PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

PRIM'S ALGORITHM

Submitted by

Nagabhushan Deshpande (PES1UG21EC155)

Nishant Sreekanteswar (PES1UG21EC915)

Meghana Devanand (PES1UG21EC147)

August – December 2023

under the guidance of

Prof. Rajeshwari B
Associate Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PES University

Bengaluru -560085

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PROGRAM B. TECH

CONTENTS

PROBLEM STATEMENT

Solving Prim’s algorithm using assembly level code.

THEORY

Like Kruskal's algorithm, Prim's algorithm is also a Greedy algorithm. This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.

The algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

ALGORITHM:

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.

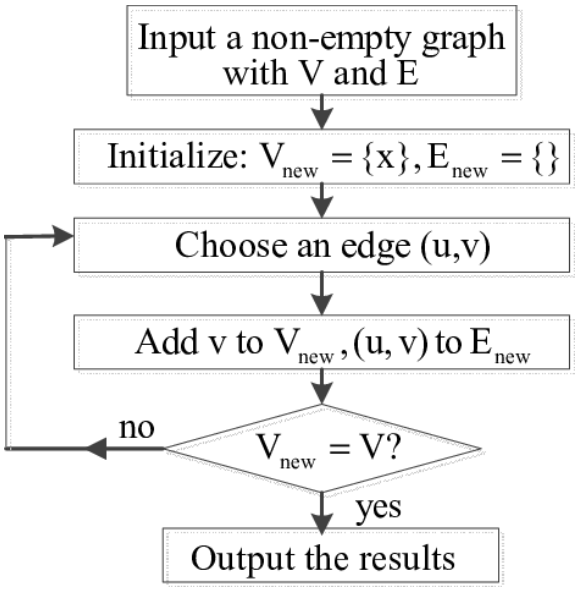
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).

Step 3: Find edges connecting any tree vertex with the fringe vertices.

Step 4: Find the minimum among these edges.

Step 5: Add the chosen edge to the MST if it does not form any cycle.

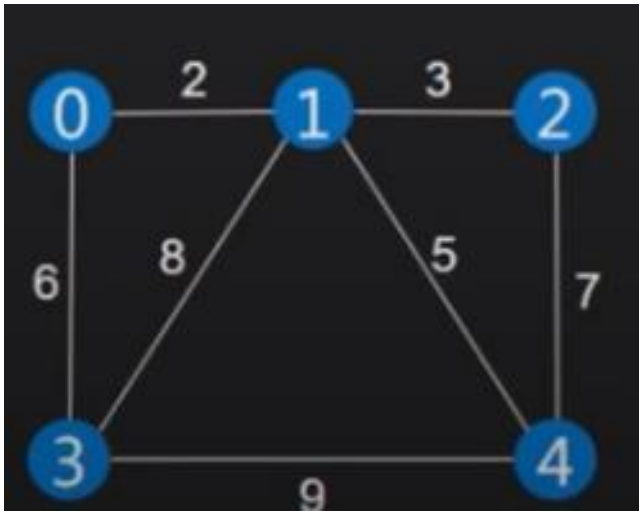
Step 6: Return the MST and exit



CODE

GRAPH FIGURE:

Number of vertices =5



ASSEMBLY CODE FOR ABOVE GRAPH:

```
.data
graph: .word  0, 2, 0, 6, 0
       .word  2, 0, 3, 8, 5
       .word  0, 3, 0, 0, 7
       .word  6, 8, 0, 0, 9
       .word  0, 5, 7, 9, 0
.equ vertex 5
parent: .zero 5*4
key: .zero 5*4
mstset: .zero 5*4

.text
li x1,1 #true
la x11,parent
la x12,key
la x13,mstset
la x21,graph

li x2,vertex    #vertex=5=x2
li x3,0         #counter1 x3=0
li x6,99        #max value
li x7,0         #false value

#for loop
forloopra:
bge x3,x2,whatnext #x3=0 x2=5
slli x4,x3,2
add x5,x12,x4      #keybase-x12
add x8,x13,x4      #mstsetbase-x13
sw x6,0(x5)        #store 99 (max value ) in every key
sw x7,0(x8)        #initializing mstset to false/0
addi x3,x3,1
j forloopra

