

Print all the cycles in an undirected graph
Detect cycle in an undirected graph using BFS
Samsung Semiconductor Institute of Research(SSIR Software) Intern/FTE Set-3
Check if a given graph is Bipartite using DFS
Largest connected component on a grid
Nutanix Interview Experience (On-Campus)
Kruskal's Algorithm (Simple Implementation for Adjacency Matrix)
Detect Cycle in a Directed Graph using BFS
Find the number of distinct islands in a 2D matrix
Sum of the minimum elements in all connected components of an undirected graph
Tree, Back, Edge and Cross Edges in DFS of Graph
Level Ancestor Problem
Maximum number of edges among all connected components of an undirected graph
Minimum cost path from source node to destination node via an intermediate node
Johnson's algorithm for All-pairs shortest paths Implementation
Dijkstra's shortest path with minimum edges
Applications of Graph Data Structure
Coloring a Cycle Graph
Number of Isosceles triangles in a binary tree
Edge Coloring of a Graph
Prim's Algorithm (Simple Implementation for Adjacency Matrix Representation)
Minimum time to return array to its original state after given modifications
Find the probability of a state at a given time in a Markov chain Set 1
Number of Walks from source to destination
Find whether it is possible to finish all tasks or not from given dependencies
Find the ordering of tasks from given dependencies
Subtree of all nodes in a tree using DFS
Graph Types and Applications
Jump Pointer Algorithm
Find maximum path length in a binary matrix

Prim's MST for Adjacency List Representation | Greedy Algo-6

We recommend to read following two posts as a prerequisite of this post.

- Greedy Algorithms | Set 5 (Prim's Minimum Spanning Tree (MST))
- Graph and its representations

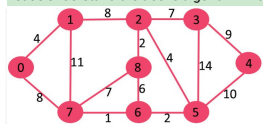
We have discussed [Prim's algorithm](#) and its implementation for adjacency matrix representation of graphs. The time complexity for the matrix representation is $O(V^2)$. In this post, $O(E \log V)$ algorithm for adjacency list representation is discussed.

As discussed in the previous post, in Prim's algorithm, two sets are maintained, one set contains list of vertices already included in MST, other set contains vertices not yet included. With adjacency list representation, all vertices of a graph can be traversed in $O(V+E)$ time using [BFS](#). The idea is to traverse all vertices of graph using [BFS](#) and use a Min Heap to store the vertices not yet included in MST. Min Heap is used as a priority queue to get the minimum weight edge from the [cut](#). Min Heap is used as time complexity of operations like extracting minimum element and decreasing key value is $O(\log V)$ in Min Heap.

Following are the detailed steps.

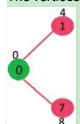
- Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and key value of the vertex.
- Initialize Min Heap with first vertex as root (the key value assigned to first vertex is 0). The key value assigned to all other vertices is INF (infinite).
- While Min Heap is not empty, do following
 - Extract the min value node from Min Heap. Let the extracted vertex be u .
 - For every adjacent vertex v of u , check if v is in Min Heap (not yet included in MST). If v is in Min Heap and its key value is more than weight of $u-v$, then update the key value of v as weight of $u-v$.

Let us understand the above algorithm with the following example:

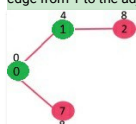


Initially, key value of first vertex is 0 and INF (infinite) for all other vertices. So vertex 0 is extracted from Min Heap and key values of vertices adjacent to 0 (1 and 7) are updated. Min Heap contains all vertices except vertex 0.

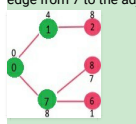
The vertices in green color are the vertices included in MST.



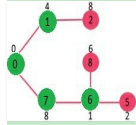
Since key value of vertex 1 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 1 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 1 to the adjacent). Min Heap contains all vertices except vertex 0 and 1.



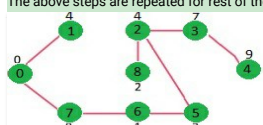
Since key value of vertex 7 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 7 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 7 to the adjacent). Min Heap contains all vertices except vertex 0, 1 and 7.







Since key value of vertex 6 is minimum among all nodes in Min Heap, it is extracted from Min Heap and key values of vertices adjacent to 6 are updated (Key is updated if the a vertex is not in Min Heap and previous key value is greater than the weight of edge from 6 to the adjacent). Min Heap contains all vertices except vertex 0, 1, 7 and 6.



