Prim's algorithm

From Wikipedia, the free encyclopedia

In computer science, **Prim's algorithm** is a greedy algorithm that finds a minimum spanning tree for a connected 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 was developed in 1930 by Czech mathematician Vojtěch Jarník and later independently by computer scientist Robert C. Prim in 1957 and rediscovered by Edsger Dijkstra in 1959. Therefore it is also sometimes called the **DJP algorithm**, the **Jarník algorithm**, or the **Prim–Jarník algorithm**.

Other algorithms for this problem include Kruskal's algorithm and Borůvka's algorithm.

Contents

Description

The only spanning tree of the empty graph (with an empty vertex set) is again the empty graph. The following description assumes that this special case is handled separately.

The algorithm continuously increases the size of a tree, one edge at a time, starting with a tree consisting of a single vertex, until it spans all vertices.

- Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be negative).
- Initialize: $V_{\text{new}} = \{x\}$, where x is an arbitrary node (starting point) from V, $E_{\text{new}} = \{\}$
- Repeat until $V_{\text{new}} = V$:
  - Choose an edge (u, v) with minimal weight such that u is in V_{new} and v is not (if there are multiple edges with the same weight, any of them may be picked)
  - Add v to V_{new}, and (u, v) to E_{new}
- Output: V_{new} and E_{new} describe a minimal spanning tree

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.

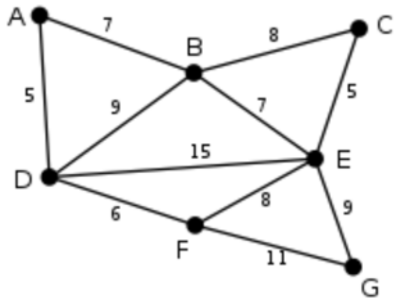
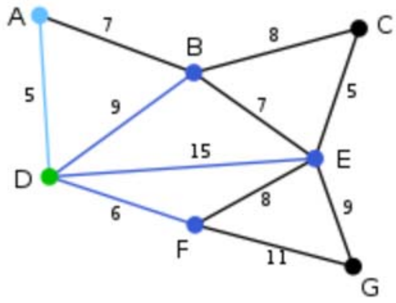
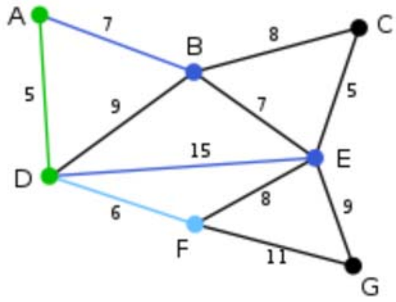
Time complexity

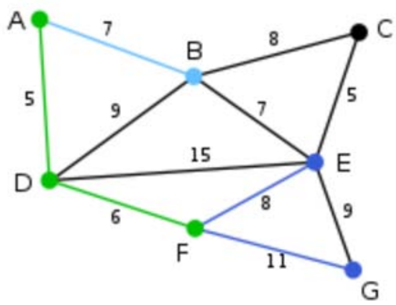
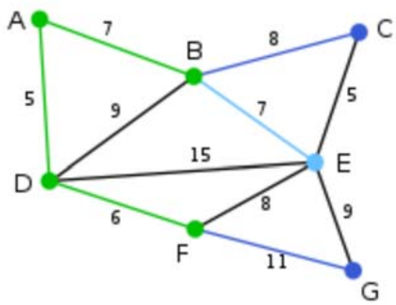
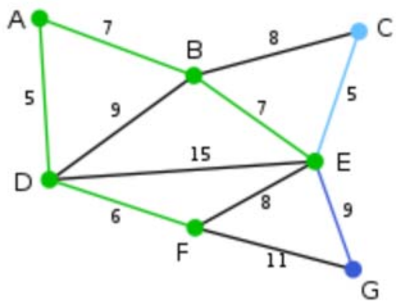
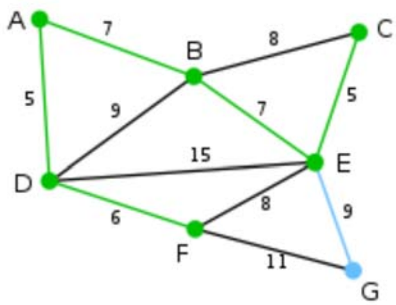
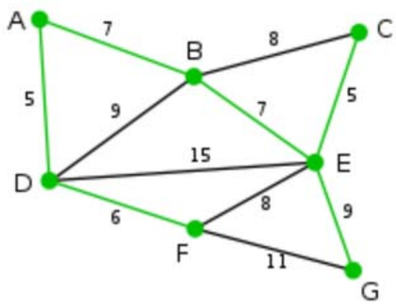
Minimum edge weight data structure	Time complexity (total)

adjacency matrix, searching	$O(V^2)$
binary heap and adjacency list	O((V + E) log V) = O(E log V)
Fibonacci heap and adjacency list	O(E + V log V)

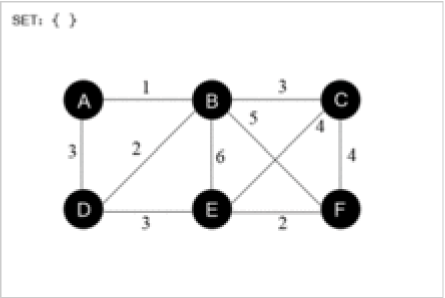
A simple implementation using an adjacency matrix graph representation and searching an array of weights to find the minimum weight edge to add requires $O(V^2)$ running time. Using a simple binary heap data structure and an adjacency list representation, Prim's algorithm can be shown to run in time 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 Ω(*V*).

