```
private boolean[] marked;
private int[] edgeTo;
private void bfs(Graph G, int s) {
  Queue<Integer> fringe = ...;
  fringe.add(s);
  marked[s] = true;
  while (!fringe.isEmpty()) {
    int v = fringe.remove();
    for (int w : G.adj(v)) {
      if (!marked[w]) {
        fringe.add(w);
        marked[w] = true;
        edgeTo[w] = v;
      }
    }
  }
}
```

BreadthFirstPaths

Demo

Instance variables store algorithm data.

- marked[**v**] is true iff **v** connected to **s**.
- edgeTo[**v**] is vertex visited to get to **v**.

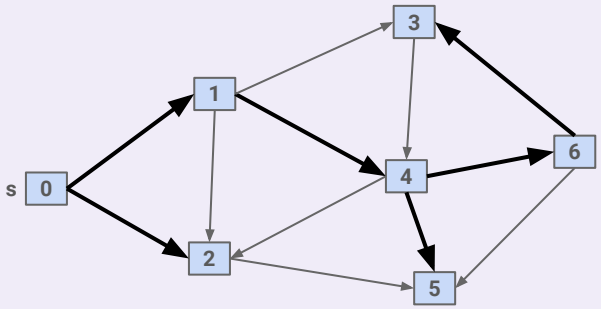
BreadthFirstPaths constructor computes the result of the algorithm with the bfs iterative method.

Cost model given undirected graph?

Each vertex is visited at most once.
Each edge is checked at most twice.

Q Shortest Paths Tree

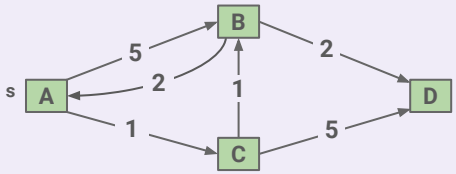
If **G** is a connected edge-weighted graph with **V** vertices and **E** edges, how many edges are in the **Shortest Paths Tree** (SPT) of G? Assume every vertex is reachable.



v	distTo[]	edgeTo[]
0	0.0	-
1	2.0	0 -> 1
2	1.0	0 -> 2
3	11.0	6 -> 3
4	5.0	1 -> 4
5	9.0	4 -> 5
6	10.0	4 -> 6

Q Finding a Shortest Paths Tree

What is the shortest paths tree for the graph below starting from the source vertex **A**?

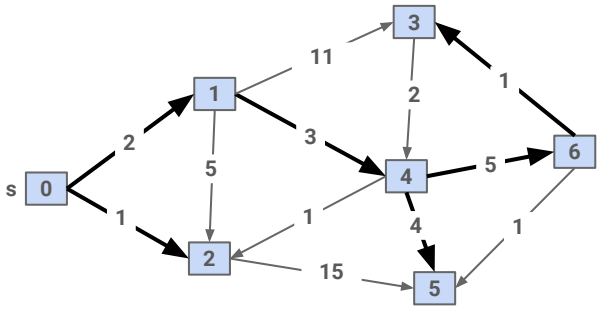


Dijkstra's Algorithm

Demo

Insert all vertices into **fringe** PQ, storing vertices in order of distance from source.

While fringe is not empty: Remove (closest) vertex **v** and **relax** all edges pointing from **v**.



Edge Relaxation (**v**, **w**).

For each edge (**v**, **w**), add edge to the SPT **only if the edge is closer than our best-so-far**.

PQ.add(**s**, 0)

For all other vertices **v**, PQ.add(**v**, infinity)

While PQ is not empty:

    p = PQ.removeSmallest()
    Relax all edges from **p**

Relaxing an edge (**v**, **w**) with **weight**:

If distTo[**w**] > distTo[**v**] + **weight**:

    distTo[**w**] = distTo[**v**] + **weight**
    edgeTo[**w**] = **v**
    PQ.changePriority(**w**, distTo[**w**])

Dijkstra's Pseudocode

Invariants

edgeTo[**v**]: best known predecessor of **v**.

distTo[**v**]: best known distance of **s** to **v**.

PQ maintains vertices based on distTo.

Important properties

Always visits vertices in order of total distance from source. Relaxation always fails on edges to visited (white) vertices.

Q Dijkstra's Algorithm Correctness

Dijkstra's algorithm. Visit vertices in order of distance from source.

    On visit, **relax** every edge from the visited vertex.

Dijkstra's can fail if the graph has negative weight edges. **Give an example graph.**

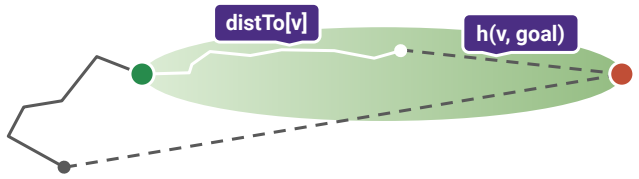
A* Search Algorithm

Demo

Dijkstra's algorithm with one modification.

- Dijkstra's algorithm: Priority is defined by distTo[**v**] only.
- **A* search**: Priority is defined by distTo[**v**] + h(**v**, **goal**).

Where h(**v**, **goal**) is a **heuristic**: an estimate of the distance from **v** to the **goal**.



Q Computing a Heuristic

Where h(**v**, **goal**) is a **heuristic**: an estimate of the distance from **v** to the **goal**.

For maps, we can use Euclidean distance (right triangle hypotenuse length).

Will A* search return the correct shortest path if h(**v**, **goal**) = **10** for every **v** in the graph?