The above steps are repeated for rest of the nodes in Min Heap till Min Heap becomes empty



C++ Java Python



```
// Java program for Prim's MST for
// adjacency list representation of graph
import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Comparator;

public class prims {
    class node1 {

        // Stores destination vertex in adjacency list
        int dest;
```

Most popular in Graph
De Bruijn sequence Set 1
Degree of a Cycle Graph
Maximum and minimum isolated vertices in a graph
Total number of Spanning trees in a Cycle Graph
Minimum difference between the highest and the smallest value of mines distributed

Most visited in Greedy
Maximum sum by picking elements from two arrays in order
Coin game of two corners (Greedy Approach)
Maximum sum of all elements of array after performing given operations
Check if an array is Wave Array
Water drop problem



Geeks Classes

Classroom program on Algorithms in Noida
Mentored by Mr. Sandeep Jain

Batch starts from **22nd Jan, 2019**
Classes on **Tuesday and Friday**



```
// Stores weight of a vertex in adjacency list
int weight;

// Constructor
node1(int a, int b)
{
    dest = a;
    weight = b;
}

static class Graph {

    // Number of vertices in the graph
    int V;

    // List of adjacent nodes of a given vertex
    LinkedList<node1>[] adj;

    // Constructor
    Graph(int e)
    {
        V = e;
        adj = new LinkedList[V];
        for (int o = 0; o < V; o++)
            adj[o] = new LinkedList<>();
    }

    // class to represent a node in PriorityQueue
    // Stores a vertex and its corresponding
    // key value
    class node {
        int vertex;
        int key;
    }

    // Comparator class created for PriorityQueue
    // returns 1 if node0.key > node1.key
    // returns 0 if node0.key < node1.key and
    // returns -1 otherwise
    class comparator implements Comparator<node> {

        @Override
        public int compare(node node0, node node1)
        {
            return node0.key - node1.key;
        }
    }

    // method to add an edge
    // between two vertices
    void addEdge(Graph graph, int src, int dest, int weight)
    {
        node1 node0 = new node1(dest, weight);
        node1 node = new node1(src, weight);
        graph.adj[src].addLast(node0);
        graph.adj[dest].addLast(node);
    }

    // method used to find the mst
    void prims_mst(Graph graph)
    {
        // Whether a vertex is in PriorityQueue or not
        Boolean[] mstset = new Boolean[graph.V];
        node[] e = new node[graph.V];

        // Stores the parents of a vertex
        int[] parent = new int[graph.V];

        for (int o = 0; o < graph.V; o++)
            e[o] = new node();

        for (int o = 0; o < graph.V; o++) {

            // Initialize to false
            mstset[o] = false;

            // Initialize key values to infinity
            e[o].key = Integer.MAX_VALUE;
            e[o].vertex = o;
            parent[o] = -1;
        }

        // Include the source vertex in mstset
        mstset[0] = true;

        // Set key value to 0
        // so that it is extracted first
        // out of PriorityQueue
        e[0].key = 0;

        // PriorityQueue
        PriorityQueue<node> queue = new PriorityQueue<>(graph.V, new comparator());

        for (int o = 0; o < graph.V; o++)
            queue.add(e[o]);

        // Loops until the PriorityQueue is not empty
        while (!queue.isEmpty()) {

            // Extracts a node with min key value
            node node0 = queue.poll();

            // Include that node into mstset
            mstset[node0.vertex] = true;

            // For all adjacent vertex of the extracted vertex V
            for (node1 iterator : graph.adj[node0.vertex]) {

                // If V is in PriorityQueue
                if (mstset[iterator.dest] == false) {
                    // If the key value of the adjacent vertex is
                    // more than the extracted key
                    // update the key value of adjacent vertex
                    // to update first remove and add the updated vertex
                    if (e[iterator.dest].key > iterator.weight) {
                        queue.remove(e[iterator.dest]);
                        e[iterator.dest].key = iterator.weight;
                        queue.add(e[iterator.dest]);
                        parent[iterator.dest] = node0.vertex;
                    }
                }
            }
        }

        // Prints the vertex pair of mst
        for (int o = 1; o < graph.V; o++)
            System.out.println(parent[o] + " "
                                + "-"
                                + " " + o);
    }

    public static void main(String[] args)
    {
        int V = 9;

        Graph graph = new Graph(V);

        prims e = new prims();

        e.addEdge(graph, 0, 1, 4);
        e.addEdge(graph, 0, 7, 8);
    }
}
```

