Project Report On

Application Of Prim Algorithm

Submitted By:

Ayush Jaiswal (19103185)

Kshitij Shakya (19103181)

Vansh Sachdeva (1903194)

(B6)

Under Guidance Of:

Dr. Parul Agarwal

Assistant Professor (Senior Grade)

Jaypee Institute Of Information Technology, Sector-62 Noida

Acknowledgement

We would like to express my special thanks of gratitude to my teacher Ms. Parul Agarwal, Our Tutorial Teacher and as well as our Lecture Teacher, Ms. Sarishty Gupta who gave me the golden opportunity to do this wonderful project on the topic "Application Off Prim's Algorithm for Minimum Spanning tree", which also helped us in doing a lot of Research and We came to know about so many new things and we am really thankful to them.

Secondly, we would also like to thank my parents and friends who helped us a lot in finalizing this project within the limited time frame.

Softwares Used

1) Visual Studio Code:

Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS.

2) GitBash:

Git Bash is a source control management system for Windows. It allows users to type **Git** commands that make source code management easier through versioning and commit history.

3) MinGW:

MinGW, formerly mingw32, is a free and open source software development environment to create Microsoft Windows applications.

Things which can be improved:

Since this is a mini project, not a minor project, all we have done is just create small prototype.

But, we can create a map using Graph (Eg: Map of a city) and using that map and input given by the Delivery Person, we can create a Sub-Graph, having only the Nodes, i.e, Places that the Delivery Guy has to visit.

Then, we can use Prim's and Dijkstra's Algorithm. We can get the **best optimised route** for the Delivery Guy, saving a lot of resources like Time, Petrol and increasing the effiency of delivering service, providing much happier experience for the customers.

We can use template classes to add any other types of data, if any.

We can also add more additional options like:

- Login page,
- Customer Review Page,
- Customer Feedback(like time between which the delivery has to be made.)

General Information

- **Abstract**: -

How does Prim's Algorithm Work? The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

- **Algorithm**

1) Create a set mstSet that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3) While mstSet doesn't include all vertices

a) Pick a vertex u which is not there in mstSet and has minimum key value.

b) Include u to mstSet.

c) Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

- **General Information: -**
  - Compiler Used: - gcc version 9.2.0 (MinGW.org GCC Build-2)
  - Text Editor Used: - Visual Studio Code

- Terminal Used: - GNU bash, version 4.4.23(1)-release (x86_64-pc-msys)

- **Functions And Class(es) Used: -**
  - AdjacentList (class) : - used for storing graph nodes
  - AdjacentList (class): - class used to create Adjacent List (List Of List AKA MultiList)
  - Graph (class): - class used to store graph using AdjacentList Method.
    - newAdjListNode(function): - function used to add nodes to list.
    - Create(int): - function used for creating Sub list in list created by above mentioned function(newAdjListNode)
    - AddEdge(int,int,int): - function used for adding edges in the graph
  - HeapNode(class): - used for creating nodes for heaps
  - MinHeap(class): - used for creating a min binary heap
    - NewHeapNode (function): - function used to create new nodes in the min binary heap
    - CreateHeap(int) (function): - function used to create minheap
    - SwapNode(HeapNode<T> **, HeapNode<T> **) (function) : - Function Used To Swap two heap nodes.
    - Heapify() (Function) : - function used to convert the data inserted using above functions into proper min binary heap
    - isEmpty() (function) : - function used to check whether the heap is empty or not.
    - extractMin() (function): - function used to return the min value of data from heap.
    - decreaseKey(int Vertice,int Key) (function): - function used to decrease key values of nodes acc' to there relative position in the heap.
  - PrintArr(int [],int n) (function) function used to print the elements of an array having size n.

- PrimMST(Graph<> *) (function) : - function which is going to perform prim's algorithm on graph we inserted above using min-heap method.

