j;

*for* (i = 0; i < MAX; i++)

    {

*for* (j = 0; j < MAX; j++)

            adjMatrix[i][j] = 0;

    }

    addVertex('A');

    addVertex('B');

    addVertex('C');

    addVertex('D');

    addVertex('E');

    addVertex('F');

    addEdge(0, 1);

    addEdge(0, 2);

    addEdge(0, 3);

    addEdge(1, 2);

    addEdge(1, 4);

    addEdge(3, 4);

    addEdge(3, 5);

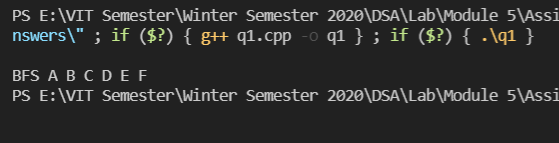
    printf("\nBFS ");

    breadthFirstSearch();

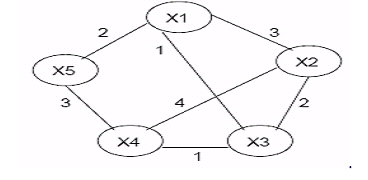
*return* 0;

}

**OUTPUT**



2. Write a program to implement Kruskal’s Algorithm and find the Minimum Spanning Tree for the following graph.



**CODE**

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* <iostream>

using namespace std;

struct Edge

{

    int src, dest, weight;

};

struct Graph

{

    int V, E;

    struct Edge \*edge;

};

struct Graph \*createGraph(int V, int E)

{

    struct Graph \*graph = (struct Graph \*)malloc(sizeof(struct Graph));

    graph->V = V;

    graph->E = E;

    graph->edge = (struct Edge \*)malloc(graph->E \* sizeof(struct Edge));

*return* graph;

}

struct subset

{

    int parent;

    int rank;

};

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 Union(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 myComp(const void \*a, const void \*b)

{

    struct Edge \*a1 = (struct Edge \*)a;

    struct Edge \*b1 = (struct Edge \*)b;

*return* a1->weight > b1->weight;

}

void KruskalMST(struct Graph \*graph)

{

    int V = graph->V;

    struct Edge result[V];

    int e = 0;

    int i = 0;

    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    struct subset \*subsets = (struct subset \*)malloc(V \* sizeof(struct subset));

*for* (int v = 0; v < V; ++v)

    {

        subsets[v].parent = v;

        subsets[v].rank = 0;

    }

*while* (e < V - 1)

    {

        struct Edge next\_edge = graph->edge[i++];

        int x = find(subsets, next\_edge.src);

        int y = find(subsets, next\_edge.dest);

*if* (x != y)

        {

            result[e++] = next\_edge;

            Union(subsets, x, y);

        }

    }

    cout << "These are the edges in the constructed MST\n";

*for* (i = 0; i < e; ++i)

        printf("%d -- %d == %d\n", result[i].src, result[i].dest,

               result[i].weight);

*return*;

}

int main()

{

    int V = 5;

    int E = 7;

    struct Graph \*graph = createGraph(V, E);

*//X1-X2*

    graph->edge[0].src = 0;

    graph->edge[0].dest = 1;

    graph->edge[0].weight = 3;

*//X2-X3*

    graph->edge[1].src = 1;

    graph->edge[1].dest = 2;

    graph->edge[1].weight = 2;

*//X3-X4*

    graph->edge[2].src = 2;

    graph->edge[2].dest = 3;

    graph->edge[2].weight = 1;

*//X4-X5*

    graph->edge[3].src = 3;

    graph->edge[3].dest = 4;

    graph->edge[3].weight = 3;

*//X5-X1*

    graph->edge[4].src = 4;

    graph->edge[4].dest = 0;

    graph->edge[4].weight = 2;

*//X1-X3*

    graph->edge[5].src = 0;

    graph->edge[5].dest = 2;

    graph->edge[5].weight = 1;

*//X2-X4*

    graph->edge[6].src = 1;

    graph->edge[6].dest = 3;

