



Aug 2025, India

Prim's Algorithm

Tutorial

Tutorial By

Vikas Awadhiya

This work is licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/)  

Prim's Algorithm Tutorial © 2025 by Vikas Awadhiya is licensed under
Creative Commons Attribution 4.0 International (CC-BY 4.0).
To view a copy of this license, visit

<https://creativecommons.org/licenses/by/4.0/>



Invented By

Vojtěch Jarník

Invented By

Robert C. Prim

Tutorial By

Vikas Awadhiya

LinkedIn profile: <https://in.linkedin.com/in/awadhiya-vikas>

1 Introduction

The Prim's algorithm finds minimum spanning tree (MST) of an undirected weighted graph. It can find minimum spanning tree of a simple/multi graph, connected/disconnected graph. For a disconnected undirected graph a separate run is required for each component as compared to Kruskal's algorithm, which find minimum spanning tree of all components in a single run.

Algorithm has $O(E \log V)$ time complexity and $O(E + V)$ space complexity, assuming Binary-Heap (priority queue) data structure is used to find an edge with smallest possible weight in each step. Here E is number of edges $|E|$ and V is number of vertices $|V|$.

prim's algorithm is more efficient for a dense graph as compared to Kruskal's algorithm because Prim's algorithm doesn't have to iterate over all the edges like Kruskal's algorithm. Kruskal's algorithm performs well with a sparse graph.

Prim's algorithm belongs to a special category of algorithms called greedy algorithms.

2 The Algorithm

Prims's algorithm finds the minimum spanning tree of an undirected weighted graph $G = (V, E, \omega)$, where V is finite set of vertices, E is finite set of edges and ω is a weight function where $\omega : E \rightarrow \mathbb{R}$ or simply $\forall e \in E, \omega(e) = \mathbb{R}$ (here ω is a Greek alphabet omega in lower case, \rightarrow means "map to" / "is a function from", \mathbb{R} is real numbers, \forall is read as "for all"/ "for every", \in is read as "element of"/"belongs to" and e represents an edge).

As a greedy algorithm it makes decision best/optimal at a step level rather than best/optimal decision at a problem level. To understand the algorithm it is first required to understand what is a minimum spanning tree and that also requires to understand few other concepts of graph-theory like tree, cut-edge and spanning tree.

Tree is an acyclic graph which doesn't contain cycle (or self loop) and every edge of the graph is a cut-edge as shown in fig 1 below on the left. So removing any edge disconnects the graph and creates two components. The cut-edge also called bridge or bond. The bond is minimum number of edges required to remove, to divide the graph into two components. In a tree, bond is only one edge and removing a single edge creates two components as shown below in fig 2 and in fig 3.

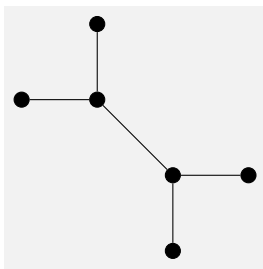


Fig 1

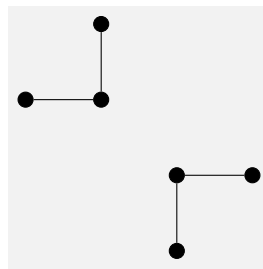


Fig 2

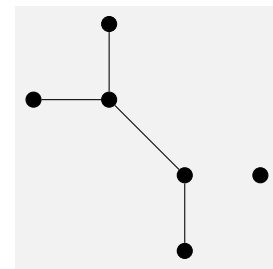


Fig 3

In other words, tree is a graph in which, there is only a single path between any two vertices. A graph shown in fig 4 below on the left is not a tree because it has multiple paths between the vertices for example multiple paths between vertex a and vertex d are highlighted by bold lines in fig 5 and fig 6 below.

