



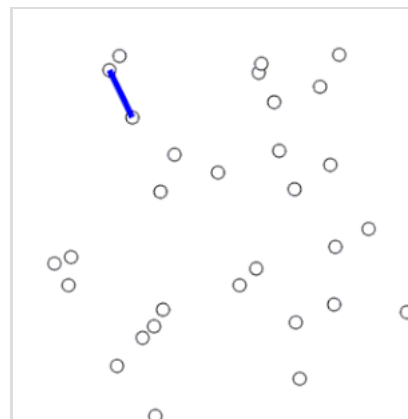
WIKIPEDIA  
The Free Encyclopedia

WIKIPEDIA

# Prim's algorithm

In computer science, **Prim's algorithm** is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

The algorithm was developed in 1930 by Czech mathematician Vojtěch Jarník<sup>[1]</sup> and later rediscovered and republished by computer scientists Robert C. Prim in 1957<sup>[2]</sup> and Edsger W. Dijkstra in 1959.<sup>[3]</sup> Therefore, it is also sometimes called the **Jarník's algorithm**,<sup>[4]</sup> **Prim–Jarník algorithm**,<sup>[5]</sup> **Prim–Dijkstra algorithm**<sup>[6]</sup> or the **DJP algorithm**.<sup>[7]</sup>



A demo for Prim's algorithm based on Euclidean distance

Other well-known algorithms for this problem include Kruskal's algorithm and Borůvka's algorithm.<sup>[8]</sup> These algorithms find the minimum spanning forest in a possibly disconnected graph; in contrast, the most basic form of Prim's algorithm only finds minimum spanning trees in connected graphs. However, running Prim's algorithm separately for each connected component of the graph, it can also be used to find the minimum spanning forest.<sup>[9]</sup> In terms of their asymptotic time complexity, these three algorithms are equally fast for sparse graphs, but slower than other more sophisticated algorithms.<sup>[7][6]</sup> However, for graphs that are sufficiently dense, Prim's algorithm can be made to run in linear time, meeting or improving the time bounds for other algorithms.<sup>[10]</sup>

## Description

The algorithm may informally be described as performing the following steps:

1. Initialize a tree with a single vertex, chosen arbitrarily from the graph.
2. Grow the tree by one edge: Of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat step 2 (until all vertices are in the tree).

In more detail, it may be implemented following the pseudocode below.

1. Associate with each vertex  $v$  of the graph a number  $C[v]$  (the cheapest cost of a connection to  $v$ ) and an edge  $E[v]$  (the edge providing that cheapest connection). To initialize these values, set all values of  $C[v]$  to  $+\infty$  (or to any number larger than the maximum edge weight) and set each  $E[v]$  to a special flag value indicating that there is no edge connecting  $v$  to earlier vertices.

2. Initialize an empty forest  $F$  and a set  $Q$  of vertices that have not yet been included in  $F$  (initially, all vertices).
3. Repeat the following steps until  $Q$  is empty:
  - a. Find and remove a vertex  $v$  from  $Q$  having the minimum possible value of  $C[v]$
  - b. Add  $v$  to  $F$
  - c. Loop over the edges  $vw$  connecting  $v$  to other vertices  $w$ . For each such edge, if  $w$  still belongs to  $Q$  and  $vw$  has smaller weight than  $C[w]$ , perform the following steps:
    - i. Set  $C[w]$  to the cost of edge  $vw$
    - ii. Set  $E[w]$  to point to edge  $vw$ .
4. Return  $F$ , which specifically includes the corresponding edges in  $E$

As described above, the starting vertex for the algorithm will be chosen arbitrarily, because the first iteration of the main loop of the algorithm will have a set of vertices in  $Q$  that all have equal weights, and the algorithm will automatically start a new tree in  $F$  when it completes a spanning tree of each connected component of the input graph. The algorithm may be modified to start with any particular vertex  $s$  by setting  $C[s]$  to be a number smaller than the other values of  $C$  (for instance, zero), and it may be modified to only find a single spanning tree rather than an entire spanning forest (matching more closely the informal description) by stopping whenever it encounters another vertex flagged as having no associated edge.

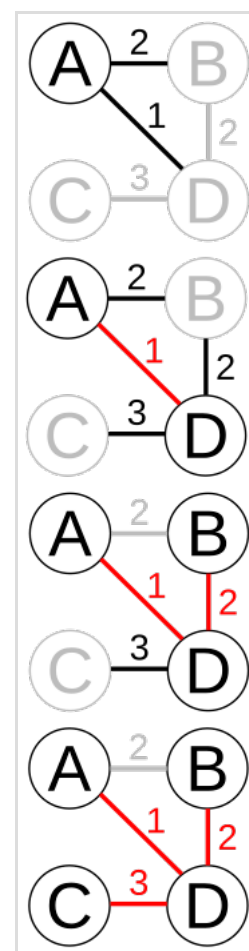
Different variations of the algorithm differ from each other in how the set  $Q$  is implemented: as a simple linked list or array of vertices, or as a more complicated priority queue data structure. This choice leads to differences in the time complexity of the algorithm. In general, a priority queue will be quicker at finding the vertex  $v$  with minimum cost, but will entail more expensive updates when the value of  $C[w]$  changes.

