Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Deutsch
Español
فارسی
Français
한국어
Հայերեն
Italiano
עברית
Magyar
Монгол
Nederlands
日本語
Polski
Português
Română
Русский
Simple English
Slovenčina
Српски / srpski
Svenska
ไทย
Українська
Tiếng Việt
中文
Edit links

Kruskal's algorithm

From Wikipedia, the free encyclopedia

> This article includes a list of references, related reading or external links, **but its sources remain unclear because it lacks inline citations.** Please improve this article by introducing more precise citations. *(June 2013)*

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.[1]It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.[1] 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).

This algorithm first appeared in *Proceedings of the American Mathematical Society*, pp. 48–50 in 1956, and was written by Joseph Kruskal.[2]

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

Graph and tree search algorithms

α–β · A* · B* · Backtracking · Beam · Bellman–Ford · Best-first · Bidirectional · Borůvka · Branch & bound · BFS · British Museum · D* · DFS · Depth-limited · Dijkstra · Edmonds · Floyd–Warshall · Fringe search · Hill climbing · IDA* · Iterative deepening · Johnson · Jump point · **Kruskal** · Lexicographic BFS · Prim · SMA*

Listings

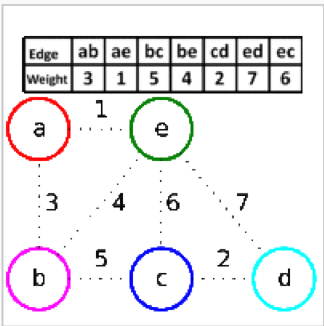
Graph algorithms · *Search algorithms* · *List of graph algorithms*

Related topics

Dynamic programming · Graph traversal · Tree traversal · Search games

v · t · e

Contents [hide]



Visualization of Kruskal's algorithm

Description [edit]

- 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 the removed edge connects two different trees then add it to the forest *F*, combining two trees into a single tree

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Pseudocode [edit]

The following code is implemented with disjoint-set data structure:

```
KRUSKAL(G):
A = ∅
foreach v ∈ G.V:
   MAKE-SET(v)
foreach (u, v) ordered by weight(u, v), increasing:
   if FIND-SET(u) ≠ FIND-SET(v):
```

```
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Complexity [edit]

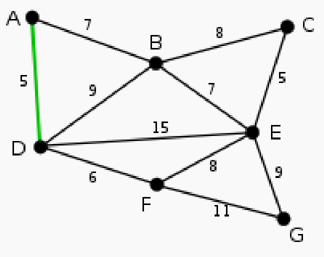
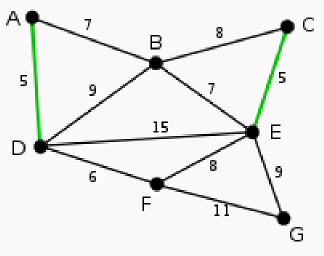
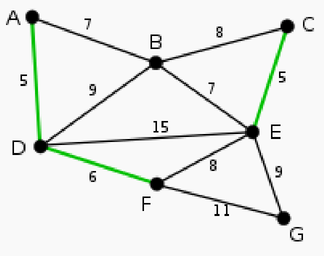
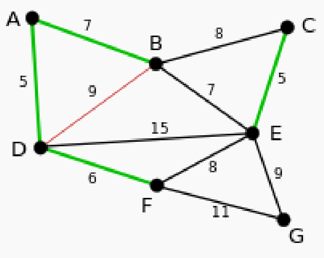
Where *E* is the number of edges in the graph and *V* is the number of vertices, Kruskal's algorithm can be shown to run in *O*(*E* log *E*) time, or equivalently, *O*(*E* log *V*) time, all with simple data structures. These running times are equivalent because:

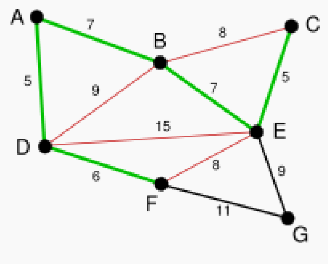
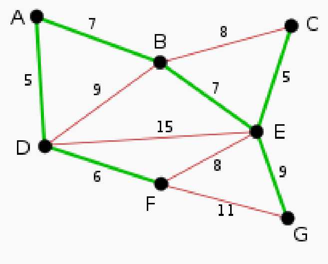
- *E* is at most V^2 and $\log V^2 = 2 \log V$ is *O*(log *V*).
- Each isolated vertex is a separate component of the minimum spanning forest. If we ignore isolated vertices we obtain *V* ≤ *E*+1, so log *V* is *O*(log *E*).

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(*V*) operations, as in each iteration we connect a vertex to the spanning tree, 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(*V*) operations in *O*(*V* 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* α(*V*)) time, where α is the extremely slowly growing inverse of the single-valued Ackermann function.

Example [edit]

<table>
<tr><th>Image</th><th>Description</th></tr>
<tr><td>
</td><td>AD and CE are the shortest edges, with length 5, and AD has been arbitrarily chosen, so it is highlighted.</td></tr>
<tr><td>
</td><td>CE is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.</td></tr>
<tr><td>
</td><td>The next edge, DF with length 6, is highlighted using much the same method.</td></tr>
<tr><td>
</td><td>The next-shortest edges are AB and BE, both with length 7. AB is chosen arbitrarily, and is highlighted. The edge 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></td></tr>
</table>

	The process continues to highlight the next-smallest edge, **BE** with length 7. Many more edges 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**.
	Finally, the process finishes with the edge **EG** of length 9, and the minimum spanning tree is found.

Proof of correctness [edit]

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 [edit]

Let P be a connected, weighted graph and let Y be the subgraph of P produced by the algorithm. Y cannot have a cycle, being 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 [edit]

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

See also [edit]

- Dijkstra's algorithm
- Prim's algorithm
- Borůvka's algorithm
- Reverse-delete algorithm
- Single-linkage clustering

References [edit]

1. ^ *a* *b* Cormen, Thomas; Charles E Leiserson, Ronald L Rivest, Clifford Stein (2009). *Introduction To Algorithms* (Third ed.). MIT Press. p. 631. ISBN 0262258102.
2. ^ Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". *Proceedings of the American Mathematical Society* **7**: 48–50. doi:10.1090/S0002-9939-1956-0078686-7. JSTOR 2033241.

- 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..

Categories: Graph algorithms | Spanning tree