

[Home](#) [Data Structure](#) [C](#) [C++](#) [C#](#) [Java](#) [SQL](#) [HTML](#) [CSS](#) [JavaScript](#) [Ajax](#)

Prim's Algorithm

In this article, we will discuss the prim's algorithm. Along with the algorithm, we will also see the complexity, working, example, and implementation of prim's algorithm.

Before starting the main topic, we should discuss the basic and important terms such as spanning tree and minimum spanning tree.

Spanning tree - A spanning tree is the subgraph of an undirected connected graph.

Minimum Spanning tree - Minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum. The weight of the spanning tree is the sum of the weights given to the edges of the spanning tree.

Now, let's start the main topic.

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

How does the prim's algorithm work?

Prim's algorithm is a greedy algorithm that starts from one vertex and continue to add the edges with the smallest weight until the goal is reached. The steps to implement the prim's algorithm are given as follows -

- First, we have to initialize an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.

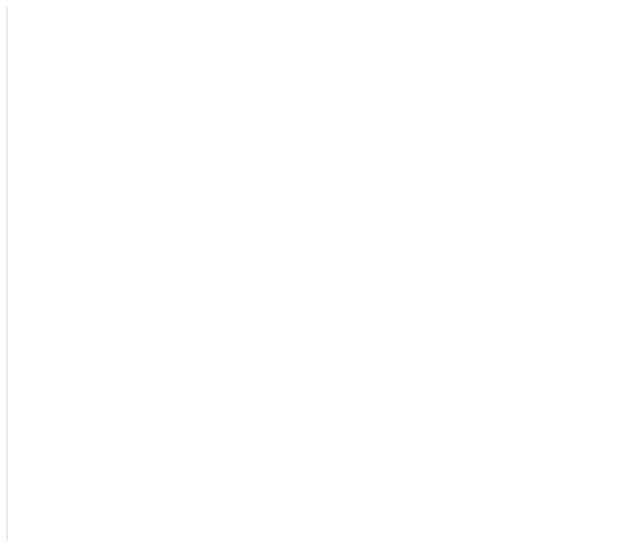
- Repeat step 2 until the minimum spanning tree is formed.

The applications of prim's algorithm are -

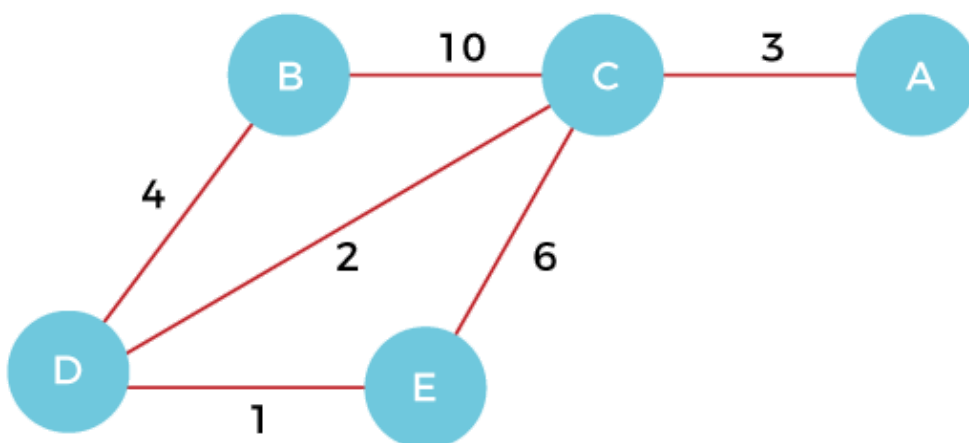
- Prim's algorithm can be used in network designing.
- It can be used to make network cycles.
- It can also be used to lay down electrical wiring cables.

Example of prim's algorithm

Now, let's see the working of prim's algorithm using an example. It will be easier to understand the prim's algorithm using an example.



