Dijkstra's algorithm

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#searchInput)

*Not to be confused with*[*Dykstra's projection algorithm*](https://en.wikipedia.org/wiki/Dykstra%27s_projection_algorithm)*.*

|  |  |
| --- | --- |
| **Dijkstra's algorithm** | |
| Dijkstra's algorithm to find the shortest path between *a* and *b*. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors. | |
| **Class** | [Search algorithm](https://en.wikipedia.org/wiki/Search_algorithm) [Greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm) [Dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming)[[1]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-1) |
| **Data structure** | [Graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) Usually used with [Priority queue](https://en.wikipedia.org/wiki/Priority_queue)/[Heap](https://en.wikipedia.org/wiki/Heap_(data_structure)) for optimization |
| [**Worst-case**](https://en.wikipedia.org/wiki/Best,_worst_and_average_case)[**performance**](https://en.wikipedia.org/wiki/Time_complexity) | {\displaystyle \Theta (|E|+|V|\log |V|)} |

|  |
| --- |
| [**Graph**](https://en.wikipedia.org/wiki/Graph_traversal)**and**[**tree search algorithms**](https://en.wikipedia.org/wiki/Tree_traversal) |
| * [α–β](https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning) * [A\*](https://en.wikipedia.org/wiki/A*_search_algorithm) * [B\*](https://en.wikipedia.org/wiki/B*) * [Backtracking](https://en.wikipedia.org/wiki/Backtracking) * [Beam](https://en.wikipedia.org/wiki/Beam_search) * [Bellman–Ford](https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm) * [Best-first](https://en.wikipedia.org/wiki/Best-first_search) * [Bidirectional](https://en.wikipedia.org/wiki/Bidirectional_search) * [Borůvka](https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm) * [Branch & bound](https://en.wikipedia.org/wiki/Branch_and_bound) * [BFS](https://en.wikipedia.org/wiki/Breadth-first_search) * [British Museum](https://en.wikipedia.org/wiki/British_Museum_algorithm) * [D\*](https://en.wikipedia.org/wiki/D*) * [DFS](https://en.wikipedia.org/wiki/Depth-first_search) * Dijkstra * [Edmonds](https://en.wikipedia.org/wiki/Edmonds%27_algorithm) * [Floyd–Warshall](https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm) * [Fringe search](https://en.wikipedia.org/wiki/Fringe_search) * [Hill climbing](https://en.wikipedia.org/wiki/Hill_climbing) * [IDA\*](https://en.wikipedia.org/wiki/Iterative_deepening_A*) * [Iterative deepening](https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search) * [Johnson](https://en.wikipedia.org/wiki/Johnson%27s_algorithm) * [Jump point](https://en.wikipedia.org/wiki/Jump_point_search) * [Kruskal](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm) * [Lexicographic BFS](https://en.wikipedia.org/wiki/Lexicographic_breadth-first_search) * [LPA\*](https://en.wikipedia.org/wiki/Lifelong_Planning_A*) * [Prim](https://en.wikipedia.org/wiki/Prim%27s_algorithm) * [SMA\*](https://en.wikipedia.org/wiki/SMA*) * [SPFA](https://en.wikipedia.org/wiki/Shortest_Path_Faster_Algorithm) |
| **Listings** |
| * [*Graph algorithms*](https://en.wikipedia.org/wiki/Category:Graph_algorithms) * [*Search algorithms*](https://en.wikipedia.org/wiki/Category:Search_algorithms) * [*List of graph algorithms*](https://en.wikipedia.org/wiki/List_of_algorithms#Graph_algorithms) |
| **Related topics** |
| * [Dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming) * [Graph traversal](https://en.wikipedia.org/wiki/Graph_traversal) * [Tree traversal](https://en.wikipedia.org/wiki/Tree_traversal) * [Search games](https://en.wikipedia.org/wiki/Search_game) |
| * [v](https://en.wikipedia.org/wiki/Template:Graph_search_algorithm) * [t](https://en.wikipedia.org/wiki/Template_talk:Graph_search_algorithm) * [e](https://en.wikipedia.org/w/index.php?title=Template:Graph_search_algorithm&action=edit) |

**Dijkstra's algorithm** (or **Dijkstra's Shortest Path First algorithm**, **SPF algorithm**)[[2]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-2) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for finding the [shortest paths](https://en.wikipedia.org/wiki/Shortest_path_problem) between [nodes](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) in a [graph](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)), which may represent, for example, [road networks](https://en.wikipedia.org/wiki/Road_network). It was conceived by [computer scientist](https://en.wikipedia.org/wiki/Computer_scientist) [Edsger W. Dijkstra](https://en.wikipedia.org/wiki/Edsger_W._Dijkstra) in 1956 and published three years later.[[3]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-3)[[4]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-Dijkstra_Interview-4)[[5]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-Dijkstra1959-5)

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes,[[5]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-Dijkstra1959-5) but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a [shortest-path tree](https://en.wikipedia.org/wiki/Shortest-path_tree).

For a given source node in the graph, the algorithm finds the shortest path between that node and every other.[[6]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-mehlhorn-6):196–206 It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road (for simplicity, ignore red lights, stop signs, toll roads and other obstructions), Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. A widely used application of shortest path algorithm is network [routing protocols](https://en.wikipedia.org/wiki/Routing_protocol), most notably [IS-IS](https://en.wikipedia.org/wiki/IS-IS) (Intermediate System to Intermediate System) and Open Shortest Path First ([OSPF](https://en.wikipedia.org/wiki/OSPF)). It is also employed as a [subroutine](https://en.wikipedia.org/wiki/Subroutine) in other algorithms such as [Johnson's](https://en.wikipedia.org/wiki/Johnson%27s_algorithm).

The Dijkstra algorithm uses labels that are positive integers or real numbers, which are [totally ordered](https://en.wikipedia.org/wiki/Total_order). It can be generalized to use any labels that are [partially ordered](https://en.wikipedia.org/wiki/Partially_ordered_set), provided the subsequent labels (a subsequent label is produced when traversing an edge) are [monotonically](https://en.wikipedia.org/wiki/Monotonic_function) non-decreasing. This generalization is called the generic Dijkstra shortest-path algorithm.[[7]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-Generic_Dijkstra-7)

Dijkstra's algorithm uses a data structure for storing and querying partial solutions sorted by distance from the start. While the original algorithm uses a [min-priority queue](https://en.wikipedia.org/wiki/Min-priority_queue) and runs in [time](https://en.wikipedia.org/wiki/Time_complexity) {\displaystyle \Theta ((|V|+|E|)\log |V|)}(where {\displaystyle |V|} is the number of nodes and {\displaystyle |E|} is the number of edges), it can also be implemented in {\displaystyle \Theta (|V|^{2})} using an array. The idea of this algorithm is also given in [Leyzorek et al. 1957](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFLeyzorekGrayJohnsonLadew1957). [Fredman & Tarjan 1984](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFFredmanTarjan1984) propose using a [Fibonacci heap](https://en.wikipedia.org/wiki/Fibonacci_heap) min-priority queue to optimize the running time complexity to {\displaystyle \Theta (|E|+|V|\log |V|)}. This is [asymptotically](https://en.wikipedia.org/wiki/Asymptotic_computational_complexity) the fastest known single-source [shortest-path algorithm](https://en.wikipedia.org/wiki/Shortest_path_problem) for arbitrary [directed graphs](https://en.wikipedia.org/wiki/Directed_graph) with unbounded non-negative weights. However, specialized cases (such as bounded/integer weights, directed acyclic graphs etc.) can indeed be improved further as detailed in [Specialized variants](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Specialized_variants).

