**ASSIGNMENT 4**: GREEDY APPROACH

**Name**  : Muhle Musa Khumalo

**StudentID** : 410921335

**Date**  : April 29, 2022

Prim’s Algorithm

// A C++ program for Prim's Minimum

// Spanning Tree (MST) algorithm. The program is

// for adjacency matrix representation of the graph

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 10

#define I INT\_MAX

// A utility function to find the vertex with

// minimum key value, from the set of vertices

// not yet included in MST

int minKey(int key[], bool mstSet[])

{

// Initialize min value

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;

}

// A utility function to print the

// constructed MST stored in parent[]

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

{

cout<<"Edge \tWeight\n";

for (int i = 1; i < V; i++)

cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";

}

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation

void primMST(int graph[V][V])

{

// Array to store constructed MST

int parent[V];

// Key values used to pick minimum weight edge in cut

int key[V];

// To represent set of vertices included in MST

bool mstSet[V];

// Initialize all keys as INFINITE

for (int i = 0; i < V; i++)

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

// Always include first 1st vertex in MST.

// Make key 0 so that this vertex is picked as first vertex.

key[0] = 0;

parent[0] = -1; // First node is always root of MST

// The MST will have V vertices

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

{

// Pick the minimum key vertex from the

// set of vertices not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

// Update key value and parent index of

// the adjacent vertices of the picked vertex.

// Consider only those vertices which are not

// yet included in MST

for (int v = 0; v < V; v++)

// graph[u][v] is non zero only for adjacent vertices of m

// mstSet[v] is false for vertices not yet included in MST

// Update the key only if graph[u][v] is smaller than key[v]

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

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

cout << "v" << u+1 << " added to MST."<< endl;

}

// print the constructed MST

printMST(parent, graph);

}

// Driver code

int main()

{

int graph[V][V] = { { 0, 32, I, 17, I, I, I, I, I, I },

{ 32, 0, I, I, 45, I, I, I, I, I },

{ I, I, 0, 18, I, I, 5, I, I, I },

{ 17, I, 18, 0, 10, I, I, 3, I, I },

{ I, 45, I, 10, 0, 28, I, I, 25, I },

{ I, I, I, I, 28, 0, I, I, I, 6},

{ I, I, 5, I, I, I, 0, 59, I, I},

{ I, I, I, 3, I, I, 59, 0, 4, I},

{ I, I, I, I, 25, I, I, 4, 0, 12},

{ I, I, I, I, I, 6, I, I, 12, 0} };

// Print the solution

primMST(graph);

return 0;

}

**Output**

Text

Description automatically generated



**Graphical Representation**



**Prim’s Algorithm**  
Greedy Aproach MST covered all vertices and mininum weight edges  
Selects the minimum edge among the avalable, Start with vertex v1

STEP 1



STEP 2



Step 3



Step 4



Step 5



Step 6



Step 7



Step 8



Step 9



Step 10



Final Minimum Spanning Tree



The minimum weight is **107**.

**Kruskal Algorithm**

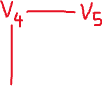
Step 1 Step 2



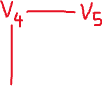
Step 3 Step 4



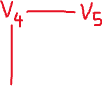
Step 5



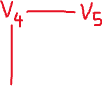
Step 6



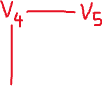
Step 7



Step 8



Step 9



**Dijkstra’s Algorithm**

Step 1) V4→V8                           =>3  
Step 2) V4→V8→V9                   =>7  
Step 3) V4→V5                            =>10  
Step 4) V4→V1                           =>17  
Step 5) V4→V3                           =>18  
Step 6) V4→V8→V9→V10         =>1  
Step 7) V4→V3→V7                   =>23  
Step 8) V4→V8→V9→V10→V6 =>25  
Step 9) V4→V1→V2                 =>49

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Step** | **Current** **Node** | **V1** | **V2** | **V3** | **V5** | **V6** | **V7** | **V8** | **V9** | **V10** |
| 1 | V4 | 17,v4 | ∞ | 18,v4 | 10,v4 | ∞ | ∞ | 3,v4 | ∞ | ∞ |
| 2 | V8 | 17,v4 | ∞ | 18,v4 | 10,v4 | ∞ | 62,v8 |  | 7,v8 | ∞ |
| 3 | V9 | 17,v4 | ∞ | 18,v4 | 10,v4 | ∞ | 62,v8 |  |  | 19,v8 |
| 4 | V5 | 17,v4 | 55,v5 | 18,v4 |  | 38,v5 | 62,v8 |  |  | 19,v8 |
| 5 | V1 |  | 49,v1 | 18,v4 |  | 38,v5 | 62,v8 |  |  | 19,v8 |
| 6 | V3 |  | 49,v1 |  |  | 38,v5 | 23,v3 |  |  | 19,v8 |
| 7 | V10 |  | 49,v1 |  |  | 25,v10 | 23,v3 |  |  |  |
| 8 | V7 |  | 49,v1 |  |  | 25,v10 |  |  |  |  |
| 9 | V6 |  | 49,v1 |  |  |  |  |  |  |  |

**Problem 19**

First arrange all jobs by service time in ascending order.

|  |  |
| --- | --- |
| Job | Service time |
| 2 | 3 |
| 4 | 5 |
| 1 | 7 |
| 3 | 10 |

Calculating total time spent by job schedule.

Total time spent

[2 4 1 3] = 3+(3+5)+(3+5+7)+(3+5+7+10)

job2 = 3

job4 = 3+5 = 8

job1 = 3+5+7 = 15

job3 = 3+5+7+10 = 25

Total time spent = 3+8+15+25 = 51

**Problem 22**

Let j represent a job

Now let us take j which has maximum deadline which is 4

Deadline=4

Profit=15

Now take j which has deadline >= 4-1 with higher profit

Profit = 15+60 = 75

Now take j which has deadline >= 3-1 with higher profit

Profit = 15+60+40

Lastly take j which has deadline >=2-1 with higher profit

Profit = 15+60+40+55

Therefore, the maximum profit will be **170.**