Kruskal's algorithm

From Wikipedia, the free encyclopedia

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted 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. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

This algorithm first appeared in *Proceedings of the American Mathematical Society*, pp. 48–50 in 1956, and was written by Joseph Kruskal.

Other algorithms for this problem include Prim's algorithm, Reverse-Delete algorithm, and Borůvka's algorithm.

Contents

Description

- create a forest *F* (a set of trees), where each vertex in the graph is a separate tree
- create a set *S* containing all the edges in the graph
- while *S* is nonempty and F is not yet spanning
  - remove an edge with minimum weight from *S*
  - if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
  - otherwise discard that edge.

At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph

Performance

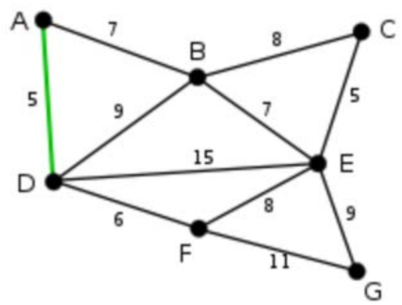
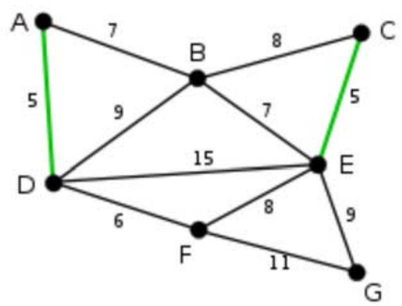
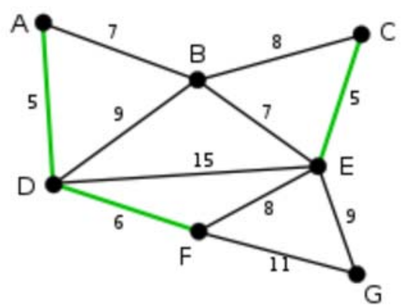
Where *E* is the number of edges in the graph and *V* is the number of vertices, Kruskal's algorithm can be shown to run in $O(E \log E)$ time, or equivalently, $O(E \log V)$ time, all with simple data structures. These running times are equivalent because:

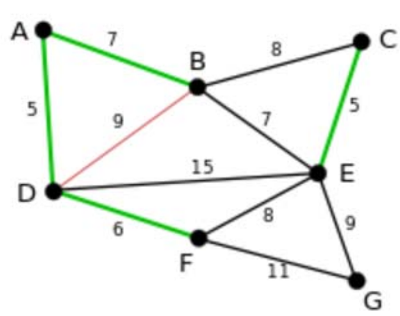
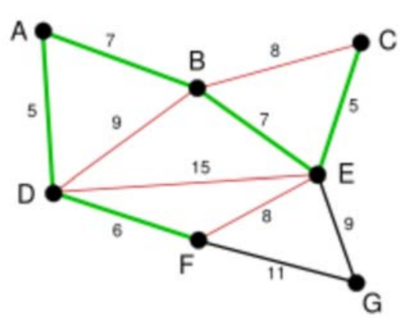
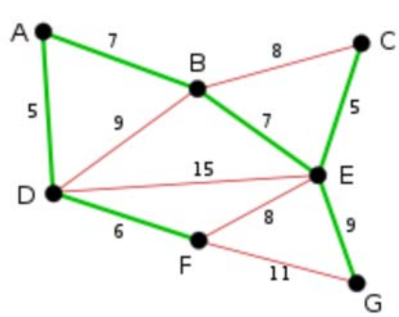
- E is at most V^2 and $\log V^2 = 2 \log V$ is $O(\log V)$.
- If we ignore isolated vertices, which will each be their own component of the minimum spanning forest, $V \leq E+1$, so $\log V$ is $O(\log E)$.

We can achieve this bound as follows: first sort the edges by weight using a comparison sort in $O(E \log E)$ time; this allows the step "remove an edge with minimum weight from S" to operate in constant time. Next, we use a disjoint-set data structure (Union&Find) to keep track of which vertices are in which components. We need to perform $O(E)$ operations, two 'find' operations and possibly one union for each edge. Even a simple disjoint-set data structure such as disjoint-set forests with union by rank can perform $O(E)$ operations in $O(E \log V)$ time. Thus the total time is $O(E \log E) = O(E \log V)$.