In some fields, [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) in particular, Dijkstra's algorithm or a variant of it is known as **uniform cost search** and formulated as an instance of the more general idea of [best-first search](https://en.wikipedia.org/wiki/Best-first_search).[[8]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-felner-8)



**Contents**

* [1History](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#History)
* [2Algorithm](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Algorithm)
* [3Description](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Description)
* [4Pseudocode](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Pseudocode)
  + [4.1Using a priority queue](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Using_a_priority_queue)
* [5Proof of correctness](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Proof_of_correctness)
* [6Running time](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Running_time)
  + [6.1Practical optimizations and infinite graphs](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Practical_optimizations_and_infinite_graphs)
  + [6.2Specialized variants](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Specialized_variants)
* [7Related problems and algorithms](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Related_problems_and_algorithms)
  + [7.1Dynamic programming perspective](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Dynamic_programming_perspective)
* [8See also](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#See_also)
* [9Notes](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Notes)
* [10References](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#References)
* [11External links](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#External_links)

History[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=1)]

What is the shortest way to travel from [Rotterdam](https://en.wikipedia.org/wiki/Rotterdam) to [Groningen](https://en.wikipedia.org/wiki/Groningen), in general: from given city to given city. [It is the algorithm for the shortest path](https://en.wikipedia.org/wiki/Shortest_path_problem), which I designed in about twenty minutes. One morning I was shopping in [Amsterdam](https://en.wikipedia.org/wiki/Amsterdam) with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in '59, three years later. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually, that algorithm became to my great amazement, one of the cornerstones of my fame.

— *Edsger Dijkstra, in an interview with Philip L. Frana, Communications of the ACM, 2001*[[4]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-Dijkstra_Interview-4)

Dijkstra thought about the shortest path problem when working at the [Mathematical Center in Amsterdam](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) in 1956 as a programmer to demonstrate the capabilities of a new computer called ARMAC.[[9]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-9) His objective was to choose both a problem and a solution (that would be produced by computer) that non-computing people could understand. He designed the shortest path algorithm and later implemented it for ARMAC for a slightly simplified transportation map of 64 cities in the Netherlands (64, so that 6 bits would be sufficient to encode the city number).[[4]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-Dijkstra_Interview-4) A year later, he came across another problem from hardware engineers working on the institute's next computer: minimize the amount of wire needed to connect the pins on the back panel of the machine. As a solution, he re-discovered the algorithm known as [Prim's minimal spanning tree algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm) (known earlier to [Jarník](https://en.wikipedia.org/wiki/Vojt%C4%9Bch_Jarn%C3%ADk), and also rediscovered by [Prim](https://en.wikipedia.org/wiki/Robert_C._Prim)).[[10]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-EWD841a-10)[[11]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-11) Dijkstra published the algorithm in 1959, two years after Prim and 29 years after Jarník.[[12]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-12)[[13]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-13)

Algorithm[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=2)]

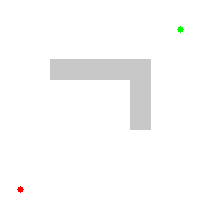
[](https://en.wikipedia.org/wiki/File:Dijkstras_progress_animation.gif)

Illustration of Dijkstra's algorithm finding a path from a start node (lower left, red) to a goal node (upper right, green) in a [robot](https://en.wikipedia.org/wiki/Robotics) [motion planning](https://en.wikipedia.org/wiki/Motion_planning) problem. Open nodes represent the "tentative" set (aka set of "unvisited" nodes). Filled nodes are visited ones, with color representing the distance: the greener, the closer. Nodes in all the different directions are explored uniformly, appearing more-or-less as a circular [wavefront](https://en.wikipedia.org/wiki/Wavefront) as Dijkstra's algorithm uses a [heuristic](https://en.wikipedia.org/wiki/Consistent_heuristic) identically equal to 0.

Let the node at which we are starting be called the **initial node**. Let the **distance of node *Y*** be the distance from the **initial node** to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.[[14]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-14)
3. For the current node, consider all of its unvisited neighbours and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbour *B* has length 2, then the distance to *B* through *A* will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current").

Description[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=3)]

***Note:****For ease of understanding, this discussion uses the terms****intersection****,****road****and****map****– however, in formal terminology these terms are****vertex****,****edge****and****graph****, respectively.*

Suppose you would like to find the *shortest path* between two [intersections](https://en.wikipedia.org/wiki/Intersection_(road)) on a city map: a *starting point* and a *destination*. Dijkstra's algorithm initially marks the distance (from the starting point) to every other intersection on the map with *infinity*. This is done not to imply that there is an infinite distance, but to note that those intersections have not been visited yet. Some variants of this method leave the intersections' distances *unlabeled*. Now select the *current intersection* at each iteration. For the first iteration, the current intersection will be the starting point, and the distance to it (the intersection's label) will be *zero*. For subsequent iterations (after the first), the current intersection will be a *closest unvisited intersection* to the starting point (this will be easy to find).

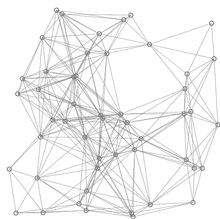
From the current intersection, *update* the distance to every unvisited intersection that is directly connected to it. This is done by determining the *sum* of the distance between an unvisited intersection and the value of the current intersection and then [relabeling](https://en.wikipedia.org/wiki/Graph_labeling) the unvisited intersection with this value (the sum) if it is less than the unvisited intersection's current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each [neighboring intersection](https://en.wikipedia.org/wiki/Neighbourhood_(graph_theory)), mark the current intersection as *visited* and select an unvisited intersection with minimal distance (from the starting point) – or the lowest label—as the current intersection. Intersections marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to.

Continue this process of updating the neighboring intersections with the shortest distances, marking the current intersection as visited, and moving onto a closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited (as is the case with any visited intersection), you have determined the shortest path to it from the starting point and can *trace your way back following the arrows in reverse*. In the algorithm's implementations, this is usually done (after the algorithm has reached the destination node) by following the nodes' parents from the destination node up to the starting node; that's why we also keep track of each node's parent.

This algorithm makes no attempt of direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm therefore expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies.

Pseudocode[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=4)]

In the following [pseudocode](https://en.wikipedia.org/wiki/Pseudocode) algorithm, the code u ← vertex in *Q* with min dist[u], searches for the vertex *u* in the vertex set *Q* that has the least dist[*u*] value. length(*u*, *v*) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes *u* and *v*. The variable *alt* on line 18 is the length of the path from the root node to the neighbor node *v* if it were to go through *u*. If this path is shorter than the current shortest path recorded for *v*, that current path is replaced with this *alt* path. The prev array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

[](https://en.wikipedia.org/wiki/File:DijkstraDemo.gif)

A demo of Dijkstra's algorithm based on Euclidean distance. Red lines are the shortest path covering, i.e., connecting *u* and prev[*u*]. Blue lines indicate where relaxing happens, i.e., connecting *v* with a node *u* in *Q*, which gives a shorter path from the source to *v*.