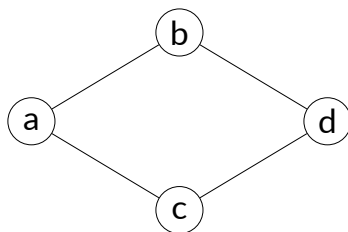


Fig 4

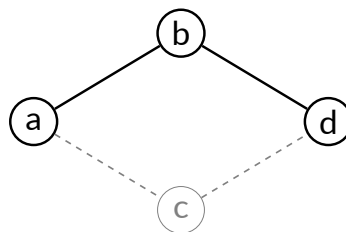


Fig 5

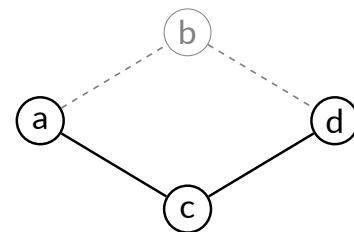


Fig 6

So the graph shown in fig 4 is not a tree and due to this removing an edge can't divide graph in two components.

The spanning tree is a tree which spans across all vertices of a graph. The tree shown in fig 8 below on the middle is a spanning tree of the graph shown below on the left in fig 7 compared to tree shown below on the right in fig 9 which is not a spanning tree because it doesn't span across all the vertices of the graph and the one vertex remains isolated vertex.

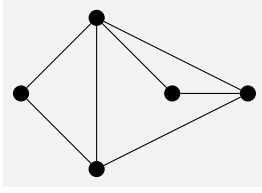


Fig 7

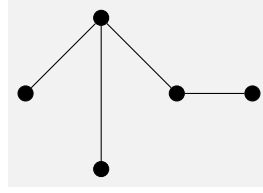


Fig 8

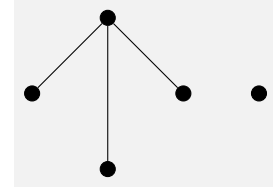


Fig 9

There is more than one spanning tree possible in a graph and number of spanning trees of a graph G is denoted by $\tau(G)$ (here τ is Greek alphabet tau in lower-case). Number of spanning trees of a complete graph K_n can be evaluated by $\tau(K_n) = n^{n-2}$, for example a triangle as a complete graph with three vertices and edges of weights 1, 1 and 2 has $\tau(K_3) = 3^{3-2} = 3$ number of spanning trees as shown below in fig 10, fig 11 and in fig 12.

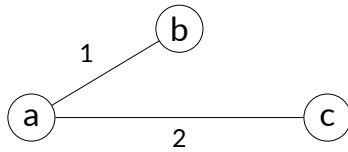


Fig 10

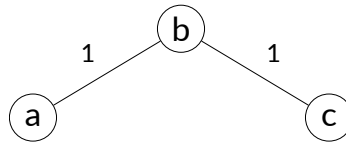


Fig 11

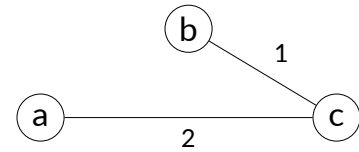


Fig 12

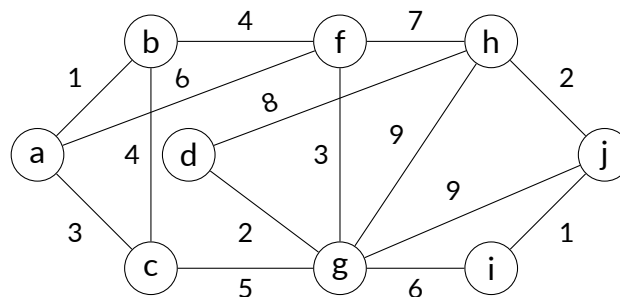
Among these spanning trees the tree shown in fig 11 above on the middle is minimum spanning tree because sum of weights of it's edges is minimum compare to sum of weights of edges of other two spanning trees. Tutorials explained minimum spanning tree and concepts related to it, now it proceeds to explain algorithm.

2.1 Algorithm's Steps

Prim's algorithm is viewed as the following steps,

1. Randomly select a vertex,
2. Select an edge with minimum weight among the edges incident to selected vertices, if the vertex at the other end of the edge is not yet selected.

An example is required to explain these steps, so let consider an undirected weighted graph G as shown below in fig 13.

Fig 13 : Graph G

This graph G is a simple undirected weighted graph. A simple graph is a graph which doesn't have a vertex with self-loop or contain parallel edges between any two vertices. That's why in a simple graph a pair of vertices $\{u, v\}$ uniquely represents an edge between any vertex u and vertex v and $\omega(uv)$ represents edge's weight. Vertices are usually labeled with numbers in a graph but here English alphabets are used to distinguish vertices from weights of edges.