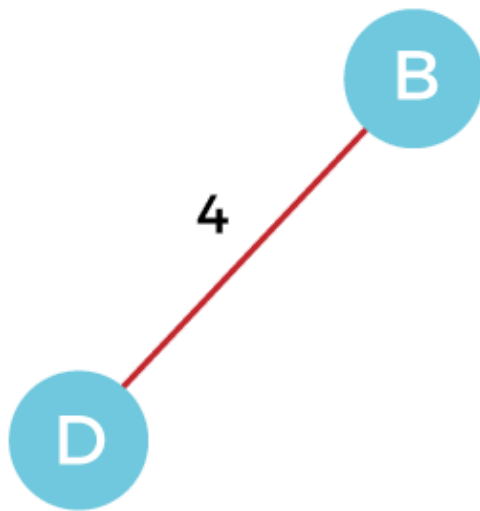
Suppose, a weighted graph is -



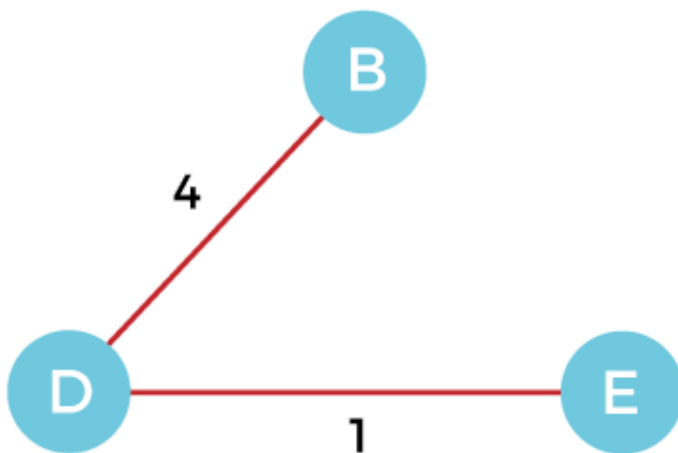
Step 1 - First, we have to choose a vertex from the above graph. Let's choose B.



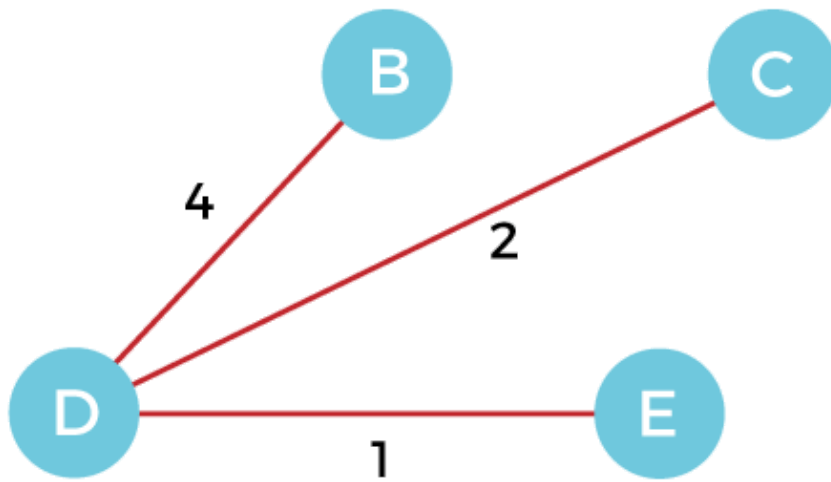
Step 2 - Now, we have to choose and add the shortest edge from vertex B. There are two edges from vertex B that are B to C with weight 10 and edge B to D with weight 4. Among the edges, the edge BD has the minimum weight. So, add it to the MST.



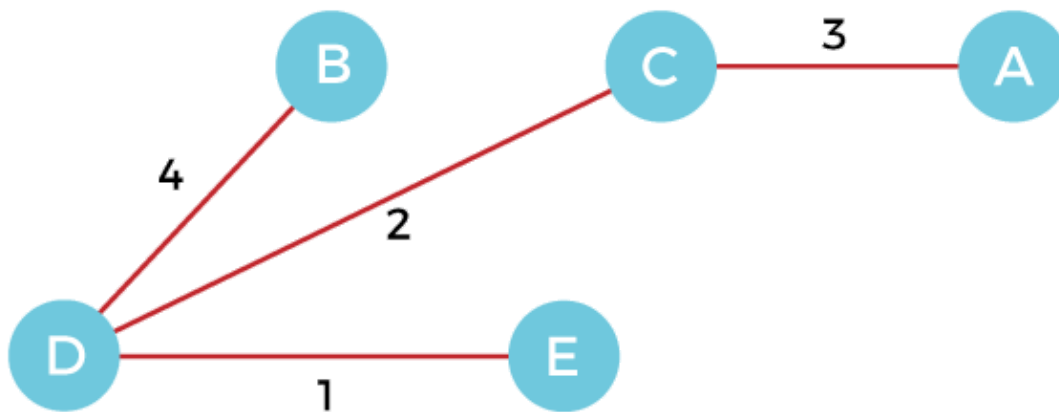
Step 3 - Now, again, choose the edge with the minimum weight among all the other edges. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C, i.e., E and A. So, select the edge DE and add it to the MST.