1 **function** Dijkstra(*Graph*, *source*):

2

3 create vertex set Q

4

5 **for each** vertex *v* in *Graph*:

6 dist[*v*] ← INFINITY

7 prev[*v*] ← UNDEFINED

8 add *v* to *Q*

10 dist[*source*] ← 0

11

12 **while** *Q* is not empty:

13 *u* ← vertex in *Q* with min dist[u]

14

15 remove *u* from *Q*

16

17 **for each** neighbor *v* of *u*: *// only v that are still in Q*

18 *alt* ← dist[*u*] + length(*u*, *v*)

19 **if** *alt* < dist[*v*]:

20 dist[*v*] ← *alt*

21 prev[*v*] ← *u*

22

23 **return** dist[], prev[]

If we are only interested in a shortest path between vertices *source* and *target*, we can terminate the search after line 15 if *u* = *target*. Now we can read the shortest path from *source* to *target* by reverse iteration:

1 *S* ← empty sequence

2 *u* ← *target*

3 **if** prev[*u*] is defined **or** *u* = *source*: *// Do something only if the vertex is reachable*

4 **while** *u* is defined: *// Construct the shortest path with a stack S*

5 insert *u* at the beginning of *S* *// Push the vertex onto the stack*

6 *u* ← prev[*u*] *// Traverse from target to source*

Now sequence *S* is the list of vertices constituting one of the shortest paths from *source* to *target*, or the empty sequence if no path exists.

A more general problem would be to find all the shortest paths between *source* and *target* (there might be several different ones of the same length). Then instead of storing only a single node in each entry of prev[] we would store all nodes satisfying the relaxation condition. For example, if both *r* and *source* connect to *target* and both of them lie on different shortest paths through *target* (because the edge cost is the same in both cases), then we would add both *r* and *source* to prev[*target*]. When the algorithm completes, prev[] data structure will actually describe a graph that is a subset of the original graph with some edges removed. Its key property will be that if the algorithm was run with some starting node, then every path from that node to any other node in the new graph will be the shortest path between those nodes in the original graph, and all paths of that length from the original graph will be present in the new graph. Then to actually find all these shortest paths between two given nodes we would use a path finding algorithm on the new graph, such as [depth-first search](https://en.wikipedia.org/wiki/Depth-first_search).

**Using a priority queue**[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=5)]

A min-priority queue is an abstract data type that provides 3 basic operations : add\_with\_priority(), decrease\_priority() and extract\_min(). As mentioned earlier, using such a data structure can lead to faster computing times than using a basic queue. Notably, [Fibonacci heap](https://en.wikipedia.org/wiki/Fibonacci_heap) ([Fredman & Tarjan 1984](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFFredmanTarjan1984)) or [Brodal queue](https://en.wikipedia.org/wiki/Brodal_queue) offer optimal implementations for those 3 operations. As the algorithm is slightly different, we mention it here, in pseudo-code as well :

1 **function** Dijkstra(*Graph*, *source*):

2 dist[*source*] ← 0 *// Initialization*

3

4 create vertex priority queue Q

5

6 **for each** vertex *v* in *Graph*:

7 **if** *v* ≠ *source*

8 dist[*v*] ← INFINITY *// Unknown distance from source to v*

9 prev[*v*] ← UNDEFINED *// Predecessor of v*

10

11 *Q*.add\_with\_priority(*v*, dist[*v*])

12

13

14 **while** *Q* is not empty: *// The main loop*

15 *u* ← *Q*.extract\_min() *// Remove and return best vertex*

16 **for each** neighbor *v* of *u*: *// only v that are still in Q*

17 *alt* ← dist[*u*] + length(*u*, *v*)

18 **if** *alt* < dist[*v*]

19 dist[*v*] ← *alt*

20 prev[*v*] ← *u*

21 *Q*.decrease\_priority(*v*, *alt*)

22

23 **return** dist, prev

Instead of filling the priority queue with all nodes in the initialization phase, it is also possible to initialize it to contain only *source*; then, inside the **if** *alt* < dist[*v*] block, the *decrease\_priority* becomes an *add\_with\_priority* operation if the node is not already in the queue.[[6]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-mehlhorn-6):198

Yet another alternative is to add nodes unconditionally to the priority queue and to instead check after extraction that no shorter connection was found yet. This can be done by additionally extracting the associated priority *p* from the queue and only processing further **if** *p* < dist[*u*] inside the **while** *Q* is not empty loop.

These alternatives can use entirely array-based priority queues without decrease-key functionality which have been found to achieve even faster computing times in practice.[[15]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-chen_07-15)

Proof of correctness[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=6)]

*Proof of Dijkstra's algorithm is constructed by induction on the number of visited nodes.*

*Invariant hypothesis*: For each node v, dist[v] is the shortest distance from source to v when traveling via visited nodes only, or infinity if no such path exists. (Note: we do not assume dist[v] is the actual shortest distance for unvisited nodes.)

The base case is when there is just one visited node, namely the initial node source, in which case the hypothesis is [trivial](https://en.wikipedia.org/wiki/Triviality_(mathematics)).

Otherwise, assume the hypothesis for *n-1* visited nodes. In which case, we choose an edge vu where u has the least dist[u] of any unvisited nodes and the edge vu is such that dist[u] = dist[v] + length[v,u]. dist[u] is considered to be the shortest distance from source to u because if there were a shorter path, and if w was the first unvisited node on that path then by the original hypothesis dist[w] > dist[u] which creates a contradiction. Similarly if there were a shorter path to u without using unvisited nodes, and if the last but one node on that path were w, then we would have had dist[u] = dist[w] + length[w,u], also a contradiction.

After processing u it will still be true that for each unvisited node w, dist[w] will be the shortest distance from source to w using visited nodes only, because if there were a shorter path that doesn't go by u we would have found it previously, and if there were a shorter path using u we would have updated it when processing u.

After all nodes are visited, the shortest path from source to any node v consists only of visited nodes, therefore dist[v] is the shortest distance.

Running time[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=7)]

Bounds of the running time of Dijkstra's algorithm on a graph with edges *E* and vertices *V* can be expressed as a function of the number of edges, denoted {\displaystyle |E|}, and the number of vertices, denoted {\displaystyle |V|}, using [big-O notation](https://en.wikipedia.org/wiki/Big-O_notation). The complexity bound depends mainly on the data structure used to represent the set *Q*. In the following, upper bounds can be simplified because {\displaystyle |E|} is {\displaystyle \Theta (|V|^{2})} for any graph, but that simplification disregards the fact that in some problems, other upper bounds on {\displaystyle |E|} may hold.

For any data structure for the vertex set *Q*, the running time is in

{\displaystyle \Theta (|E|\cdot T\_{\mathrm {dk} }+|V|\cdot T\_{\mathrm {em} }),}

where {\displaystyle T\_{\mathrm {dk} }} and {\displaystyle T\_{\mathrm {em} }} are the complexities of the *decrease-key* and *extract-minimum* operations in *Q*, respectively. The simplest version of Dijkstra's algorithm stores the vertex set *Q* as an ordinary linked list or array, and extract-minimum is simply a linear search through all vertices in *Q*. In this case, the running time is {\displaystyle \Theta (|E|+|V|^{2})=\Theta (|V|^{2})}.