The initial state where no vertex is selected yet, is shown below on the left in fig 14. The first step of the algorithm is to select a vertex randomly so let's consider vertex a is selected as shown below on the right in fig 15.

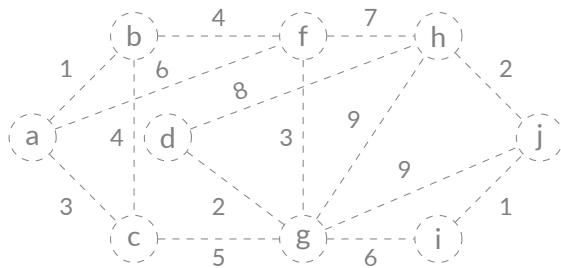


Fig 14

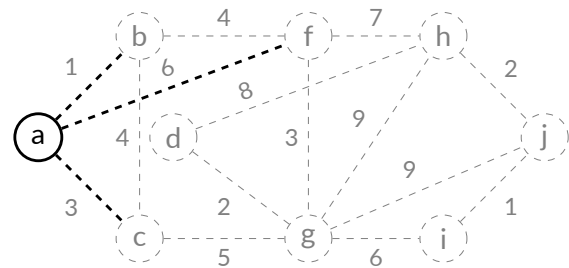


Fig 15

The vertex a is selected and the edges $\{a, b\}$, $\{a, c\}$ and edge $\{a, f\}$ are incident to it. In next step algorithm select an edge with minimum weighted among these incident edges. Algorithm selects an edge among these edges because the vertices at the other end of these edges are not yet selected or in other words not marked as selected.

In tutorial it is visible above in fig 15 but in implement algorithm maintains select status of all vertices and updates the status on the selection of a vertex and also inserts edges incident to this newly selected vertex in min-heap if the vertices at the other end of these edges are not marked as selected. The min-heap data structure provides an edge with minimum weight among these incident edges in each step.

The figures represent select status of a vertex as marked by bold circle node, selected edge by bold line and bold dashed lines are used to represents edges incident to selected vertices and represents potential candidates of next edge selection.

As shown above in fig 15 among the edges $\{a, b\}$, $\{a, f\}$ and edge $\{a, c\}$ the weight of edge $\{a, b\}$ is minimum and algorithm selects it as shown below on the left in fig 16.

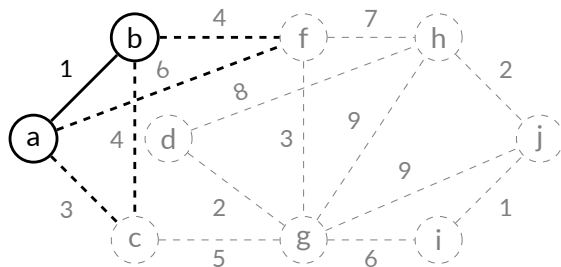


Fig 16

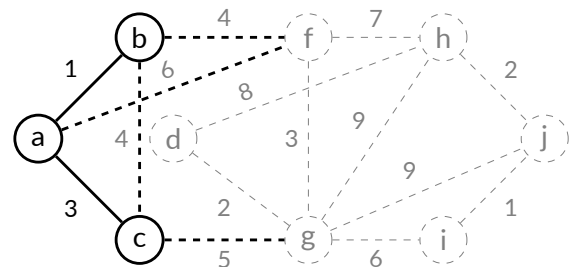


Fig 17

Because edge $\{a, b\}$ is selected the vertex b at the other end of it, is marked as selected and edges $\{b, f\}$ and edge $\{b, c\}$ are now also among the potential candidates of next edge selection as shown in fig 16 above on the left.

Now algorithm selects edge $\{a, c\}$ of minimum weight among all other incident edges and mark vertex c as selected. The edge $\{c, g\}$ incident to vertex c is added as potential candidate of next edge selection (here the other incident edge $\{b, c\}$ is already a known edge) as show in fig 17 above on the right.