• Space-Time Complexity: -

The time complexity of the above code/algorithm looks O(V^2) as there are two nested while loops. If we take a closer look, we can observe that the statements in inner loop are executed O(V+E) times (similar to BFS). The inner loop has decreaseKey() operation which takes O(LogV) time. So overall time complexity is O(E+V)*O(LogV) which is O((E+V)*LogV) = O(ELogV) (For a connected graph, V = O(E))

Code: -

```
/*
Mini Project

Subject: - Data Structure
Subject Code: -

Topic: - Prim's Algorthim
 Made By: -
Vansh Sachdev
 Ayush Jaiswal
 Kshitij Shakya

 */
#include <bits/stdc++.h>
using namespace std;
// namespace
class node
{
public:
    int val;
    int weight;
    node *next;
};

class edge
{
public:
    int weight;
    int u, v; // edge from u ---> v
};

// Adds an edge to an Adjacency List element
node *AddEdge(node *head, int num, int weight)
{
    node *p = new node;

    p->val = num;
    p->next = head;
    p->weight = weight;
    head = p;
```

```
    return p;
}

// enqueues an entry into the Priority Queue
void enqueue(edge heap[], int size, edge value)
{
    heap[size] = value;

    int i = size;
    edge temp;

    while (i >= 1)
    {
        if (heap[i / 2].weight > heap[i].weight)
        {
            temp = heap[i / 2];
            heap[i / 2] = heap[i];
            heap[i] = temp;

            i = i / 2;
        }
        else
        {
            break;
        }
    }
}

void heapify(edge heap[], int size, int index)
{
    int i = index;
    edge temp;

    while ((2 * i) < size)
    {
        if ((2 * i) + 1 >= size)
        {
            if (heap[i].weight > heap[2 * i].weight)
            {
                temp = heap[i];
                heap[i] = heap[2 * i];
                heap[2 * i] = temp;

                break;
```

```
            }
        }

        if (heap[i].weight > heap[2 * i].weight || heap[i].weight > heap[2
 * i + 1].weight)
        {
            if (heap[2 * i].weight <= heap[(2 * i) + 1].weight)
            {
                temp = heap[2 * i];
                heap[2 * i] = heap[i];
                heap[i] = temp;

                i = 2 * i;
            }
            else if (heap[2 * i].weight > heap[(2 * i) + 1].weight)
            {
                temp = heap[(2 * i) + 1];
                heap[(2 * i) + 1] = heap[i];
                heap[i] = temp;

                i = (2 * i) + 1;
            }
        }
        else
        {
            break;
        }
    }
}

// Deletes and entry in the Priority Queue
void deleteNode(edge heap[], int size, int index)
{
    edge temp = heap[index];

    heap[index] = heap[size - 1];
    heap[size - 1] = temp;

    int i = index;
    --size;

    heapify(heap, size, i);
}

// Returns the element with
```