If the graph is stored as an adjacency list, the running time for a dense graph (i.e., where {\displaystyle |E|\in \Theta (|V|^{2})}) is

{\displaystyle \Theta ((|V|^{2})\log |V|)}.

For [sparse graphs](https://en.wikipedia.org/wiki/Sparse_graph), that is, graphs with far fewer than {\displaystyle |V|^{2}} edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of [adjacency lists](https://en.wikipedia.org/wiki/Adjacency_list) and using a [self-balancing binary search tree](https://en.wikipedia.org/wiki/Self-balancing_binary_search_tree), [binary heap](https://en.wikipedia.org/wiki/Binary_heap), [pairing heap](https://en.wikipedia.org/wiki/Pairing_heap), or [Fibonacci heap](https://en.wikipedia.org/wiki/Fibonacci_heap) as a [priority queue](https://en.wikipedia.org/wiki/Priority_queue) to implement extracting minimum efficiently. To perform decrease-key steps in a binary heap efficiently, it is necessary to use an auxiliary data structure that maps each vertex to its position in the heap, and to keep this structure up to date as the priority queue *Q* changes. With a self-balancing binary search tree or binary heap, the algorithm requires

{\displaystyle \Theta ((|E|+|V|)\log |V|)}

time in the worst case (where {\displaystyle \log } denotes the binary logarithm {\displaystyle \log \_{2}}); for connected graphs this time bound can be simplified to {\displaystyle \Theta (|E|\log |V|)}. The [Fibonacci heap](https://en.wikipedia.org/wiki/Fibonacci_heap) improves this to

{\displaystyle \Theta (|E|+|V|\log |V|).}

When using binary heaps, the [average case](https://en.wikipedia.org/wiki/Best,_worst_and_average_case) time complexity is lower than the worst-case: assuming edge costs are drawn independently from a common [probability distribution](https://en.wikipedia.org/wiki/Probability_distribution), the expected number of *decrease-key* operations is bounded by {\displaystyle \Theta (|V|\log(|E|/|V|))}, giving a total running time of[[6]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-mehlhorn-6):199–200

{\displaystyle O\left(|E|+|V|\log {\frac {|E|}{|V|}}\log |V|\right).}

**Practical optimizations and infinite graphs**[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=8)]

In common presentations of Dijkstra's algorithm, initially all nodes are entered into the priority queue. This is, however, not necessary: the algorithm can start with a priority queue that contains only one item, and insert new items as they are discovered (instead of doing a decrease-key, check whether the key is in the queue; if it is, decrease its key, otherwise insert it).[[6]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-mehlhorn-6):198 This variant has the same worst-case bounds as the common variant, but maintains a smaller priority queue in practice, speeding up the queue operations.[[8]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-felner-8)

Moreover, not inserting all nodes in a graph makes it possible to extend the algorithm to find the shortest path from a single source to the closest of a set of target nodes on infinite graphs or those too large to represent in memory. The resulting algorithm is called *uniform-cost search* (UCS) in the artificial intelligence literature[[8]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-felner-8)[[16]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-aima-16)[[17]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-17) and can be expressed in pseudocode as

**procedure** uniform\_cost\_search(Graph, start, goal) **is**

node ← start

cost ← 0

frontier ← priority queue containing node only

explored ← empty set

**do**

**if** frontier is empty **then**

**return** failure

node ← frontier.pop()

**if** node is goal **then**

**return** solution

explored.add(node)

**for each** of node's neighbors *n* **do**

**if** *n* is not in explored **then**

frontier.add(*n*)

The complexity of this algorithm can be expressed in an alternative way for very large graphs: when *C*\* is the length of the shortest path from the start node to any node satisfying the "goal" predicate, each edge has cost at least *ε*, and the number of neighbors per node is bounded by *b*, then the algorithm's worst-case time and space complexity are both in *O*(*b*1+⌊*C*\* ​⁄*ε*⌋).[[16]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-aima-16)

Further optimizations of Dijkstra's algorithm for the single-target case include [bidirectional](https://en.wikipedia.org/wiki/Bidirectional_search) variants, goal-directed variants such as the [A\* algorithm](https://en.wikipedia.org/wiki/A*_algorithm) (see [§ Related problems and algorithms](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Related_problems_and_algorithms)), graph pruning to determine which nodes are likely to form the middle segment of shortest paths (reach-based routing), and hierarchical decompositions of the input graph that reduce *s*–*t* routing to connecting *s* and *t* to their respective "[transit nodes](https://en.wikipedia.org/wiki/Transit_Node_Routing)" followed by shortest-path computation between these transit nodes using a "highway".[[18]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-speedup-18) Combinations of such techniques may be needed for optimal practical performance on specific problems.[[19]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-19)

**Specialized variants**[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=9)]

When arc weights are small integers (bounded by a parameter *C*), a [monotone priority queue](https://en.wikipedia.org/wiki/Monotone_priority_queue) can be used to speed up Dijkstra's algorithm. The first algorithm of this type was **Dial's algorithm**, which used a [bucket queue](https://en.wikipedia.org/wiki/Bucket_queue) to obtain a running time {\displaystyle \Theta (|E|+\operatorname {diam} (G))} that depends on the weighted [diameter](https://en.wikipedia.org/wiki/Distance_(graph_theory)) of a graph with integer edge weights ([Dial 1969](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFDial1969)). The use of a [Van Emde Boas tree](https://en.wikipedia.org/wiki/Van_Emde_Boas_tree) as the priority queue brings the complexity to {\displaystyle \Theta (|E|\log \log C)} ([Ahuja et al. 1990](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFAhujaMehlhornOrlinTarjan1990)). Another interesting variant based on a combination of a new [radix heap](https://en.wikipedia.org/wiki/Radix_heap) and the well-known Fibonacci heap runs in time {\displaystyle \Theta (|E|+|V|{\sqrt {\log C}})} ([Ahuja et al. 1990](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFAhujaMehlhornOrlinTarjan1990)). Finally, the best algorithms in this special case are as follows. The algorithm given by ([Thorup 2000](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFThorup2000)) runs in {\displaystyle \Theta (|E|\log \log |V|)} time and the algorithm given by ([Raman 1997](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFRaman1997)) runs in {\displaystyle \Theta (|E|+|V|\min\{(\log |V|)^{1/3+\varepsilon },(\log C)^{1/4+\varepsilon }\})} time.

Also, for [directed acyclic graphs](https://en.wikipedia.org/wiki/Directed_acyclic_graph), it is possible to find shortest paths from a given starting vertex in linear {\displaystyle \Theta (|E|+|V|)} time, by processing the vertices in a [topological order](https://en.wikipedia.org/wiki/Topological_sorting), and calculating the path length for each vertex to be the minimum length obtained via any of its incoming edges.[[20]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-20)[[21]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-21)

In the special case of integer weights and undirected connected graphs, Dijkstra's algorithm can be completely countered with a linear {\displaystyle \Theta (|E|)} complexity algorithm, given by ([Thorup 1999](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFThorup1999)).