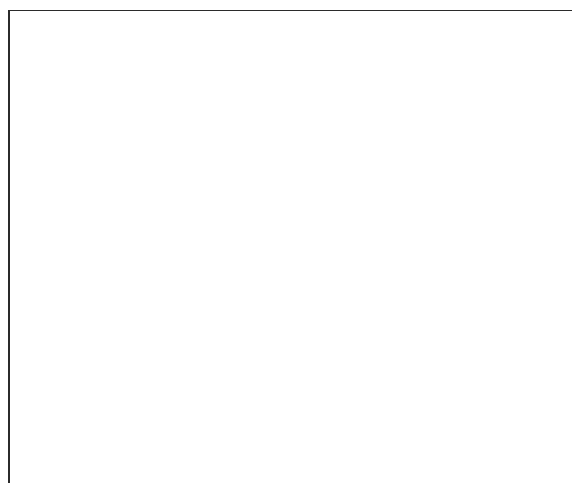
Step 4 - Now, select the edge CD, and add it to the MST.



Step 5 - Now, choose the edge CA. Here, we cannot select the edge CE as it would create a cycle to the graph. So, choose the edge CA and add it to the MST.



So, the graph produced in step 5 is the minimum spanning tree of the given graph. The cost of the MST is given below -



Cost of MST = $4 + 2 + 1 + 3 = 10$ units.

Algorithm

Step 1: Select a starting vertex

Step 2: Repeat Steps 3 and 4 until there are fringe vertices

Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight

Step 4: Add the selected edge and the vertex to the minimum spanning tree T

[END OF LOOP]

Step 5: EXIT

Complexity of Prim's algorithm

Now, let's see the time complexity of Prim's algorithm. The running time of the prim's algorithm depends upon using the data structure for the graph and the ordering of edges. Below table shows some choices -

- **Time Complexity**

Data structure used for the minimum edge weight	Time Complexity
Adjacency matrix, linear searching	$O(V ^2)$
Adjacency list and binary heap	$O(E \log V)$
Adjacency list and Fibonacci heap	$O(E + V \log V)$

Prim's algorithm can be simply implemented by using the adjacency matrix or adjacency list graph representation, and to add the edge with the minimum weight requires the linearly searching of an array of weights. It requires $O(|V|^2)$ running time. It can be improved further by using the implementation of heap to find the minimum weight edges in the inner loop of the algorithm.

The time complexity of the prim's algorithm is $O(E \log V)$ or $O(V \log V)$, where E is the no. of edges, and V is the no. of vertices.

Implementation of Prim's algorithm

Now, let's see the implementation of prim's algorithm.

Program: Write a program to implement prim's algorithm in C language.

```
#include <stdio.h>
#include <limits.h>
#define vertices 5 /*Define the number of vertices in the graph*/
/* create minimum_key() method for finding the vertex that has minimum key-
value and that is not added in MST yet */
int minimum_key(int k[], int mst[])
{
    int minimum = INT_MAX, min,i;

    /*iterate over all vertices to find the vertex with minimum key-value*/
    for (i = 0; i < vertices; i++)
        if (mst[i] == 0 && k[i] < minimum )
            minimum = k[i], min = i;
    return min;
}
/* create prim() method for constructing and printing the MST.
The g[vertices][vertices] is an adjacency matrix that defines the graph for MST.*/
void prim(int g[vertices][vertices])
{