## Time complexity

The time complexity of Prim's algorithm depends on the data structures used for the graph and for ordering the edges by weight, which can be done using a priority queue. The following table shows the typical choices:

Minimum edge weight data structure	Time complexity (total)
<u>adjacency matrix</u> , searching	$O( V ^2)$
<u>binary heap</u> and <u>adjacency list</u>	$O(( V  +  E ) \log  V ) = O( E  \log  V )$
<u>Fibonacci heap</u> and <u>adjacency list</u>	$O( E  +  V  \log  V )$

A simple implementation of Prim's, using an adjacency matrix or an adjacency list graph representation



Prim's algorithm starting at vertex A. In the third step, edges BD and AB both have weight 2, so BD is chosen arbitrarily. After that step, AB is no longer a candidate for addition to the tree because it links two nodes that are already in the tree.

and linearly searching an array of weights to find the minimum weight edge to add, requires  $O(|V|^2)$  running time. However, this running time can be greatly improved by using heaps to implement finding minimum weight edges in the algorithm's inner loop.

A first improved version uses a heap to store all edges of the input graph, ordered by their weight. This leads to an  $O(|E| \log |E|)$  worst-case running time. But storing vertices instead of edges can improve it still further. The heap should order the vertices by the smallest edge-weight that connects them to any vertex in the partially constructed minimum spanning tree (MST) (or infinity if no such edge exists). Every time a vertex  $v$  is chosen and added to the MST, a decrease-key operation is performed on all vertices  $w$  outside the partial MST such that  $v$  is connected to  $w$ , setting the key to the minimum of its previous value and the edge cost of  $(v,w)$ .

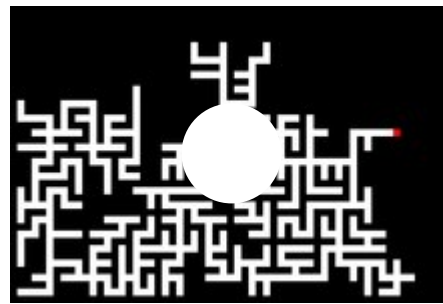
Using a simple binary heap data structure, Prim's algorithm can now be shown to run in time  $O(|E| \log |V|)$  where  $|E|$  is the number of edges and  $|V|$  is the number of vertices. Using a more sophisticated Fibonacci heap, this can be brought down to  $O(|E| + |V| \log |V|)$ , which is asymptotically faster when the graph is dense enough that  $|E|$  is  $\omega(|V|)$ , and linear time when  $|E|$  is at least  $|V| \log |V|$ . For graphs of even greater density (having at least  $|V|^c$  edges for some  $c > 1$ ), Prim's algorithm can be made to run in linear time even more simply, by using a  $d$ -ary heap in place of a Fibonacci heap.<sup>[10][11]</sup>

## Proof of correctness

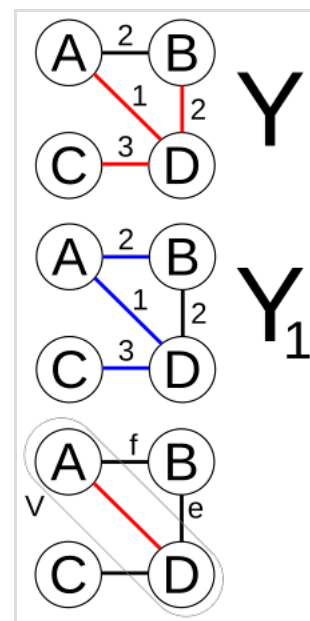
Let  $P$  be a connected, weighted graph. At every iteration of Prim's algorithm, an edge must be found that connects a vertex in a subgraph to a vertex outside the subgraph. Since  $P$  is connected, there will always be a path to every vertex. The output  $Y$  of Prim's algorithm is a tree, because the edge and vertex added to tree  $Y$  are connected. Let  $Y_I$  be a minimum spanning tree of graph  $P$ . If  $Y_I = Y$  then  $Y$  is a minimum spanning tree. Otherwise, let  $e$  be the first edge added during the construction of tree  $Y$  that is not in tree  $Y_I$ , and  $V$  be the set of vertices connected by the edges added before edge  $e$ . Then one endpoint of edge  $e$  is in set  $V$  and the other is not. Since tree  $Y_I$  is a spanning tree of graph  $P$ , there is a path in tree  $Y_I$  joining the two endpoints. As one travels along the path, one must encounter an edge  $f$  joining a vertex in set  $V$  to one that is not in set  $V$ . Now, at the iteration when edge  $e$  was added to tree  $Y$ , edge  $f$  could also have been added and it would be added instead of edge  $e$  if its weight was less than  $e$ , and since edge  $f$  was not added, we conclude that

$$w(f) \geq w(e).$$

Let tree  $Y_2$  be the graph obtained by removing edge  $f$  from and adding edge  $e$  to tree  $Y_I$ . It is easy to show that tree  $Y_2$  is connected, has the same number of edges as tree  $Y_I$ , and the total weights of its edges is not larger than that of tree  $Y_I$ , therefore it is also a