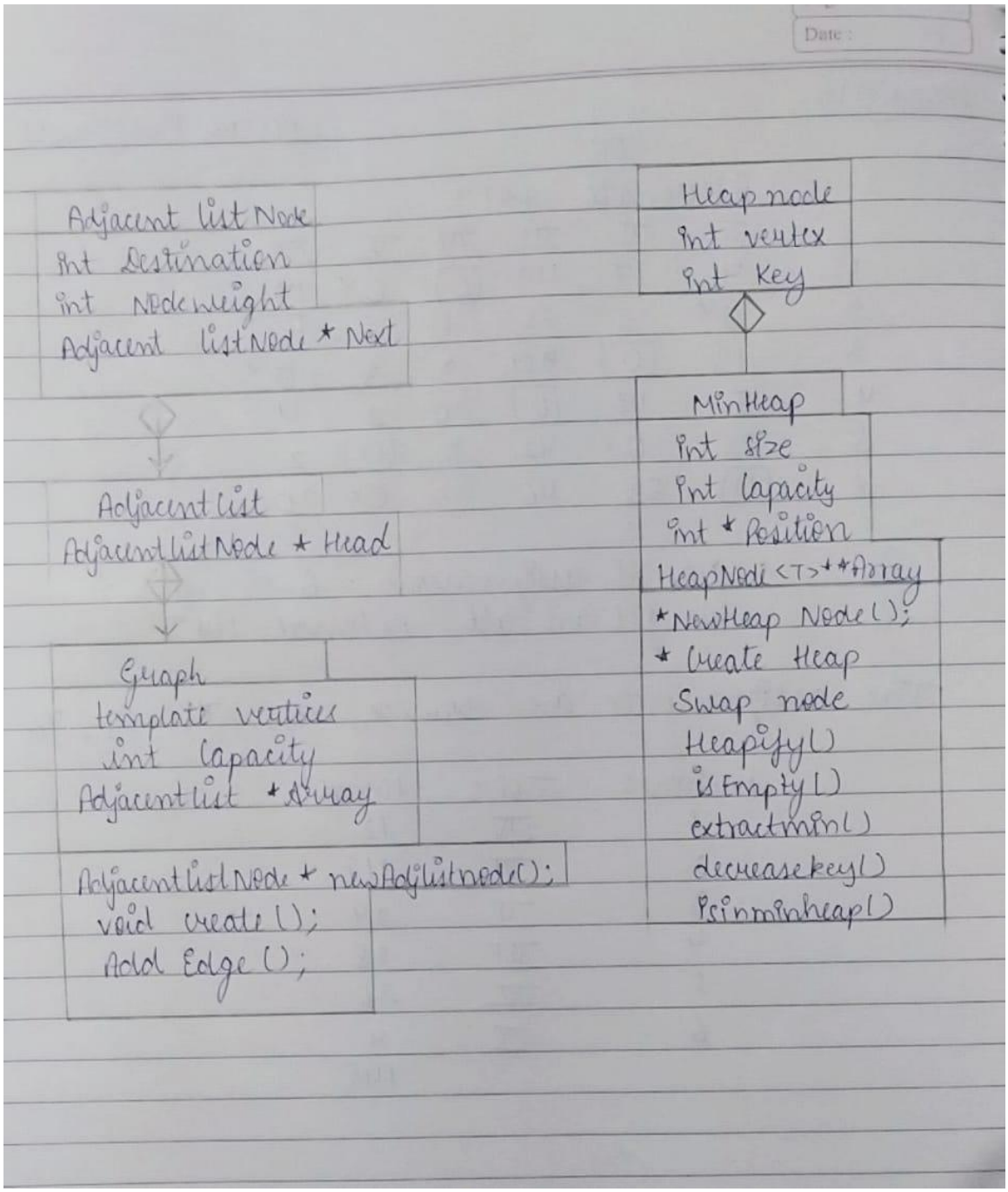
```
// Minimum Priority and deletes it
edge extractMin(edge heap[], int size)
{
    edge min = heap[0];

    deleteNode(heap, size, 0);

    return min;
}

// PrimMST's Algorithm function
void PrimMST(node *adjacencylist[], int vertices, int edges, int startVert
ex, node *MST[])
{
    int current = startVertex, newVertex;
    int visited[vertices + 1];
    node *temp;
    edge var;
    edge priorityQueue[2 * edges];
    int queueSize = 0;

    int i;

    for (i = 0; i <= vertices; ++i)
    { // Initializing
        visited[i] = 0;
    }

    i = 0;

    while (i < vertices)
    {
        if (visited[current] == 0)
        {                          // If current node is not visited
            visited[current] = 1; // Mark it visited
            temp = adjacencylist[current];

            while (temp != NULL)
            {
                var.u = current;
                var.v = temp->val;
                var.weight = temp->weight;

                if (!visited[var.v])
                {
```

```
                    // If the edge leads to an un-visited
                    // vertex only then queueSize it
                    enqueue(priorityQueue, queueSize, var);
                    ++queueSize;
                }

                temp = temp->next;
            }

            var = extractMin(priorityQueue, queueSize); // The greedy choi
ce
            --queueSize;

            newVertex = var.v;
            current = var.u;

            if (visited[newVertex] == 0)
            {
                // If it leads to an un-visited vertex, add it to MST
                MST[current] = AddEdge(MST[current], newVertex, var.weight
);
                MST[newVertex] = AddEdge(MST[newVertex], current, var.weig
ht);
            }

            current = newVertex;
            ++i;
        }
        else
        {
            var = extractMin(priorityQueue, queueSize);
            --queueSize;
            newVertex = var.v;
            current = var.u;

            if (visited[newVertex] == 0)
            {
                MST[current] = AddEdge(MST[current], newVertex, var.weight
);
                MST[newVertex] = AddEdge(MST[newVertex], current, var.weig
ht);
            }

            current = newVertex;
        }
```

```
    }
}

int main()
{
    int vertices, edges, i, j, v1, v2, weight;

    cout << "Enter the Number of Places You Have To Deliver -\n";
    cin >> vertices;

    cout << "Enter the Number of Connections Between Places -\n";
    cin >> edges;

    node *adjacencyList[vertices + 1];
    node *MST[vertices + 1];

    for (i = 0; i <= vertices; ++i)
    {
        adjacencyList[i] = NULL;
        MST[i] = NULL;
    }

    for (i = 1; i <= edges; ++i)
    {
        cout << "Enter Source ";
        cin >> v1;
        cout << "Enter Distination ";
        cin >> v2;
        cout << "Enter Weight ";
        cin >> weight;
        adjacencyList[v1] = AddEdge(adjacencyList[v1], v2, weight); //Addi
ng edge v1 ---W---> v2
        adjacencyList[v2] = AddEdge(adjacencyList[v2], v1, weight); //Addi
ng edge v2 ---W---> v1
    }

    // Printing Adjacency List
    cout << "\nAdjacency List -\n\n";
    for (i = 1; i <= vertices; ++i)
    {
        cout << "adjacencyList[" << i << "]";

        node *temp = adjacencyList[i];

        while (temp != NULL)
```

```
        {
            cout << temp->val << " (" <<temp->weight << ") ";
            temp = temp->next;
        }

        cout << "\n";
    }

    int startVertex;

    cout << "\nEnter a Start Vertex - ";
    cin >> startVertex;
    PrimMST(adjacencyList, vertices, edges, startVertex, MST);

    // Printing MST
    cout << "\nMinimum Spanning Tree -\n\n";
    for (i = 1; i <= vertices; ++i)
    {
        cout << "MST[" << i << "] ";

        node *temp = MST[i];

        while (temp != NULL)
        {
            cout << temp->val <<   " ";
            temp = temp->next;
        }

        cout << "\n";
    }

    return 0;
}
```