Example run

Image	U	Edge(u,v)	V \ U	Description
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{}		{A,B,C,D,E,F,G}	This is our original weighted graph. The numbers near the edges indicate their weight.
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{D}	(D,A) = 5 **V** (D,B) = 9 (D,E) = 15 (D,F) = 6	{A,B,C,E,F,G}	Vertex **D** has been arbitrarily chosen as a starting point. Vertices **A**, **B**, **E** and **F** are connected to **D** through a single edge. **A** is the vertex nearest to **D** and will be chosen as the second vertex along with the edge **AD**.
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{A,D}	(D,B) = 9 (D,E) = 15 (D,F) = 6 **V** (A,B) = 7	{B,C,E,F,G}	The next vertex chosen is the vertex nearest to *either* **D** or **A**. **B** is 9 away from **D** and 7 away from **A**, **E** is 15, and **F** is 6. **F** is the smallest distance away, so we highlight the vertex **F** and the arc **DF**.

 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{A,D,F}	(D,B) = 9 (D,E) = 15 (A,B) = 7 **V** (F,E) = 8 (F,G) = 11	{B,C,E,G}	The algorithm carries on as above. Vertex **B**, which is 7 away from **A**, is highlighted.
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{A,B,D,F}	(B,C) = 8 (B,E) = 7 **V** (D,B) = 9 cycle (D,E) = 15 (F,E) = 8 (F,G) = 11	{C,E,G}	In this case, we can choose between **C**, **E**, and **G**. **C** is 8 away from **B**, **E** is 7 away from **B**, and **G** is 11 away from **F**. **E** is nearest, so we highlight the vertex **E** and the arc **BE**.
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{A,B,D,E,F}	(B,C) = 8 (D,B) = 9 cycle (D,E) = 15 cycle (E,C) = 5 **V** (E,G) = 9 (F,E) = 8 cycle (F,G) = 11	{C,G}	Here, the only vertices available are **C** and **G**. **C** is 5 away from **E**, and **G** is 9 away from **E**. **C** is chosen, so it is highlighted along with the arc **EC**.
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{A,B,C,D,E,F}	(B,C) = 8 cycle (D,B) = 9 cycle (D,E) = 15 cycle (E,G) = 9 **V** (F,E) = 8 cycle (F,G) = 11	{G}	Vertex **G** is the only remaining vertex. It is 11 away from **F**, and 9 away from **E**. **E** is nearer, so we highlight **G** and the arc **EG**.
 A 7 B 8 C 5 9 7 5 15 D E 6 8 9 F 11 G	{A,B,C,D,E,F,G}	(B,C) = 8 cycle (D,B) = 9 cycle (D,E) = 15 cycle (F,E) = 8 cycle (F,G) = 11 cycle	{}	Now all the vertices have been selected and the minimum spanning tree is shown in green. In this case, it has weight 39.

Proof of correctness



This animation shows how the Prim's algorithm runs in a graph.(Click the image to see the animation)

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 Y are connected. Let Y_1 be a minimum spanning tree of P. If Y_1=Y then Y is a minimum spanning tree. Otherwise, let e be the first edge added during the construction of Y that is not in Y_1, and V be the set of vertices connected by the edges added before e. Then one endpoint of e is in V and the other is not. Since Y_1 is a spanning tree of P, there is a path in Y_1 joining the two endpoints. As one travels along the path, one must encounter an edge f joining a vertex in V to one that is not in V. Now, at the iteration when e was added to Y, f could also have been added and it would be added instead of e if its weight was less than e. Since f was not added, we conclude that

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

Let Y_2 be the graph obtained by removing f from and adding e to Y_1. It is easy to show that Y_2 is connected, has the same number of edges as Y_1, and the total weights of its edges is not larger than that of Y_1, therefore it is also a minimum spanning tree of P and it contains e and all the edges added before it during the construction of V. Repeat the steps above and we will eventually obtain a minimum spanning tree of P that is identical to Y. This shows Y is a minimum spanning tree.

References

- V. Jarník: *O jistém problému minimálním* [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63. (in Czech)
- R. C. Prim: *Shortest connection networks and some generalizations*. In: *Bell System Technical Journal*, 36 (1957), pp. 1389–1401
- D. Cheriton and R. E. Tarjan: *Finding minimum spanning trees*. In: *SIAM Journal on Computing*, 5 (Dec. 1976), pp. 724–741
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Third Edition. MIT Press, 2009. ISBN 0-262-03384-4. Section 23.2: The algorithms of Kruskal and Prim, pp. 631–638.

External links

- Animated example of Prim's algorithm (http://students.ceid.upatras.gr/~papagel/project/prim.htm)
- Prim's Algorithm (Java Applet) (http://www.mincel.com/java/prim.html)
- Minimum spanning tree demonstration Python program by Ronald L. Rivest (http://people.csail.mit.edu/rivest/programs.html)
- Open Source Java Graph package with implementation of Prim's Algorithm (http://code.google.com/p/annas/)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Prim%27s_algorithm&oldid=484712762"

Categories: Graph algorithms | Spanning tree

- This page was last modified on 30 March 2012 at 15:56.