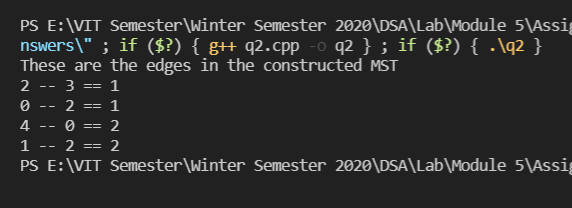
    graph->edge[6].weight = 4;

    KruskalMST(graph);

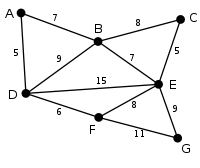
*return* 1;

}

**OUTPUT**



3. Write a program to implement Prim’s Algorithm and find the Minimum Spanning Tree for the following graph.



**CODE**

*#include* <stdio.h>

*#include* <limits.h>

*#include* <iostream>

using namespace std;

*#define* V 7

int minKey(int key[], bool mstSet[])

{

    int min = INT\_MAX, min\_index;

*for* (int v = 0; v < V; v++)

*if* (mstSet[v] == false && key[v] < min)

            min = key[v], min\_index = v;

*return* min\_index;

}

int printMST(int parent[], int n, int graph[V][V])

{

    cout << "Edge Weight\n";

*for* (int i = 1; i < V; i++)

        printf("%d - %d %d \n", parent[i], i, graph[i][parent[i]]);

}

void primMST(int graph[V][V])

{

    int parent[V];

    int key[V];

    bool mstSet[V];

*for* (int i = 0; i < V; i++)

        key[i] = INT\_MAX, mstSet[i] = false;

    key[0] = 0;

    parent[0] = -1;

*for* (int count = 0; count < V - 1; count++)

    {

        int u = minKey(key, mstSet);

        mstSet[u] = true;

*for* (int v = 0; v < V; v++)

*if* (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

                parent[v] = u, key[v] = graph[u][v];

    }

    printMST(parent, V, graph);

}

int main()

{

*//0-A,1-B,2-C,3-D,4-E,5-F,6-G*

    int graph[V][V] = {{0, 7, 0, 5, 0, 0, 0},

                       {7, 0, 8, 9, 7, 0, 0},

                       {0, 8, 0, 0, 5, 0, 0},

                       {5, 9, 0, 0, 15, 6, 0},

                       {0, 7, 5, 15, 0, 8, 9},

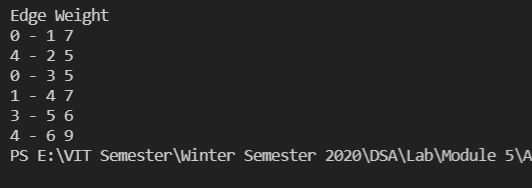
                       {0, 0, 0, 6, 8, 0, 11},

                       {0, 0, 0, 0, 9, 11, 0}};

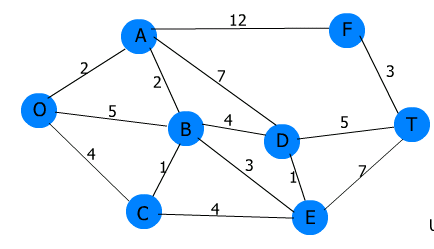
    primMST(graph);

*return* 0;

}



4. Write a program that creates the following graph and finds the shortest path from the vertex O to all the other vertices using Dijikstra’s Algorithm.



**CODE**

*#include* <iostream>

*#include* <stdio.h>

using namespace std;

*#define* INFINITY 9999

*#define* max 8

void dijkstra(int G[max][max], int n, int startnode);

void dijkstra(int G[max][max], int n, int startnode)

{

    int cost[max][max], distance[max], pred[max];

    int visited[max], count, mindistance, nextnode, i, j;

*for* (i = 0; i < n; i++)

*for* (j = 0; j < n; j++)

*if* (G[i][j] == 0)

                cost[i][j] = INFINITY;

*else*

                cost[i][j] = G[i][j];

*for* (i = 0; i < n; i++)

    {

        distance[i] = cost[startnode][i];

        pred[i] = startnode;

        visited[i] = 0;

    }

    distance[startnode] = 0;

    visited[startnode] = 1;

    count = 1;

*while* (count < n - 1)

    {

        mindistance = INFINITY;

*for* (i = 0; i < n; i++)

*if* (distance[i] < mindistance && !visited[i])