Related problems and algorithms[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=10)]

The functionality of Dijkstra's original algorithm can be extended with a variety of modifications. For example, sometimes it is desirable to present solutions which are less than mathematically optimal. To obtain a ranked list of less-than-optimal solutions, the optimal solution is first calculated. A single edge appearing in the optimal solution is removed from the graph, and the optimum solution to this new graph is calculated. Each edge of the original solution is suppressed in turn and a new shortest-path calculated. The secondary solutions are then ranked and presented after the first optimal solution.

Dijkstra's algorithm is usually the working principle behind [link-state routing protocols](https://en.wikipedia.org/wiki/Link-state_routing_protocol), [OSPF](https://en.wikipedia.org/wiki/OSPF) and [IS-IS](https://en.wikipedia.org/wiki/IS-IS) being the most common ones.

Unlike Dijkstra's algorithm, the [Bellman–Ford algorithm](https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm) can be used on graphs with negative edge weights, as long as the graph contains no [negative cycle](https://en.wikipedia.org/wiki/Negative_cycle) reachable from the source vertex *s*. The presence of such cycles means there is no shortest path, since the total weight becomes lower each time the cycle is traversed. (This statement assumes that a "path" is allowed to repeat vertices. In [graph theory](https://en.wikipedia.org/wiki/Graph_theory) that is normally not allowed. In [theoretical computer science](https://en.wikipedia.org/wiki/Theoretical_computer_science) it often is allowed.) It is possible to adapt Dijkstra's algorithm to handle negative weight edges by combining it with the Bellman-Ford algorithm (to remove negative edges and detect negative cycles), such an algorithm is called [Johnson's algorithm](https://en.wikipedia.org/wiki/Johnson%27s_algorithm).

The [A\* algorithm](https://en.wikipedia.org/wiki/A-star_algorithm) is a generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored, if additional information is available that provides a lower bound on the "distance" to the target. This approach can be viewed from the perspective of [linear programming](https://en.wikipedia.org/wiki/Linear_programming): there is a natural [linear program for computing shortest paths](https://en.wikipedia.org/wiki/Shortest_path_problem#Linear_programming_formulation), and solutions to its [dual linear program](https://en.wikipedia.org/wiki/Dual_linear_program) are feasible if and only if they form a [consistent heuristic](https://en.wikipedia.org/wiki/Consistent_heuristic) (speaking roughly, since the sign conventions differ from place to place in the literature). This feasible dual / consistent heuristic defines a non-negative [reduced cost](https://en.wikipedia.org/wiki/Reduced_cost) and A\* is essentially running Dijkstra's algorithm with these reduced costs. If the dual satisfies the weaker condition of [admissibility](https://en.wikipedia.org/wiki/Admissible_heuristic), then A\* is instead more akin to the Bellman–Ford algorithm.

The process that underlies Dijkstra's algorithm is similar to the [greedy](https://en.wikipedia.org/wiki/Greedy_algorithm) process used in [Prim's algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm). Prim's purpose is to find a [minimum spanning tree](https://en.wikipedia.org/wiki/Minimum_spanning_tree) that connects all nodes in the graph; Dijkstra is concerned with only two nodes. Prim's does not evaluate the total weight of the path from the starting node, only the individual edges.

[Breadth-first search](https://en.wikipedia.org/wiki/Breadth-first_search) can be viewed as a special-case of Dijkstra's algorithm on unweighted graphs, where the priority queue degenerates into a FIFO queue.

The [fast marching method](https://en.wikipedia.org/wiki/Fast_marching_method) can be viewed as a continuous version of Dijkstra's algorithm which computes the geodesic distance on a triangle mesh.

**Dynamic programming perspective**[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=11)]

From a [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming) point of view, Dijkstra's algorithm is a successive approximation scheme that solves the dynamic programming functional equation for the shortest path problem by the **Reaching** method.[[22]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-sniedovich_06-22)[[23]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-denardo_03-23)[[24]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-sniedovich_10-24)

In fact, Dijkstra's explanation of the logic behind the algorithm,[[25]](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-25) namely

**Problem 2.** Find the path of minimum total length between two given nodes {\displaystyle P} and {\displaystyle Q}.

We use the fact that, if {\displaystyle R} is a node on the minimal path from {\displaystyle P} to {\displaystyle Q}, knowledge of the latter implies the knowledge of the minimal path from {\displaystyle P} to {\displaystyle R}.

is a paraphrasing of [Bellman's](https://en.wikipedia.org/wiki/Richard_Bellman) famous [Principle of Optimality](https://en.wikipedia.org/wiki/Bellman_equation#Bellman's_principle_of_optimality) in the context of the shortest path problem.

See also[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=12)]

* [A\* search algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
* [Bellman–Ford algorithm](https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm)
* [Euclidean shortest path](https://en.wikipedia.org/wiki/Euclidean_shortest_path)
* [Flood fill](https://en.wikipedia.org/wiki/Flood_fill)
* [Floyd–Warshall algorithm](https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm)
* [Johnson's algorithm](https://en.wikipedia.org/wiki/Johnson%27s_algorithm)
* [Longest path problem](https://en.wikipedia.org/wiki/Longest_path_problem)
* [Parallel all-pairs shortest path algorithm](https://en.wikipedia.org/wiki/Parallel_all-pairs_shortest_path_algorithm)

Notes[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=13)]