There are two edges, edge $\{b, c\}$ and edge $\{b, f\}$ with smallest weights and let's consider edge $\{b, c\}$ is processed first (top entry of priority queue) but the vertices at both side of it are already

selected and due to this algorithm discard it. Algorithm next selects and edge $\{b, f\}$ with smallest weight because vertex f at other side of it, is an unselected vertex then algorithm mark vertex f as selected. The edge $\{f, g\}$ and edge $\{f, h\}$ are now also among the incident edges algorithm considers for next edge selection as shown below on the left in fig 18.

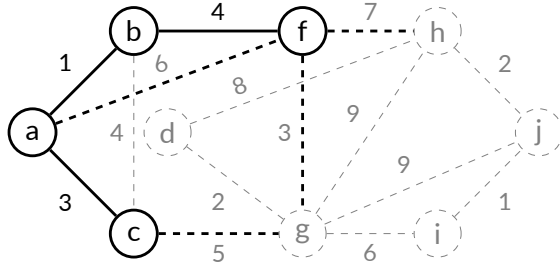


Fig 18

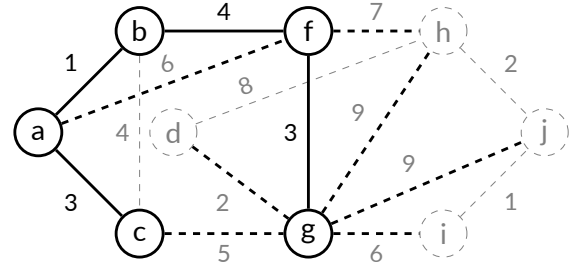


Fig 19

As algorithm proceeds, it has four incident edges, edge $\{a, f\}$, edge $\{c, g\}$, edge $\{f, g\}$ and edge $\{f, h\}$ to consider as shown in fig 18 above. It selects edge $\{f, g\}$ with smallest weight compared to other three edges and mark vertex g as selected. The edges incident to vertex g are now also the potential candidates for next edge selection as show above on the right in fig 19.

The edge $\{d, g\}$ with minimum weight is selected. Algorithm marks vertex d as selected and consider edge $\{d, h\}$ among other incident edges for next edge selection as shown below on the left in fig 20.

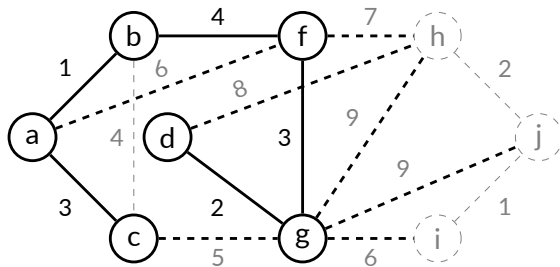


Fig 20

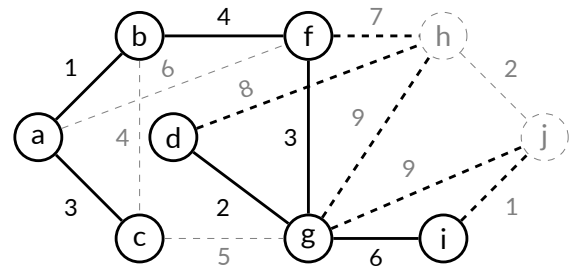


Fig 21

Now edge $\{c, g\}$ is an edge with minimum weight among incident edges but algorithm reject it because vertices at both ends of it are already selected. Then edge $\{a, f\}$ and edge $\{g, i\}$ have minimum weighted and let's consider algorithm process edge $\{a, f\}$ first but also rejects it. Finally algorithm selects edge $\{g, i\}$, marks vertex i as selected and also considers edge $\{i, j\}$ among other incident edges for next edge selection as shown above on the right in fig 21.

Next edge $\{i, j\}$ with minimum weight among incident edges is selected as shown below in fig 22.