```

```
whatnext:
li x10,-1
sw x0,0(x12)    ##initializing key(0) to 0
sw x10,0(x11)   #initializing parent(0) to -1

li x3,0             #counter x3=0
addi x9,x2,-1       # x9 = V-1 = 4
code:bge x3,x9,whatnext2
# minkey function
li x18,0      #v counter in minkey function
li x19,99     #minimum
li x30,0      #MIN-index
something:bge x18,x2,endminkey #for loop
slli x24,x18,2
add x25,x13,x24    #mstset
add x26,x12,x24    #key
lw x27,0(x25)
lw x28,0(x26)
bne x27,x7,endif
bge x28,x19,endif
addi x19,x28,0
addi x30,x18,0 # returning the value u from minkey function
endif:
addi x18,x18,1
j something
endminkey:
slli x31,x30,2
add x31,x31,x13
sw x1,0(x31) #mstset{u}=true

li x15,0
anything:bge x15,x2,endinnerloop
mul x20,x30,x2
add x20,x20,x15
slli x20,x20,2
add x20,x20,x21         #to find address of u*V+v
lw x24,0(x20)           #loading value of graphuv
slli x23,x15,2
add x18,x13,x23
lw x22,0(x18)           # loading value of mstset(v)
add x25,x23,x12
lw x26,0(x25)           #loading value of key{v}
beq x24,x0,endifcondition  # mstSet[v] == false
bne x22,x7,endifcondition  # graph[u][v]
bge x24,x26,endifcondition # graph[u][v] < key[v]
add x28,x23,x11
sw x30,0(x28)              #parent(u) = u
sw x24,0(x25)              #key[v] = graph[u][v]
endifcondition:
addi x15,x15,1   #increment the inner forloop counter
j anything
endinnerloop:
addi x3,x3,1     #increment the outer forloop counter
j code
whatnext2:nop
```

OUTPUT

STORING THE KEY VALUES IN GRAPH:

0x1000004c	9	9	0	0	0
0x10000048	0	0	0	0	0
0x10000044	0	0	0	0	0
0x10000040	8	8	0	0	0
0x1000003c	6	6	0	0	0
0x10000038	7	7	0	0	0
0x10000034	0	0	0	0	0
0x10000030	0	0	0	0	0
0x1000002c	3	3	0	0	0
0x10000028	0	0	0	0	0
0x10000024	5	5	0	0	0
0x10000020	8	8	0	0	0
0x1000001c	3	3	0	0	0
0x10000018	0	0	0	0	0
0x10000014	2	2	0	0	0
0x10000010	0	0	0	0	0
0x1000000c	6	6	0	0	0
0x10000008	0	0	0	0	0
0x10000004	2	2	0	0	0
0x10000000	0	0	0	0	0

PARENT VALUES FOR RANGE 0 TO 4 (V-1 WHERE V OR NO. OF VERTICES=5):

0x10000074	1	1	0	0	0
0x10000070	0	0	0	0	0
0x1000006c	1	1	0	0	0
0x10000068	0	0	0	0	0
0x10000064	-1	255	255	255	255

KEYS OR WEIGHTS OF THE EDGES WHICH ARE A PART OF THE MINIMUM SPANNING TREE:

0x10000088	5	5	0	0	0
0x10000084	6	6	0	0	0
0x10000080	3	3	0	0	0
0x1000007c	2	2	0	0	0
0x10000078	0	0	0	0	0

STORES THE MSTSET VALUE:

(If the edge is a part of the minimum spanning tree , mstset value is set as true else it retains it's default value, which is , false.)