* 1. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-1) Controversial,see *Moshe Sniedovich (2006).*[*"Dijkstra's algorithm revisited: the dynamic programming connexion"*](https://www.infona.pl/resource/bwmeta1.element.baztech-article-BAT5-0013-0005/tab/summary)*. Control and Cybernetics.****35****: 599–620.* and [below part](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#Dynamic_programming_perspective).
  2. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-2) [*"OSPF Incremental SPF"*](https://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/ospfispf.html)*. Cisco.*
  3. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-3) *Richards, Hamilton.*[*"Edsger Wybe Dijkstra"*](http://amturing.acm.org/award_winners/dijkstra_1053701.cfm)*. A.M. Turing Award. Association for Computing Machinery. Retrieved 16 October 2017. At the Mathematical Centre a major project was building the ARMAC computer. For its official inauguration in 1956, Dijkstra devised a program to solve a problem interesting to a nontechnical audience: Given a network of roads connecting cities, what is the shortest route between two designated cities?*
  4. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-Dijkstra_Interview_4-0) [***b***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-Dijkstra_Interview_4-1) [***c***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-Dijkstra_Interview_4-2) *Frana, Phil (August 2010).*[*"An Interview with Edsger W. Dijkstra"*](https://doi.org/10.1145/1787234.1787249)*. Communications of the ACM.****53****(8): 41–47.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1787234.1787249*](https://doi.org/10.1145%2F1787234.1787249)*.*
  5. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-Dijkstra1959_5-0) [***b***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-Dijkstra1959_5-1) [*Dijkstra, E. W.*](https://en.wikipedia.org/wiki/Edsger_W._Dijkstra)*(1959).*[*"A note on two problems in connexion with graphs"*](http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf)*(PDF). Numerische Mathematik.****1****: 269–271.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/BF01386390*](https://doi.org/10.1007%2FBF01386390)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*123284777*](https://api.semanticscholar.org/CorpusID:123284777)*.*
  6. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-mehlhorn_6-0) [***b***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-mehlhorn_6-1) [***c***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-mehlhorn_6-2) [***d***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-mehlhorn_6-3) [*Mehlhorn, Kurt*](https://en.wikipedia.org/wiki/Kurt_Mehlhorn)*;*[*Sanders, Peter*](https://en.wikipedia.org/wiki/Peter_Sanders_(computer_scientist))*(2008).*[*"Chapter 10. Shortest Paths"*](http://people.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/ShortestPaths.pdf)*(PDF). Algorithms and Data Structures: The Basic Toolbox. Springer.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/978-3-540-77978-0*](https://doi.org/10.1007%2F978-3-540-77978-0)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-3-540-77977-3*](https://en.wikipedia.org/wiki/Special:BookSources/978-3-540-77977-3)*.*
  7. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-Generic_Dijkstra_7-0) *Szcześniak, Ireneusz; Jajszczyk, Andrzej; Woźna-Szcześniak, Bożena (2019). "Generic Dijkstra for optical networks". Journal of Optical Communications and Networking.****11****(11): 568–577.*[*arXiv*](https://en.wikipedia.org/wiki/ArXiv_(identifier))*:*[*1810.04481*](https://arxiv.org/abs/1810.04481)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1364/JOCN.11.000568*](https://doi.org/10.1364%2FJOCN.11.000568)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*52958911*](https://api.semanticscholar.org/CorpusID:52958911)*.*
  8. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-felner_8-0) [***b***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-felner_8-1) [***c***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-felner_8-2) *Felner, Ariel (2011).*[*Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm*](http://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/view/4017/4357)*. Proc. 4th Int'l Symp. on Combinatorial Search.* In a route-finding problem, Felner finds that the queue can be a factor 500–600 smaller, taking some 40% of the running time.
  9. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-9) [*"ARMAC"*](https://web.archive.org/web/20131113021126/http:/www-set.win.tue.nl/UnsungHeroes/machines/armac.html)*. Unsung Heroes in Dutch Computing History. 2007. Archived from*[*the original*](http://www-set.win.tue.nl/UnsungHeroes/machines/armac.html)*on 13 November 2013.*
  10. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-EWD841a_10-0) *Dijkstra, Edsger W.,*[*Reflections on "A note on two problems in connexion with graphs*](https://www.cs.utexas.edu/users/EWD/ewd08xx/EWD841a.PDF)*(PDF)*
  11. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-11) [*Tarjan, Robert Endre*](https://en.wikipedia.org/wiki/Robert_Endre_Tarjan)*(1983), Data Structures and Network Algorithms, CBMS\_NSF Regional Conference Series in Applied Mathematics,****44****, Society for Industrial and Applied Mathematics, p. 75, The third classical minimum spanning tree algorithm was discovered by Jarník and rediscovered by Prim and Dikstra; it is commonly known as Prim's algorithm.*
  12. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-12) *Prim, R.C. (1957).*[*"Shortest connection networks and some generalizations"*](https://web.archive.org/web/20170718230207/http:/bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Prim1957.pdf)*(PDF). Bell System Technical Journal.****36****(6): 1389–1401.*[*Bibcode*](https://en.wikipedia.org/wiki/Bibcode_(identifier))*:*[*1957BSTJ...36.1389P*](https://ui.adsabs.harvard.edu/abs/1957BSTJ...36.1389P)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1002/j.1538-7305.1957.tb01515.x*](https://doi.org/10.1002%2Fj.1538-7305.1957.tb01515.x)*. Archived from*[*the original*](http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Prim1957.pdf)*(PDF) on 18 July 2017. Retrieved 18 July 2017.*
  13. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-13) V. Jarník: *O jistém problému minimálním* [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63. (in Czech)
  14. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-14) *Gass, Saul; Fu, Michael (2013). Gass, Saul I; Fu, Michael C (eds.). "Dijkstra's Algorithm". Encyclopedia of Operations Research and Management Science. Springer.****1****.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/978-1-4419-1153-7*](https://doi.org/10.1007%2F978-1-4419-1153-7)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-4419-1137-7*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-4419-1137-7)*– via Springer Link.*
  15. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-chen_07_15-0) *Chen, M.; Chowdhury, R. A.; Ramachandran, V.; Roche, D. L.; Tong, L. (2007).*[*Priority Queues and Dijkstra's Algorithm – UTCS Technical Report TR-07-54 – 12 October 2007*](http://www.cs.sunysb.edu/~rezaul/papers/TR-07-54.pdf)*(PDF). Austin, Texas: The University of Texas at Austin, Department of Computer Sciences.*
  16. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-aima_16-0) [***b***](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-aima_16-1) [*Russell, Stuart*](https://en.wikipedia.org/wiki/Stuart_J._Russell)*;*[*Norvig, Peter*](https://en.wikipedia.org/wiki/Peter_Norvig)*(2009) [1995].*[*Artificial Intelligence: A Modern Approach*](https://en.wikipedia.org/wiki/Artificial_Intelligence:_A_Modern_Approach)*(3rd ed.). Prentice Hall. pp. 75, 81.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-13-604259-4*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-13-604259-4)*.*
  17. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-17) Sometimes also *least-cost-first search*: *Nau, Dana S. (1983).*[*"Expert computer systems"*](https://www.cs.umd.edu/~nau/papers/nau1983expert.pdf)*(PDF). Computer. IEEE.****16****(2): 63–85.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/mc.1983.1654302*](https://doi.org/10.1109%2Fmc.1983.1654302)*.*
  18. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-speedup_18-0) *Wagner, Dorothea; Willhalm, Thomas (2007). Speed-up techniques for shortest-path computations. STACS. pp. 23–36.*
  19. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-19) *Bauer, Reinhard; Delling, Daniel; Sanders, Peter; Schieferdecker, Dennis; Schultes, Dominik; Wagner, Dorothea (2010).*[*"Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm"*](https://publikationen.bibliothek.kit.edu/1000014952)*. J. Experimental Algorithmics.****15****: 2.1.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/1671970.1671976*](https://doi.org/10.1145%2F1671970.1671976)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*1661292*](https://api.semanticscholar.org/CorpusID:1661292)*.*
  20. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-20) [*"Boost Graph Library: Directed Acyclic Graph Shortest Paths – 1.44.0"*](https://www.boost.org/doc/libs/1_44_0/libs/graph/doc/dag_shortest_paths.html)*. www.boost.org.*
  21. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-21) [Cormen et al. 2001](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFCormenLeisersonRivestStein2001), p. 655
  22. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-sniedovich_06_22-0) *Sniedovich, M. (2006).*[*"Dijkstra's algorithm revisited: the dynamic programming connexion"*](http://matwbn.icm.edu.pl/ksiazki/cc/cc35/cc3536.pdf)*(PDF). Journal of Control and Cybernetics.****35****(3): 599–620.* [Online version of the paper with interactive computational modules.](http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra_new/index.html)
  23. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-denardo_03_23-0) *Denardo, E.V. (2003). Dynamic Programming: Models and Applications. Mineola, NY:*[*Dover Publications*](https://en.wikipedia.org/wiki/Dover_Publications)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-486-42810-9*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-486-42810-9)*.*
  24. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-sniedovich_10_24-0) *Sniedovich, M. (2010). Dynamic Programming: Foundations and Principles.*[*Francis & Taylor*](https://en.wikipedia.org/w/index.php?title=Francis_%26_Taylor&action=edit&redlink=1)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-0-8247-4099-3*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-8247-4099-3)*.*
  25. [**^**](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_ref-25) [Dijkstra 1959](https://en.wikipedia.org/wiki/Dijkstra's_algorithm#CITEREFDijkstra1959), p. 270

