Analysis & Design of Algorithms

Semester III 2018-19

Lab - 7

Topics: To find the Minimum Spanning Tree (MST) using Prim's algorithm

---

INTRODUCTION

In this lab we would be finding Minimum Spanning Tree (MST) using Prim's algorithm. Graph is represented using adjacency list. The implementation can be also done using adjacency list. The code for constructing an graph is given below. Priority queue or Minheap can be used for computing the minimum cost.

Graph using adjacency list

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

// A structure to represent a node in adjacency list
struct AdjListNode {
        int dest;
        int weight;
        struct AdjListNode* next;
};

// A structure to represent an adjacency list
struct AdjList {
        struct AdjListNode* head; // pointer to head node of list
};
// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph {
        int V;
        struct AdjList* array;
```

```
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest, int weight)
{
        struct AdjListNode* newNode = (struct AdjListNode*)
        malloc(sizeof(struct AdjListNode));
        newNode->dest = dest;
        newNode->weight = weight;
        newNode->next = NULL;
        return newNode;
}
// A function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
        struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
        graph->V = V;
        // Create an array of adjacency lists.  Size of array will be V
        graph->array = (struct AdjList*)malloc(V * sizeof(struct AdjList));
        // Initialize each adjacency list as empty by making head as NULL
        for (int i = 0; i < V; ++i)
                graph->array[i].head = NULL;
        return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest, int weight) {
// Add an edge from src to dest.  A new node is added to the adjacency
// list of src.  The node is added at the begining
        struct AdjListNode* newNode = newAdjListNode(dest, weight);
        newNode->next = graph->array[src].head;
        graph->array[src].head = newNode;
// Since graph is undirected, add an edge from dest to src also
        newNode = newAdjListNode(src, weight);
        newNode->next = graph->array[dest].head;
```

```
        graph->array[dest].head = newNode;
}
void PrimMST(struct Graph* graph) {
//Write the code here
// Implement either with Priority queue or MinHeap
}
int main() {
// Creating the graph
        int V = 9;
        struct Graph* graph = createGraph(V);
        addEdge(graph, 0, 1, 4);
        addEdge(graph, 0, 7, 8);
        addEdge(graph, 1, 2, 8);
        addEdge(graph, 1, 7, 11);
        addEdge(graph, 2, 3, 7);
        addEdge(graph, 2, 8, 2);
        addEdge(graph, 2, 5, 4);
        addEdge(graph, 3, 4, 9);
        addEdge(graph, 3, 5, 14);
        addEdge(graph, 4, 5, 10);
        addEdge(graph, 5, 6, 2);
        addEdge(graph, 6, 7, 1);
        addEdge(graph, 6, 8, 6);
        addEdge(graph, 7, 8, 7);
        PrimMST(graph);
        return 0;
}
```

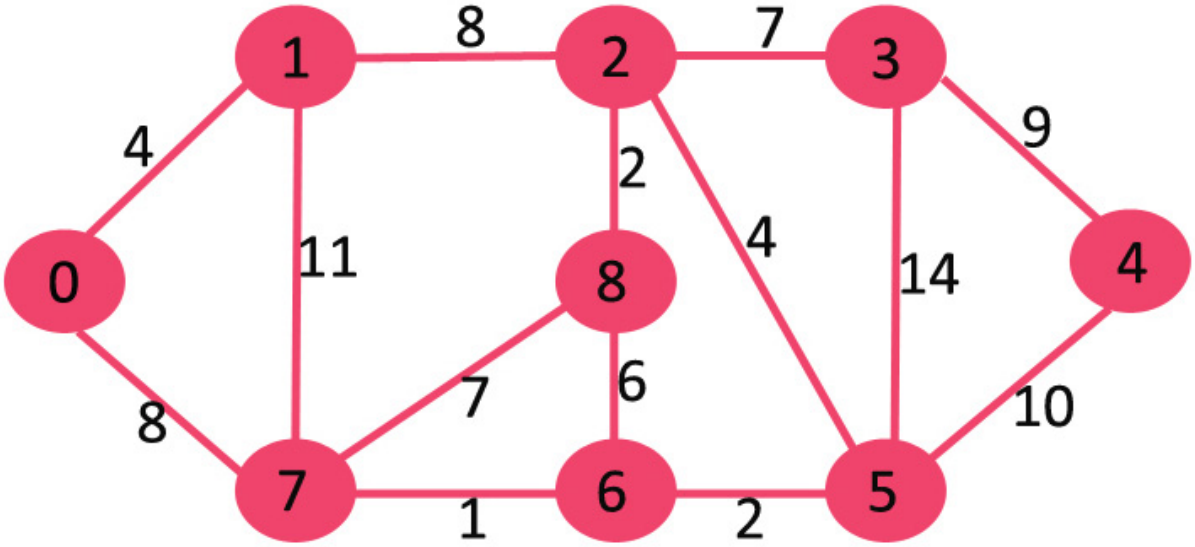

Figure 1: Given Graph

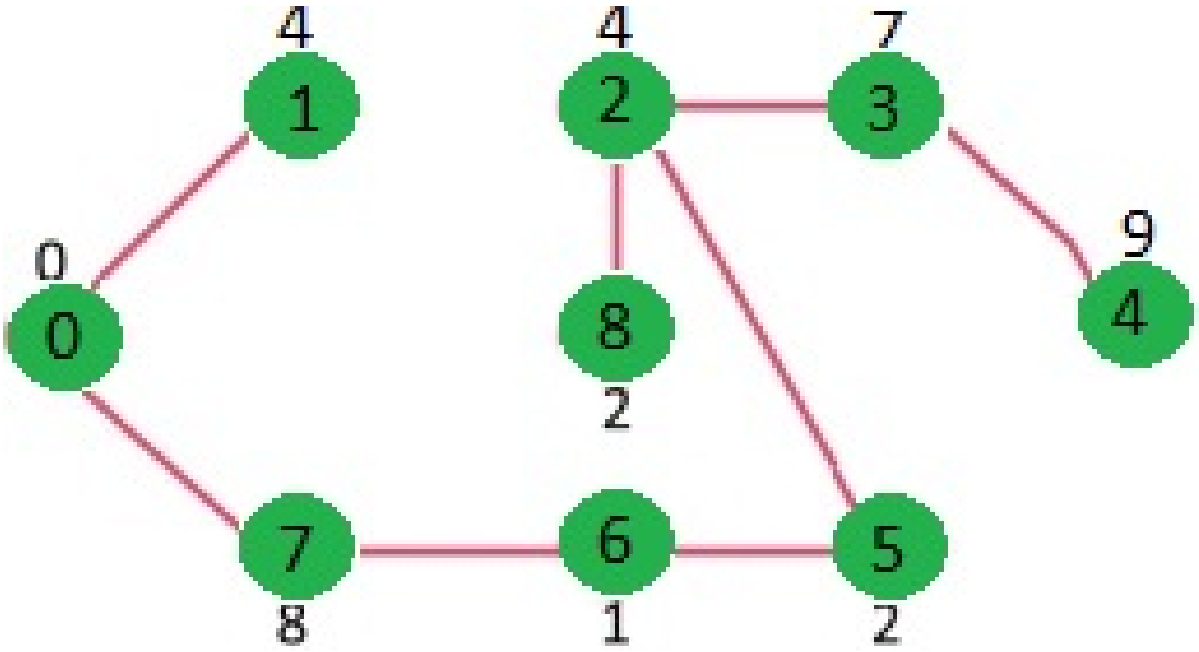

Figure 2: MST of the tree