Shortest Paths
Part One
Recap from Last Time
A *graph* is a mathematical structure for representing relationships.
A *graph* is a mathematical structure for representing relationships.

A graph consists of a set of *nodes* connected by *edges*.
A graph is a mathematical structure for representing relationships.

A graph consists of a set of nodes connected by edges.
A graph is a mathematical structure for representing relationships. A graph consists of a set of nodes connected by edges.
Breadth-First Search
Breadth-First Search

CAN - MAN - RAN

CAT - SAT - RAT

MAT
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

**Gray** nodes are unexplored.

**Yellow** nodes are in the queue.

**Green** nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
Breadth-First Search

Gray nodes are unexplored.

Yellow nodes are in the queue.

Green nodes have had the best path discovered.
**BFS Pseudocode**

```plaintext
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}
```
The Limits of Breadth-First Search
Each intersection is a node.

Edges represent roads.

Different roads have different lengths.

**Question:** What’s the best way to get from point A to point B?
The Model

- We have a graph in which each edge has a nonnegative *cost* or *weight* associated with it.
- We want to find the lowest-cost path from point A to point B.
- BFS does not take edge weights into account.
- How might we go about solving this problem?
Option 1: Brute-Force!

- We could conceivably solve this problem using brute force and a backtracking recursion.
- **Problem:** There can be a lot of different paths in a graph!
- This is way too inefficient to use in practice.
Option 2: Expand the Graph

- BFS works in the case where each edge has equal weight.

- **Idea:** What if we split each edge of length $k$ into $k$ smaller edges?
Option 2: Expand the Graph

- BFS works in the case where each edge has equal weight.
- **Idea:** What if we split each edge of length $k$ into $k$ smaller edges?
Option 2: Expand the Graph

- BFS works in the case where each edge has equal weight.
- **Idea:** What if we split each edge of length $k$ into $k$ smaller edges?
- What if there are fractional edges? Or large weights?
Option 2: Expand the Graph

- BFS works in the case where each edge has equal weight.
- **Idea:** What if we split each edge of length $k$ into $k$ smaller edges?
- What if there are fractional edges? Or large weights?
Option 3: Look at the problem more closely
Observation: The shortest path from A to F can’t start by going directly from A to D.
We have a valid path that goes from A to B, and no alternate path could possibly be as good. *This must be the shortest path!*
The Pattern

All yellow nodes (nodes we've seen, but don't know the distance to.)
The Pattern

Look at the lowest-cost yellow node.
The Pattern
The Pattern

[Diagram of a network with nodes labeled 106?, 137?, 271?, and 161?]
The Pattern
The Pattern

No other path to this node can be better than the one we already know about!
The Pattern

106?
137?
271?
161?
At a Glance

- The approach suggested here gives rise to **Dijkstra’s algorithm**, a fast, powerful, and famous algorithm for computing shortest paths.

- **Key idea:** As in BFS, split nodes into
  - **gray nodes** we haven’t seen,
  - **yellow nodes** that are on the frontier, and
  - **green nodes** we have the best path to,

then repeatedly turn the lowest-cost yellow node into a green node.
Implementing Dijkstra’s Algorithm
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.

        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don't use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstras-algorithm() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ▲ Don’t use this as a reference!
breadth-first-search() {
   make a queue of nodes.
   enqueue start node.
   color the start node yellow.

   while (the queue is not empty) {
      dequeue a node from the queue.
      color that node green.

      for (each neighboring node) {
         if (that node is gray) {
            color the node yellow.
            enqueue it.
         }
      }
   }
}

dijkstra's algorithm() {
   make a queue of nodes.
   enqueue start node.
   color the start node yellow.

   while (the queue is not empty) {
      dequeue a node from the queue.
      color that node green.

      for (each neighboring node) {
         if (that node is gray) {
            color the node yellow.
            enqueue it.
         }
      }
   }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {  
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.

    color that node green.

    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

Dijkstra's algorithm() {  
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.

    color that node green.

    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don't use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's algorithm() {
    make a priority queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!