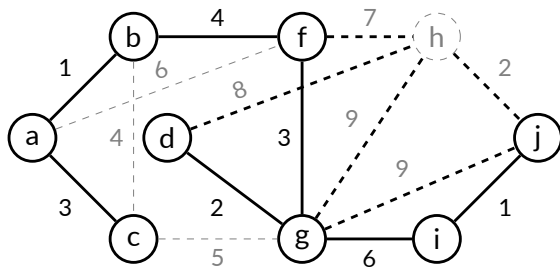


Fig 22

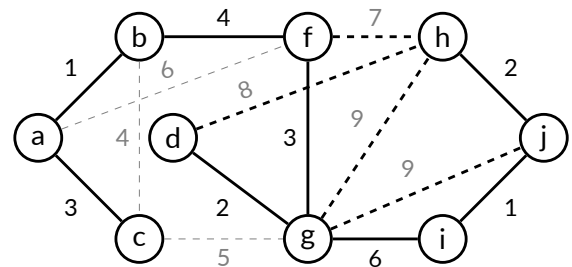


Fig 23

Then algorithm marks vertex j as selected and also considers edge $\{h, j\}$ incident to vertex j among the other incident edges for next edge selection as shown above on the left in fig 22.

Now edge $\{h, j\}$ has minimum weight compared to other incident edge $\{d, h\}$, edge $\{g, h\}$ and edge $\{g, j\}$ as shown above on the left in fig 22. Algorithm proceeds and selects edge $\{h, j\}$ and marks the vertex h selected as shown above on the right in fig 23.

Algorithm continues and process the remaining incident vertices but no edge can be selected further because minimum spanning tree of graph G is already completed where all vertices are connected and no vertex remain isolated. So the spanning tree of graph G is shown belong in fig 24.

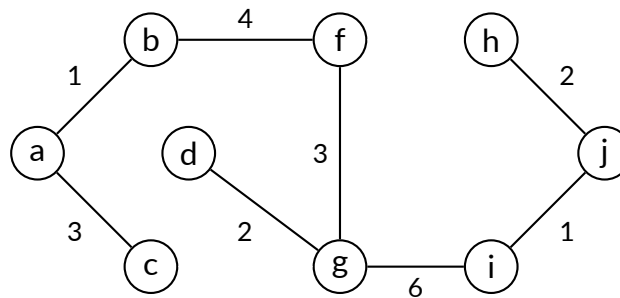


Fig 24 : Spanning Tree

The Prim's algorithm doesn't process all edges rather stops once all V vertices are selected. It may process more than $V - 1$ edges but even with the dense graph like complete graph to find minimum spanning tree all edges are not required to process.

3 Complexity Analysis

Min-Heap/Priority-Queue data structure requires $O(\log n)$ time complexity for insert(push) and remove(pop) operations. The undirected complete graph K_n helps to evaluate time complexity of Prim's algorithm. A complete graph K_n is a graph where each vertex is neighbor of every other vertex or in other words each vertex is directly connected to every other vertex and each vertex has degree $d(V - 1)$ (number of incident edges).

Algorithm begins by randomly selecting a vertex in $O(1)$ constant time and populate min-heap with all the edges incident to the vertex in $O(V)$ complexity. The complexity of initial step is linear non dominating compare to the overall complexity the next step defines. Now min-heap/priority-queue has $V - 1$ edges and an insert or a remove operation requires $O(\log V)$ time complexity. Big O complexity analysis represents worst scenario of asymptotic analysis and ignores the constant. So the insert and remove operations require $O(\log V)$ time complexity rather than $O(\log(V - 1))$ time complexity.

Algorithm at each step remove top element (edge) of min-heap and insert $V - 1$ edges in min-heap incident to the unselected vertex at other end of the selected edge. To remove the element takes $O(\log V)$ time and to insert $V - 1$ edges takes $O(V \log V)$ time, So the complexity of a step is,

$$\begin{aligned} \text{Time Complexity of a step} &= \text{complexity of top edge removal} + \text{complexity of insertion of } V - 1 \text{ edges} \\ &= O(\log V) + O(V \log V) \end{aligned}$$

To find minimum spanning tree this step is performed $V - 1$ times and the overall time complexity of Prim's algorithm is $O(V)$ multiply to the complexity of a single step,

$$\begin{aligned} \text{Time Complexity} &= V \times \text{Time complexity of a step} \\ &= O(V \log V) + O(V^2 \log V) \\ &= O((V + V^2) \log V) \text{ but } V^2 \text{ is dominating and big } O \text{ ignores } V \\ &= O(V^2 \log V) \end{aligned}$$

$$\begin{aligned} \text{But in a complete undirected graph, edges } E &= V(V - 1) * \frac{1}{2} = O(V^2), \text{ constant ignored} \\ &= O(E \log V) \end{aligned}$$

In real world scenario few edges are rejected and only removal of top element of min-heap is performed (the half step). This increase the number of removal required to select $V - 1$ edges by some constant factor but it stay remains below V^2 and doesn't affect the complexity.