Prim's algorithm has many applications, such as in the generation of this maze, which applies Prim's algorithm to a randomly weighted grid graph.



Demonstration of proof. In this case, the graph  $Y_1 = Y - f + e$  is already equal to  $Y$ . In general, the process may need to be repeated.

minimum spanning tree of graph  $P$  and it contains edge  $e$  and all the edges added before it during the construction of set  $V$ . Repeat the steps above and we will eventually obtain a minimum spanning tree of graph  $P$  that is identical to tree  $Y$ . This shows  $Y$  is a minimum spanning tree. The minimum spanning tree allows for the first subset of the sub-region to be expanded into a larger subset  $X$ , which we assume to be the minimum.

## Parallel algorithm

The main loop of Prim's algorithm is inherently sequential and thus not parallelizable. However, the inner loop, which determines the next edge of minimum weight that does not form a cycle, can be parallelized by dividing the vertices and edges between the available processors.<sup>[12]</sup> The following pseudocode demonstrates this.

1. Assign each processor  $P_i$  a set  $V_i$  of consecutive vertices of length  $\frac{|V|}{|P|}$ .
2. Create  $C$ ,  $E$ ,  $F$ , and  $Q$  as in the sequential algorithm and divide  $C$ ,  $E$ , as well as the graph between all processors such that each processor holds the incoming edges to its set of vertices. Let  $C_i$ ,  $E_i$  denote the parts of  $C$ ,  $E$  stored on processor  $P_i$ .
3. Repeat the following steps until  $Q$  is empty:
  - a. On every processor: find the vertex  $v_i$  having the minimum value in  $C_i[v_i]$  (local solution).
  - b. Min-reduce the local solutions to find the vertex  $v$  having the minimum possible value of  $C[v]$  (global solution).
  - c. Broadcast the selected node to every processor.
  - d. Add  $v$  to  $F$  and, if  $E[v]$  is not the special flag value, also add  $E[v]$  to  $F$ .
  - e. On every processor: update  $C_i$  and  $E_i$  as in the sequential algorithm.
4. Return  $F$

This algorithm can generally be implemented on distributed machines<sup>[12]</sup> as well as on shared memory machines.<sup>[13]</sup> The running time is  $O(\frac{|V|^2}{|P|}) + O(|V| \log |P|)$ , assuming that the *reduce* and *broadcast* operations can be performed in  $O(\log |P|)$ .<sup>[12]</sup> A variant of Prim's algorithm for shared memory machines, in which Prim's sequential algorithm is being run in parallel, starting from different vertices, has also been explored.<sup>[14]</sup> It should, however, be noted that more sophisticated algorithms exist to solve the distributed minimum spanning tree problem in a more efficient manner.

	$P_0$	$P_1$	...	$P_{p-1}$
	$v_0 - v_{k-1}$	$v_k - v_{2k-1}$	...	$v_{(p-1)k} - v_{n-1}$
$v_0$				
$v_1$				
$v_2$				
$v_3$				
$v_4$				
$v_5$				
$v_6$				
$v_7$				
$v_8$				
$v_9$				
$v_{10}$				
...				
$v_{n-4}$				
$v_{n-3}$				
$v_{n-2}$				
$v_{n-1}$				

The adjacency matrix distributed between multiple processors for parallel Prim's algorithm. In each iteration of the algorithm, every processor updates its part of  $C$  by inspecting the row of the newly inserted vertex in its set of columns in the adjacency matrix. The results are then collected and the next vertex to include in the MST is selected globally.