**Breadth-First Search**

- Make a queue of nodes.
- Enqueue the start node.
- Color the start node yellow.

while (the queue is not empty) {
  Dequeue a node from the queue.
  Color that node green.
  For (each neighboring node) {
    If (that node is gray) {
      Color the node yellow.
      Enqueue it.
    }
  }
}

**Dijkstra's Algorithm**

- Make a priority queue of nodes.
- Enqueue the start node at distance 0.
- Color the start node yellow.

while (the queue is not empty) {
  Dequeue a node from the queue.
  Color that node green.
  For (each neighboring node) {
    If (that node is gray) {
      Color the node yellow.
      Enqueue it.
    }
  }
}

⚠️ Still under construction! ⚠️

Don't use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don't use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}  
⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}
⚠ Still under construction! △
Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.

        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.

        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

⚠ Still under construction!⚠
Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.

    color that node green.

    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.

    color that node green.

    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
enqueue start node.
color the start node yellow.

while (the queue is not empty) {
    dequeue a node from the queue.
color that node green.

    for (each neighboring node) {
        if (that node is gray) {
            color the node yellow.
enqueue it.
        }
    }
}
}
dijkstra's-algorithm() {
    make a priority queue of nodes.
enqueue start node at distance 0.
color the start node yellow.

while (the queue is not empty) {
    dequeue a node from the queue.
color that node green.

    for (each neighboring node) {
        if (that node is gray) {
            color the node yellow.
enqueue it at the new distance.
        }
    }
}
⚠ Still under construction! ⚠
Don't use this as a reference!
`breadth-first-search()` {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

`dijkstra's-algorithm()` {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it at the new distance.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.

    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.

    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it at the new distance.
      }
    }
  }

  △ Still under construction! △
  Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it at the new distance.
      }
    }
  }
 ⚠ Still under construction! ⚠
  Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it at the new distance.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
} ⚠ Still under construction! ▲ Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! △ Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it at the new distance.
      }
    }
  }
}  

⚠ Still under construction! ⚠
Don't use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}  

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}  

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
enqueue start node.
color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
enqueue start node at distance 0.
color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
enqueue start node.
color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
enqueue start node at distance 0.
color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
enqueue it at the new distance.
            }
        }
    }
}
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }

}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }

}⚠ Still under construction!⚠ Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}
**breadth-first-search()**

- make a queue of nodes.
- enqueue start node.
- color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }

**dijkstra's-algorithm()**

- make a priority queue of nodes.
- enqueue start node at distance 0.
- color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }

⚠️ Still under construction! ⚠️
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
  make a queue of nodes.
  enqueue start node.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is gray) {
        color the node yellow.
        enqueue it.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!

Dijkstra's-algorithm() {
  make a priority queue of nodes.
  enqueue start node at distance 0.
  color the start node yellow.

  while (the queue is not empty) {
    dequeue a node from the queue.
    color that node green.
    for (each neighboring node) {
      if (that node is not green) {
        color the node yellow.
        enqueue it at the new distance.
      }
    }
  }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is not green) {
                color the node yellow.
                enqueue it at the new distance.
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.
            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
}

⚠ Still under construction! ⚠
Don’t use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.
        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.
            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
} ⚠ Still under construction! ⚠

Don't use this as a reference!
breadth-first-search() {
    make a queue of nodes.
    enqueue start node.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        color that node green.

        for (each neighboring node) {
            if (that node is gray) {
                color the node yellow.
                enqueue it.
            }
        }
    }
}

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.

            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
}
⚠ Still under construction! ⚠
Don't use this as a reference!
The Finished Product

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.

            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
}
The Finished Product

dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.

            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
}
dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.

            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
}
dijkstra's-algorithm() {
    make a priority queue of nodes.
    enqueue start node at distance 0.
    color the start node yellow.

    while (the queue is not empty) {
        dequeue a node from the queue.
        if (that node isn't green) {
            color that node green.
            for (each neighboring node) {
                if (that node is not green) {
                    color the node yellow.
                    enqueue it at the new distance.
                }
            }
        }
    }
}