Class Diagrams



I/O Outputs

Taking Information About Number Of Places Meant To Be visited

```
vansh@MrArthor-Pc MINGW64 ~/Desktop/Stuff/Google Driv/Codes/C++
des/C++
$ run
Enter the Number of Places You Have To Deliver -
6
Enter the Number of Connections Between Places -
9
```

Taking Information about those places and there connections

```
Enter Source 1
Enter Distination 3
Enter Weight 4
Enter Source 1
Enter Distination 5
Enter Weight 5
Enter Source 3
Enter Distination 4
Enter Weight 2
Enter Source 3
Enter Distination 2
Enter Weight 3
Enter Source 3
Enter Distination 5
Enter Weight 3
Enter Source 5
Enter Distination 6
Enter Weight 4
Enter Source 5
Enter Distination 2
Enter Weight 2
Enter Source 2
Enter Distination 6
Enter Weight 3
Enter Source 4
Enter Distination 2
Enter Weight 1
```

Showing The Input entered by user

```
Adjacency List -

adjacencyList[1]5 (5) 3 (4)
adjacencyList[2]4 (1) 6 (3) 5 (2) 3 (3)
adjacencyList[3]5 (3) 2 (3) 4 (2) 1 (4)
adjacencyList[4]2 (1) 3 (2)
adjacencyList[5]2 (2) 6 (4) 3 (3) 1 (5)
adjacencyList[6]2 (3) 5 (4)
```

Showing Best Route

```
Enter a Start Vertex - 1

Minimum Spanning Tree -

MST[1] 3
MST[2] 6 5 4
MST[3] 4 1
MST[4] 2 3
MST[5] 2
MST[6] 2
```