0x1000009c	1	1	0	0	0
0x10000098	0	0	0	0	0
0x10000094	1	1	0	0	0
0x10000090	1	1	0	0	0
0x1000008c	1	1	0	0	0

CODE EXPLANATION

This assembly code appears to be an implementation of Prim's algorithm for finding the minimum spanning tree of a connected, undirected graph. It initializes arrays to store information about vertices, sets initial values, and then iteratively selects the minimum-weight edge to expand the minimum spanning tree.

STEPS:

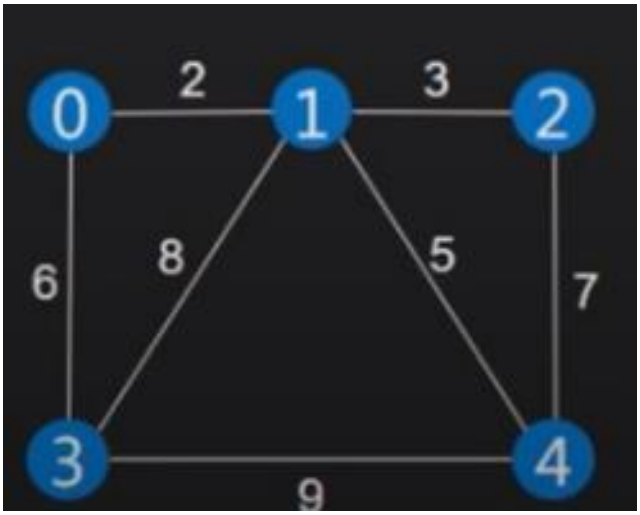
1. The .data section initializes a graph with 5 vertices, and .equ vertex 5 sets the variable vertex to 5. Three arrays (parent, key, and mstset) are initialized with 20 bytes (5 * 4) each to store parent pointers, key values, and MST set information.
2. Initialize the key array with 99 (max value) and the mstset array with 0 (false) for all vertices.
3. Initialize the key for the first vertex to 0 and the parent to -1.
4. Initializes counters for loops.
5. Implement the minkey function to find the minimum key value among the vertices that are not in the MST set. Sets the corresponding vertex as part of the MST set.
6. Update Key and Parent Arrays**:** For each vertex v adjacent to u:

If v is not in the MST set and the weight of the edge (u, v) is less than the current key value of v, update key[v] with the weight and set parent[v] to u.

7. The iterations occur until all the vertices are included in the minimum spanning tree.

THEORETICAL SOLUTION

GRAPH:



Number of vertices=5

Parent node=0

STEPS:

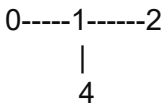
1. 0 is the first (parent node). All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 3}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.

   0------1

2. The edges connecting the incomplete MST to other vertices are {0, 3}, {1, 2},{1,4} and {1, 3}. Among these edges the minimum weight is 3 which is of the edge{1, 2}. Let us here include the edge {1,2} and the vertex 2 in the MST.

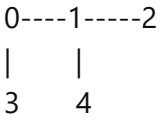
   0-----1------2

3. IN the next iteration, }. Among the edges the minimum weight is 5 which is of the edge{1, 4}. Let us here include the edge {1,4} and the vertex 4 in the MST.



4. Since vertex 3 is left, compare the edges connected to 3 and find the minimum weight. }. Among the edges the minimum weight is 6 which is of the edge{0,3}. Let us here include the edge {0,3} and the vertex 3 in the MST.

   Thus, following is the result obtained for MST:



CONCLUSION

The provided assembly code implements Prim's algorithm for finding the minimum spanning tree (MST) of a connected, undirected graph.

The code demonstrates correctness in its logic, effectively navigating through vertices and edges to identify the minimum-weight edges at each step. The time complexity, estimated at O(V^2) for the provided adjacency matrix representation, highlights a reasonable level of efficiency in handling the graph structure. Overall, this assembly code provides a reliable and efficient implementation of Prim's algorithm for finding the minimum spanning tree in a connected graph.