## See also

- [Dijkstra's algorithm](#), a very similar algorithm for the [shortest path problem](#)
- [Greedoids](#) offer a general way to understand the correctness of Prim's algorithm

## References


---

1. Jarník, V. (1930), "O jistém problému minimálním" [About a certain minimal problem], *Práce Moravské Přírodovědecké Společnosti* (in Czech), **6** (4): 57–63, hdl:10338.dmlcz/500726 (<https://hdl.handle.net/10338.dmlcz%2F500726>).
2. Prim, R. C. (November 1957), "Shortest connection networks And some generalizations" (<https://archive.org/details/bstj36-6-1389>), *Bell System Technical Journal*, **36** (6): 1389–1401, Bibcode:1957BSTJ...36.1389P (<https://ui.adsabs.harvard.edu/abs/1957BSTJ...36.1389P>), doi:10.1002/j.1538-7305.1957.tb01515.x (<https://doi.org/10.1002%2Fj.1538-7305.1957.tb01515.x>).
3. Dijkstra, E. W. (December 1959), "A note on two problems in connexion with graphs" (<https://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf>) (PDF), *Numerische Mathematik*, **1** (1): 269–271, CiteSeerX 10.1.1.165.7577 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.7577>), doi:10.1007/BF01386390 (<https://doi.org/10.1007%2FBF01386390>), S2CID 123284777 (<https://api.semanticscholar.org/CorpusID:123284777>).
4. Sedgewick, Robert; Wayne, Kevin Daniel (2011), *Algorithms* (<https://books.google.com/books?id=MTpsAQAAQBAJ&pg=PA628>) (4th ed.), Addison-Wesley, p. 628, ISBN 978-0-321-57351-3.
5. Rosen, Kenneth (2011), *Discrete Mathematics and Its Applications* (<https://books.google.com/books?id=6EJOCAAQBAJ&pg=PA798>) (7th ed.), McGraw-Hill Science, p. 798.
6. Cheriton, David; Tarjan, Robert Endre (1976), "Finding minimum spanning trees", *SIAM Journal on Computing*, **5** (4): 724–742, doi:10.1137/0205051 (<https://doi.org/10.1137%2F0205051>), MR 0446458 (<https://mathscinet.ams.org/mathscinet-getitem?mr=0446458>).
7. Pettie, Seth; Ramachandran, Vijaya (January 2002), "An optimal minimum spanning tree algorithm" (<http://www.cs.utexas.edu/~vlr/papers/optmsf-jacm.pdf>) (PDF), *Journal of the ACM*, **49** (1): 16–34, CiteSeerX 10.1.1.110.7670 (<https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.7670>), doi:10.1145/505241.505243 (<https://doi.org/10.1145%2F505241.505243>), MR 2148431 (<https://mathscinet.ams.org/mathscinet-getitem?mr=2148431>), S2CID 5362916 (<https://api.semanticscholar.org/CorpusID:5362916>).
8. Tarjan, Robert Endre (1983), "Chapter 6. Minimum spanning trees. 6.2. Three classical algorithms", *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 44, Society for Industrial and Applied Mathematics, pp. 72–77.
9. Kepner, Jeremy; Gilbert, John (2011), *Graph Algorithms in the Language of Linear Algebra* (<https://books.google.com/books?id=JBXDc83jRBwC&pg=PA55>), Software, Environments, and Tools, vol. 22, Society for Industrial and Applied Mathematics, p. 55, ISBN 9780898719901.
10. Tarjan (1983), p. 77.

11. [Johnson, Donald B.](#) (December 1975), "Priority queues with update and finding minimum spanning trees", *Information Processing Letters*, **4** (3): 53–57, doi:[10.1016/0020-0190\(75\)90001-0](https://doi.org/10.1016/0020-0190(75)90001-0) (<https://doi.org/10.1016%2F0020-0190%2875%2990001-0>).
12. [Grama, Ananth](#); [Gupta, Anshul](#); [Karypis, George](#); [Kumar, Vipin](#) (2003), *Introduction to Parallel Computing*, Addison-Wesley, pp. 444–446, ISBN 978-0201648652
13. [Quinn, Michael J.](#); [Deo, Narsingh](#) (1984), "Parallel graph algorithms", *ACM Computing Surveys*, **16** (3): 319–348, doi:[10.1145/2514.2515](https://doi.org/10.1145/2514.2515) (<https://doi.org/10.1145%2F2514.2515>) , S2CID 6833839 (<https://api.semanticscholar.org/CorpusID:6833839>)
14. [Setia, Rohit](#) (2009), "A new parallel algorithm for minimum spanning tree problem" (<https://ncit-cluster.grid.pub.ro/trac/PP2009/export/157/proiecte/pgraph/Documentation/parallelsparntree.pdf>) (PDF), *Proc. International Conference on High Performance Computing (HiPC)*

## External links

---

- [Prim's Algorithm progress on randomly distributed points](https://meyavuz.wordpress.com/2017/03/10/prims-algorithm-animation-for-randomly-distributed-points) (<https://meyavuz.wordpress.com/2017/03/10/prims-algorithm-animation-for-randomly-distributed-points>)
  -  Media related to [Prim's algorithm](#) at Wikimedia Commons
- 

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Prim%27s\\_algorithm&oldid=1253353638](https://en.wikipedia.org/w/index.php?title=Prim%27s_algorithm&oldid=1253353638)"