```

```

/* create array of size equal to total number of vertices for storing the MST*/
int parent[vertices];
/* create k[vertices] array for selecting an edge having minimum weight*/
int k[vertices];
int mst[vertices];
int i, count, edge, v; /*Here 'v' is the vertex*/
for (i = 0; i < vertices; i++)
{
    k[i] = INT_MAX;
    mst[i] = 0;
}
k[0] = 0; /*It select as first vertex*/
parent[0] = -1; /* set first value of parent[] array to -1 to make it root of MST*/
for (count = 0; count < vertices-1; count++)
{
    /*select the vertex having minimum key and that is not added in the MST yet from the set of v
    edge = minimum_key(k, mst);
    mst[edge] = 1;
    for (v = 0; v < vertices; v++)
    {
        if (g[edge][v] && mst[v] == 0 && g[edge][v] < k[v])
        {
            parent[v] = edge, k[v] = g[edge][v];
        }
    }
}
/*Print the constructed Minimum spanning tree*/
printf("\n Edge \t Weight\n");
for (i = 1; i < vertices; i++)
printf(" %d <-> %d   %d \n", parent[i], i, g[i][parent[i]]);

}

int main()
{
    int g[vertices][vertices] = {{0, 0, 3, 0, 0},
                                   {0, 0, 10, 4, 0},
                                   {3, 10, 0, 2, 6},
                                   {0, 4, 2, 0, 1},
                                   {0, 0, 0, 0, 0}};
}

```



```
        {0, 0, 6, 1, 0},  
        };  
  
    prim(g);  
    return 0;  
}
```

Output

```
Edge      Weight  
3 <-> 1    4  
0 <-> 2    3  
2 <-> 3    2  
3 <-> 4    1
```

So, that's all about the article. Hope, the article will be helpful and informative to you.

[< Prev](#)[Next >](#)

 **For Videos Join Our Youtube Channel: [Join Now](#)**

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share




Learn Latest Tutorials

 Splunk tutorial

Splunk

 SPSS tutorial

SPSS

 Swagger
tutorial

Swagger

 T-SQL tutorial

Transact-SQL

 Tumblr tutorial


Tumblr

 React tutorial

ReactJS

 Regex tutorial

Regex

 Reinforcement
learning tutorialReinforcement
Learning R Programming
tutorial

R Programming

 RxJS tutorial

RxJS

 React Native
tutorial

React Native

 Python Design
PatternsPython Design
Patterns Python Pillow
tutorial

Python Pillow

 Python Turtle
tutorial

Python Turtle

 Keras tutorial

Keras

Preparation

 Aptitude

Aptitude

 Logical
Reasoning

Reasoning

 Verbal Ability

Verbal Ability













 Interview
Questions

Interview Questions





 Company
Interview
Questions

Company Questions

Trending Technologies

 Artificial Intelligence Artificial Intelligence	 AWS Tutorial AWS	 Selenium tutorial Selenium	 Cloud Computing Cloud Computing
 Hadoop tutorial Hadoop	 ReactJS Tutorial ReactJS	 Data Science Tutorial Data Science	 Angular 7 Tutorial Angular 7
 Blockchain Tutorial Blockchain	 Git Tutorial Git	 Machine Learning Tutorial Machine Learning	 DevOps Tutorial DevOps

B.Tech / MCA

 DBMS tutorial DBMS	 Data Structures tutorial Data Structures	 DAA tutorial DAA	 Operating System Operating System
 Computer Network tutorial Computer Network	 Compiler Design tutorial Compiler Design	 Computer Organization and Architecture Computer Organization	 Discrete Mathematics Tutorial Discrete Mathematics
 Ethical Hacking Ethical Hacking	 Computer Graphics Tutorial Computer Graphics	 Software Engineering Software Engineering	 html tutorial Web Technology
 Cyber Security tutorial Cyber Security	 Automata Tutorial Automata	 C Language tutorial C Programming	 C++ tutorial C++



Java tutorial

Java



.Net
Framework
tutorial

.Net



Python tutorial

Python



List of
Programs
Programs



Control
Systems tutorial

Control System



Data Mining
Tutorial

Data Mining



Data
Warehouse
Tutorial

Data Warehouse