Most Trending Articles

Find postorder traversal of BST from preorder traversal
Implementation of Locking in DBMS
Sorting a Hashmap according to values
Find the smallest positive number missing from an unsorted array Set 2
Amazon Recruitment Process
Placement Preparation Guide
Flatten a binary tree into linked list
Unknown facts of Networking
Must have books for Placements Preparation
Java How to start learning Java
How does Floyd's slow and fast pointers approach work?
Top 10 algorithms in Interview Questions Set 2
StringBuffer appendCodePoint() Method in Java with Examples
DBMS Tuple Relational Calculus
map insert() in C++ STL

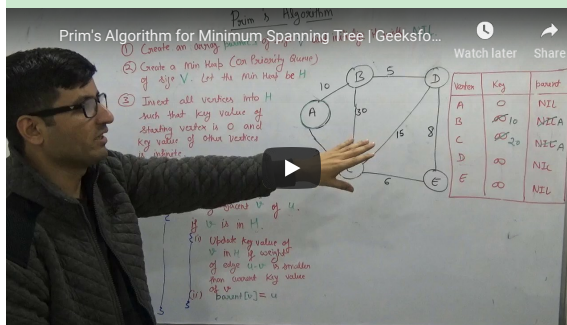
```
e.addEdge(graph, 1, 2, 8);
e.addEdge(graph, 1, 7, 11);
e.addEdge(graph, 2, 3, 7);
e.addEdge(graph, 2, 8, 2);
e.addEdge(graph, 2, 5, 4);
e.addEdge(graph, 3, 4, 9);
e.addEdge(graph, 3, 5, 14);
e.addEdge(graph, 4, 5, 10);
e.addEdge(graph, 5, 6, 2);
e.addEdge(graph, 6, 7, 1);
e.addEdge(graph, 6, 8, 6);
e.addEdge(graph, 7, 8, 7);

// Method invoked
e.prims_mst(graph);
}
}
// This code is contributed by Vikash Kumar Dubey
```

Output:

```
0 - 1
5 - 2
2 - 3
3 - 4
6 - 5
7 - 6
0 - 7
2 - 8
```

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 $O(\text{Log}V)$ time. So overall time complexity is $O(E+V)*O(\text{Log}V)$ which is $O((E+V)*\text{Log}V) = O(E\text{Log}V)$ (For a connected graph, $V = O(E)$)



References:

Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

http://en.wikipedia.org/wiki/Prim's_algorithm

This article is compiled by Aashish Barnwal and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Dijkstra's Algorithm for Adjacency List Representation | Greedy Algo-8
Prim's Algorithm (Simple Implementation for Adjacency Matrix Representation)
DFS for a n-ary tree (acyclic graph) represented as adjacency list
Kruskal's Algorithm (Simple Implementation for Adjacency Matrix)
Correctness of Greedy Algorithms
Job Sequencing Problem | Set 1 (Greedy Algorithm)
Graph Coloring | Set 2 (Greedy Algorithm)
Huffman Coding | Greedy Algo-3
Boruvka's algorithm | Greedy Algo-9
Top 20 Greedy Algorithms Interview Questions
Greedy Algorithm for Egyptian Fraction
Activity Selection Problem | Greedy Algo-1
Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
K Centers Problem | Set 1 (Greedy Approximate Algorithm)
Coin game of two corners (Greedy Approach)

Improved By : VikashDubey, Sumon Nath

Article Tags : Graph Greedy MST

Practice Tags : Greedy Graph



Be the First to upvote.

☐ To-do ☐ Done

4.4

Based on 81 vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

GeeksforGeeks
A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

[About Us](#)
[Careers](#)
[Privacy Policy](#)
[Contact Us](#)

LEARN

[Algorithms](#)
[Data Structures](#)
[Languages](#)
[CS Subjects](#)
[Video Tutorials](#)

PRACTICE

[Company-wise](#)
[Topic-wise](#)
[Contests](#)
[Subjective Questions](#)

CONTRIBUTE

[Write an Article](#)
[Write Interview Experience](#)
[Internships](#)
[Videos](#)

[@geeksforgeeks](#), Some rights reserved.