Provided that the edges are either already sorted or can be sorted in linear time (for example with counting sort or radix sort), the algorithm can use more sophisticated disjoint-set data structure to run in $O(E\ \alpha(V))$ time, where α is the extremely slowly-growing inverse of the single-valued Ackermann function.

Example

Image	Description
	**AD** and **CE** are the shortest arcs, with length 5, and **AD** has been arbitrarily chosen, so it is highlighted.
	**CE** is now the shortest arc that does not form a cycle, with length 5, so it is highlighted as the second arc.
	The next arc, **DF** with length 6, is highlighted using much the same method.

<table>
<tr>
<td>
</td>
<td>The next-shortest arcs are AB and BE, both with length 7. AB is chosen arbitrarily, and is highlighted. The arc BD has been highlighted in red, because there already exists a path (in green) between B and D, so it would form a cycle (ABD) if it were chosen.</td>
</tr>
<tr>
<td>
</td>
<td>The process continues to highlight the next-smallest arc, BE with length 7. Many more arcs are highlighted in red at this stage: BC because it would form the loop BCE, DE because it would form the loop DEBA, and FE because it would form FEBAD.</td>
</tr>
<tr>
<td>
</td>
<td>Finally, the process finishes with the arc EG of length 9, and the minimum spanning tree is found.</td>
</tr>
</table>

Proof of correctness

The proof consists of two parts. First, it is proved that the algorithm produces a spanning tree. Second, it is proved that the constructed spanning tree is of minimal weight.

Spanning Tree

Let P be a connected, weighted graph and let Y be the subgraph of P produced by the algorithm. Y cannot have a cycle, since the last edge added to that cycle would have been within one subtree and not between two different trees. Y cannot be disconnected, since the first encountered edge that joins two components of Y would have been added by the algorithm. Thus, Y is a spanning tree of P.

Minimality

We show that the following proposition ***P*** is true by induction: If *F* is the set of edges chosen at any stage of the algorithm, then there is some minimum spanning tree that contains *F*.

- Clearly ***P*** is true at the beginning, when *F* is empty: any minimum spanning tree will do, and there exists one because a weighted connected graph always has a minimum spanning tree.
- Now assume ***P*** is true for some non-final edge set *F* and let *T* be a minimum spanning tree that contains *F*.

If the next chosen edge e is also in T, then $\boldsymbol{P}$ is true for $F + e$. Otherwise, $T + e$ has a cycle C and there is another edge f that is in C but not F. (If there were no such edge f, then e could not have been added to F, since doing so would have created the cycle C.) Then $T - f + e$ is a tree, and it has the same weight as T, since T has minimum weight and the weight of f cannot be less than the weight of e, otherwise the algorithm would have chosen f instead of e. So $T - f + e$ is a minimum spanning tree containing $F + e$ and again $\boldsymbol{P}$ holds.

- Therefore, by the principle of induction, $\boldsymbol{P}$ holds when F has become a spanning tree, which is only possible if F is a minimum spanning tree itself.

See also

- Reverse-Delete algorithm
- Dijkstra's algorithm
- Prim's algorithm

References

- Joseph. B. Kruskal: *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem (http://links.jstor.org/sici?sici=0002-9939(195602)7%3A1%3C48%3AOTSSSO%3E2.0.CO%3B2-M)* . In: *Proceedings of the American Mathematical Society*, Vol 7, No. 1 (Feb, 1956), pp. 48–50
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 23.2: The algorithms of Kruskal and Prim, pp. 567–574.
- Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*, Fourth Edition. John Wiley & Sons, Inc., 2006. ISBN 0-471-73884-0. Section 13.7.1: Kruskal's Algorithm, pp. 632.

External links

- Animation of Kruskal's algorithm (Requires Java plugin) (http://students.ceid.upatras.gr/~papagel/project/kruskal.htm)
- C# Implementation (http://www.codeproject.com/KB/recipes/Kruskal_Algorithm.aspx)