References[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=14)]

* [*Cormen, Thomas H.*](https://en.wikipedia.org/wiki/Thomas_H._Cormen)*;*[*Leiserson, Charles E.*](https://en.wikipedia.org/wiki/Charles_E._Leiserson)*;*[*Rivest, Ronald L.*](https://en.wikipedia.org/wiki/Ronald_L._Rivest)*;*[*Stein, Clifford*](https://en.wikipedia.org/wiki/Clifford_Stein)*(2001). "Section 24.3: Dijkstra's algorithm".*[*Introduction to Algorithms*](https://en.wikipedia.org/wiki/Introduction_to_Algorithms)*(Second ed.).*[*MIT Press*](https://en.wikipedia.org/wiki/MIT_Press)*and*[*McGraw–Hill*](https://en.wikipedia.org/wiki/McGraw%E2%80%93Hill)*. pp. 595–601.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-262-03293-7*](https://en.wikipedia.org/wiki/Special:BookSources/0-262-03293-7)*.*
* *Dial, Robert B. (1969). "Algorithm 360: Shortest-path forest with topological ordering [H]".*[*Communications of the ACM*](https://en.wikipedia.org/wiki/Communications_of_the_ACM)*.****12****(11): 632–633.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/363269.363610*](https://doi.org/10.1145%2F363269.363610)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*6754003*](https://api.semanticscholar.org/CorpusID:6754003)*.*
* [*Fredman, Michael Lawrence*](https://en.wikipedia.org/wiki/Michael_Fredman)*;*[*Tarjan, Robert E.*](https://en.wikipedia.org/wiki/Robert_Tarjan)*(1984). Fibonacci heaps and their uses in improved network optimization algorithms. 25th Annual Symposium on Foundations of Computer Science.*[*IEEE*](https://en.wikipedia.org/wiki/IEEE)*. pp. 338–346.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/SFCS.1984.715934*](https://doi.org/10.1109%2FSFCS.1984.715934)*.*
* [*Fredman, Michael Lawrence*](https://en.wikipedia.org/wiki/Michael_Fredman)*;*[*Tarjan, Robert E.*](https://en.wikipedia.org/wiki/Robert_Tarjan)*(1987). "Fibonacci heaps and their uses in improved network optimization algorithms". Journal of the Association for Computing Machinery.****34****(3): 596–615.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/28869.28874*](https://doi.org/10.1145%2F28869.28874)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*7904683*](https://api.semanticscholar.org/CorpusID:7904683)*.*
* *Zhan, F. Benjamin; Noon, Charles E. (February 1998). "Shortest Path Algorithms: An Evaluation Using Real Road Networks".*[*Transportation Science*](https://en.wikipedia.org/wiki/Transportation_Science)*.****32****(1): 65–73.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1287/trsc.32.1.65*](https://doi.org/10.1287%2Ftrsc.32.1.65)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*14986297*](https://api.semanticscholar.org/CorpusID:14986297)*.*
* *Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, Jr., S. R.; Petry, R. M.; Seitz, R. N. (1957). Investigation of Model Techniques – First Annual Report – 6 June 1956 – 1 July 1957 – A Study of Model Techniques for Communication Systems. Cleveland, Ohio: Case Institute of Technology.*
* [*Knuth, D.E.*](https://en.wikipedia.org/wiki/Donald_Knuth)*(1977). "A Generalization of Dijkstra's Algorithm".*[*Information Processing Letters*](https://en.wikipedia.org/wiki/Information_Processing_Letters)*.****6****(1): 1–5.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1016/0020-0190(77)90002-3*](https://doi.org/10.1016%2F0020-0190%2877%2990002-3)*.*
* *Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990).*[*"Faster Algorithms for the Shortest Path Problem"*](https://dspace.mit.edu/bitstream/1721.1/47994/1/fasteralgorithms00sloa.pdf)*(PDF). Journal of the ACM.****37****(2): 213–223.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/77600.77615*](https://doi.org/10.1145%2F77600.77615)*.*[*hdl*](https://en.wikipedia.org/wiki/Hdl_(identifier))*:*[*1721.1/47994*](https://hdl.handle.net/1721.1%2F47994)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*5499589*](https://api.semanticscholar.org/CorpusID:5499589)*.*
* *Raman, Rajeev (1997). "Recent results on the single-source shortest paths problem". SIGACT News.****28****(2): 81–87.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/261342.261352*](https://doi.org/10.1145%2F261342.261352)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*18031586*](https://api.semanticscholar.org/CorpusID:18031586)*.*
* *Thorup, Mikkel (2000). "On RAM priority Queues". SIAM Journal on Computing.****30****(1): 86–109.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1137/S0097539795288246*](https://doi.org/10.1137%2FS0097539795288246)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*5221089*](https://api.semanticscholar.org/CorpusID:5221089)*.*
* *Thorup, Mikkel (1999).*[*"Undirected single-source shortest paths with positive integer weights in linear time"*](http://www.diku.dk/~mthorup/PAPERS/sssp.ps.gz)*. Journal of the ACM.****46****(3): 362–394.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/316542.316548*](https://doi.org/10.1145%2F316542.316548)*.*[*S2CID*](https://en.wikipedia.org/wiki/S2CID_(identifier))[*207654795*](https://api.semanticscholar.org/CorpusID:207654795)*.*

External links[[edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit&section=15)]

|  |  |
| --- | --- |
|  | Wikimedia Commons has media related to [***Dijkstra's algorithm***](https://commons.wikimedia.org/wiki/Category:Dijkstra%27s_algorithm). |

* [Oral history interview with Edsger W. Dijkstra](http://purl.umn.edu/107247), [Charles Babbage Institute](https://en.wikipedia.org/wiki/Charles_Babbage_Institute), University of Minnesota, Minneapolis
* [Implementation of Dijkstra's algorithm using TDD](http://blog.cleancoder.com/uncle-bob/2016/10/26/DijkstrasAlg.html), [Robert Cecil Martin](https://en.wikipedia.org/wiki/Robert_Cecil_Martin), The Clean Code Blog
* [Graphical explanation of Dijkstra's algorithm step-by-step on an example](http://www.gilles-bertrand.com/2014/03/disjkstra-algorithm-description-shortest-path-pseudo-code-data-structure-example-image.html), [Gilles Bertrand](https://en.wikipedia.org/w/index.php?title=Gilles_Bertrand&action=edit&redlink=1)