The incident edges are inserted only $V - 1$ times because a vertex is selected only once. The min-heap may contains $V^2 = E$ number of edges at a given point because the first step insert $V - 1$ number of incident edges but as algorithm proceeds number of selected vertices increase and as a result the number of edges inserted at each step decreases. That's why the second last step insert only 1 edge and max possible edges in a min-heap $(V - 1) + \dots + 1 = V^2 = E$.

But due to power property of log function $\log_x(y^n) = n \log_x y$ complexity of insert operation of min-heap remains same to $O(\log(V^2)) = O(2 \log V) = O(\log V)$.

The $O(E)$ space is required to store $|E|$ number of edges in min-heap and $O(V)$ to maintain select status of $|V|$ vertices. So the overall space complexity of Prim's algorithm is $O(E + V)$. In dense graph K_n where $E = V(V - 1)$ and space complexity is $O(V^2 + V) = O(V^2)$ but the $O(E + V)$ form is always used and it is applicable to all types of graphs.

4 Directed Graph Constraint

It is obvious question why Prim's algorithm can't be used with directed weighted graph to find something similar to a minimum spanning tree?

Analogue of a minimum spanning tree in a directed weighted graph is called minimum spanning arborescence (MSA). It is fundamentally differ from a minimum spanning tree.

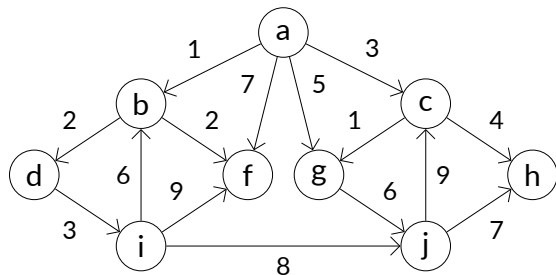


Fig 25

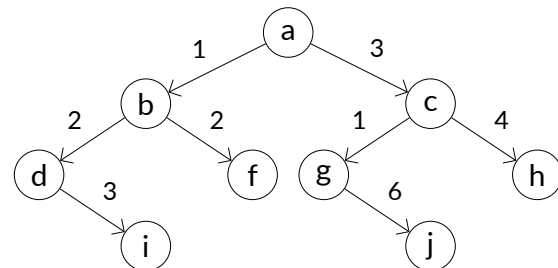


Fig 26

Arborescence (spanning arborescence) is an acyclic directed graph in which there is an unique path from a vertex (root vertex) to every other vertex of a directed graph. It is also called a directed rooted tree. The minimum spanning arborescence is a spanning arborescence with minimum sum of weight of edges. A directed graph shown above on the left in fig 25 and it's minimum spanning arborescence is shown above on the right in fig 26.

The Prim's algorithm doesn't work with a directed graph because min-heap/priority-queue data structure arrange edges according to their weights and provide an edge at each step with minimum possible weight but this greedy approach only care about the minimum weight and can't select the edges by considering the direction as well.

An arborescence (spanning arborescence) is not always possible for directed graphs and it may happen that a directed connected weighted graph doesn't have a vertex such that there is a unique path from it to every other vertex. This makes finding of a minimum spanning arborescence more complex. The directed connected graph shown below in fig 27, there is no arborescence possible for it.

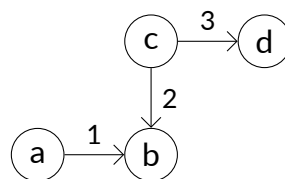


Fig 27

So if there is no arborescence possible for a directed connected graph then how Prim's algorithm can find minimum spanning arborescence? The Prim's algorithm works with an undirected connected graph because an undirected connected graph always has at least one spanning tree due to undirected edges. Undirected edges are used in both direction and makes every vertex reachable from every other vertex.

5 Implementation

The C++ implementation of the Prim's algorithm is provided under the MIT License. It also contains the `main.cpp` file to demonstrate its usage.

Code available at: <https://github.com/vikasawadhiya/Prim-Algorithm>

August 2025, India