            {

                mindistance = distance[i];

                nextnode = i;

            }

        visited[nextnode] = 1;

*for* (i = 0; i < n; i++)

*if* (!visited[i])

*if* (mindistance + cost[nextnode][i] < distance[i])

                {

                    distance[i] = mindistance + cost[nextnode][i];

                    pred[i] = nextnode;

                }

        count++;

    }

*for* (i = 0; i < n; i++)

*if* (i != startnode)

        {

            cout << "\nDistance of node " << i << " from node 0(O) = " << distance[i];

            cout << "\nPath Taken = " << i;

            j = i;

*do*

            {

                j = pred[j];

                cout << " <- " << j;

            } *while* (j != startnode);

            cout << endl;

        }

}

int main()

{

*//O-0,A-1,B-2,C-3,D-4,E-5,F-6,T-7*

    int G[max][max] = {

        {0, 2, 5, 4, 0, 0, 0, 0},

        {2, 0, 2, 0, 7, 0, 12, 0},

        {5, 2, 0, 1, 4, 3, 0, 0},

        {4, 0, 1, 0, 0, 4, 0, 0},

        {0, 7, 4, 0, 0, 1, 0, 5},

        {0, 0, 3, 4, 1, 0, 0, 7},

        {0, 12, 0, 0, 0, 0, 0, 3},

        {0, 0, 0, 0, 5, 7, 3, 0},

    };

    int n = 8;

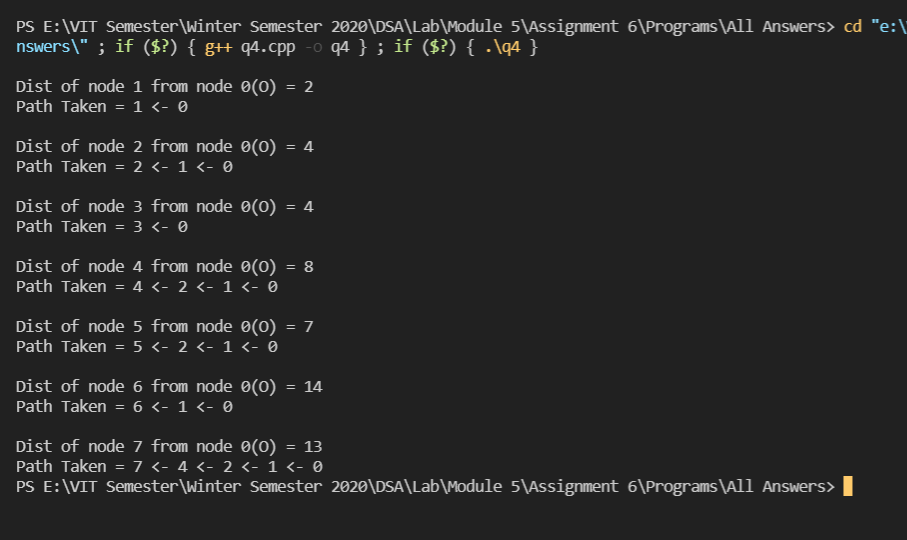
    int u = 0;

    dijkstra(G, n, u);

*return* 0;

}

**OUTPUT**



5. Given an array of elements, construct a min Heap and perform in-place sorting (ascending order) using heap sort.

**CODE**

*#include* <iostream>

using namespace std;

void heapify(int arr[], int n, int i)

{

    int smallest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

*if* (l < n && arr[l] < arr[smallest])

        smallest = l;

*if* (r < n && arr[r] < arr[smallest])

        smallest = r;

*if* (smallest != i)

    {

        swap(arr[i], arr[smallest]);

        heapify(arr, n, smallest);

    }

}

void heapSort(int arr[], int n)

{

*for* (int i = n / 2 - 1; i >= 0; i--)

        heapify(arr, n, i);

*for* (int i = n - 1; i >= 0; i--)

    {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0);

    }

}

void printArray(int arr[], int n)

{

*for* (int i = n - 1; i >= 0; i--)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    int arr[6] = {4, 25, 10, 1, 8, 3};

    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is :" << endl;

    printArray(arr, n);

*return* 1;

}

**OUTPUT**