|  |  |
| --- | --- |
| show   * [v](https://en.wikipedia.org/wiki/Template:Edsger_Dijkstra) * [t](https://en.wikipedia.org/w/index.php?title=Template_talk:Edsger_Dijkstra&action=edit&redlink=1) * [e](https://en.wikipedia.org/w/index.php?title=Template:Edsger_Dijkstra&action=edit)   [**Edsger Dijkstra**](https://en.wikipedia.org/wiki/Edsger_W._Dijkstra) | |
| hide   * [v](https://en.wikipedia.org/wiki/Template:Optimization_algorithms) * [t](https://en.wikipedia.org/wiki/Template_talk:Optimization_algorithms) * [e](https://en.wikipedia.org/w/index.php?title=Template:Optimization_algorithms&action=edit)   [**Optimization**](https://en.wikipedia.org/wiki/Mathematical_optimization)**:**[**Algorithms**](https://en.wikipedia.org/wiki/Optimization_algorithm)**,**[**methods**](https://en.wikipedia.org/wiki/Iterative_method)**, and**[**heuristics**](https://en.wikipedia.org/wiki/Heuristic_algorithm) | | |
| |  | | --- | | show  [**Unconstrained nonlinear**](https://en.wikipedia.org/wiki/Nonlinear_programming) | | [Graph of a strictly concave quadratic function with unique maximum.](https://en.wikipedia.org/wiki/File:Max_paraboloid.svg) | |
| |  | | --- | | show  [**Constrained nonlinear**](https://en.wikipedia.org/wiki/Nonlinear_programming) | |
| |  | | --- | | show  [**Convex optimization**](https://en.wikipedia.org/wiki/Convex_optimization) | |
| |  | | --- | | show  [**Combinatorial**](https://en.wikipedia.org/wiki/Combinatorial_optimization) | |
| |  | | --- | | show  [**Metaheuristics**](https://en.wikipedia.org/wiki/Metaheuristic) | |
| * [Software](https://en.wikipedia.org/wiki/Comparison_of_optimization_software) | | |

[Categories](https://en.wikipedia.org/wiki/Help:Category):

* [Edsger W. Dijkstra](https://en.wikipedia.org/wiki/Category:Edsger_W._Dijkstra)
* [1959 in computing](https://en.wikipedia.org/wiki/Category:1959_in_computing)
* [Graph algorithms](https://en.wikipedia.org/wiki/Category:Graph_algorithms)
* [Search algorithms](https://en.wikipedia.org/wiki/Category:Search_algorithms)
* [Routing algorithms](https://en.wikipedia.org/wiki/Category:Routing_algorithms)
* [Combinatorial optimization](https://en.wikipedia.org/wiki/Category:Combinatorial_optimization)
* [Dutch inventions](https://en.wikipedia.org/wiki/Category:Dutch_inventions)

Navigation menu

* Not logged in
* [Talk](https://en.wikipedia.org/wiki/Special:MyTalk)
* [Contributions](https://en.wikipedia.org/wiki/Special:MyContributions)
* [Create account](https://en.wikipedia.org/w/index.php?title=Special:CreateAccount&returnto=Dijkstra%27s+algorithm)
* [Log in](https://en.wikipedia.org/w/index.php?title=Special:UserLogin&returnto=Dijkstra%27s+algorithm)
* [Article](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)
* [Talk](https://en.wikipedia.org/wiki/Talk:Dijkstra%27s_algorithm)
* [Read](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)
* [Edit](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=edit)
* [View history](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=history)

**Search**

Top of Form



Bottom of Form

* [Main page](https://en.wikipedia.org/wiki/Main_Page)
* [Contents](https://en.wikipedia.org/wiki/Wikipedia:Contents)
* [Current events](https://en.wikipedia.org/wiki/Portal:Current_events)
* [Random article](https://en.wikipedia.org/wiki/Special:Random)
* [About Wikipedia](https://en.wikipedia.org/wiki/Wikipedia:About)
* [Contact us](https://en.wikipedia.org/wiki/Wikipedia:Contact_us)
* [Donate](https://donate.wikimedia.org/wiki/Special:FundraiserRedirector?utm_source=donate&utm_medium=sidebar&utm_campaign=C13_en.wikipedia.org&uselang=en)

Contribute

* [Help](https://en.wikipedia.org/wiki/Help:Contents)
* [Community portal](https://en.wikipedia.org/wiki/Wikipedia:Community_portal)
* [Recent changes](https://en.wikipedia.org/wiki/Special:RecentChanges)
* [Upload file](https://en.wikipedia.org/wiki/Wikipedia:File_Upload_Wizard)

Tools

* [What links here](https://en.wikipedia.org/wiki/Special:WhatLinksHere/Dijkstra%27s_algorithm)
* [Related changes](https://en.wikipedia.org/wiki/Special:RecentChangesLinked/Dijkstra%27s_algorithm)
* [Special pages](https://en.wikipedia.org/wiki/Special:SpecialPages)
* [Permanent link](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&oldid=976902372)
* [Page information](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&action=info)
* [Cite this page](https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Dijkstra%27s_algorithm&id=976902372&wpFormIdentifier=titleform)
* [Wikidata item](https://www.wikidata.org/wiki/Special:EntityPage/Q8548)

Print/export

* [Download as PDF](https://en.wikipedia.org/w/index.php?title=Special:DownloadAsPdf&page=Dijkstra%27s_algorithm&action=show-download-screen)
* [Printable version](https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&printable=yes)

In other projects

* [Wikimedia Commons](https://commons.wikimedia.org/wiki/Category:Dijkstra%27s_algorithm)

Languages

* [العربية](https://ar.wikipedia.org/wiki/%D8%AE%D9%88%D8%A7%D8%B1%D8%B2%D9%85%D9%8A%D8%A9_%D8%AF%D9%8A%D9%83%D8%B3%D8%AA%D8%B1%D8%A7)
* [Español](https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra)
* [فارسی](https://fa.wikipedia.org/wiki/%D8%A7%D9%84%DA%AF%D9%88%D8%B1%DB%8C%D8%AA%D9%85_%D8%AF%DA%A9%D8%B3%D8%AA%D8%B1%D8%A7)
* [Français](https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra)
* [हिन्दी](https://hi.wikipedia.org/wiki/%E0%A4%A1%E0%A4%BF%E0%A4%9C%E0%A4%95%E0%A5%8D%E0%A4%B8%E0%A5%8D%E0%A4%9F%E0%A5%8D%E0%A4%B0%E0%A4%BE_%E0%A4%95%E0%A4%BE_%E0%A4%85%E0%A4%B2%E0%A5%8D%E0%A4%97%E0%A5%8B%E0%A4%B0%E0%A4%BF%E0%A4%A6%E0%A5%8D%E0%A4%AE)
* [Bahasa Indonesia](https://id.wikipedia.org/wiki/Algoritme_Dijkstra)
* [Português](https://pt.wikipedia.org/wiki/Algoritmo_de_Dijkstra)
* [Русский](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B)
* [中文](https://zh.wikipedia.org/wiki/%E6%88%B4%E5%85%8B%E6%96%AF%E7%89%B9%E6%8B%89%E7%AE%97%E6%B3%95)

37 more

[Edit links](https://www.wikidata.org/wiki/Special:EntityPage/Q8548#sitelinks-wikipedia)

* This page was last edited on 5 September 2020, at 19:09 (UTC).
* Text is available under the [Creative Commons Attribution-ShareAlike License](https://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License); additional terms may apply. By using this site, you agree to the [Terms of Use](https://foundation.wikimedia.org/wiki/Terms_of_Use) and [Privacy Policy](https://foundation.wikimedia.org/wiki/Privacy_policy). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](https://www.wikimediafoundation.org/), a non-profit organization.