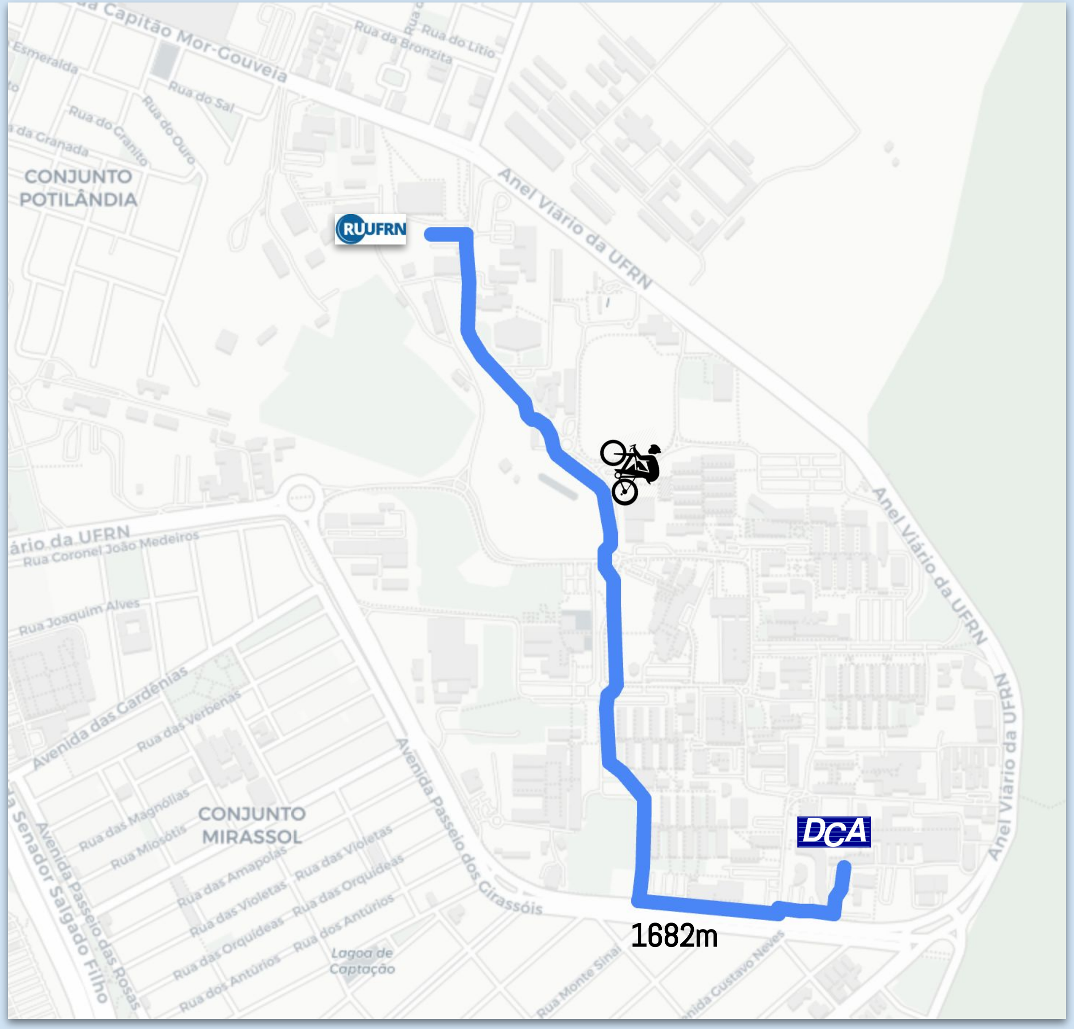
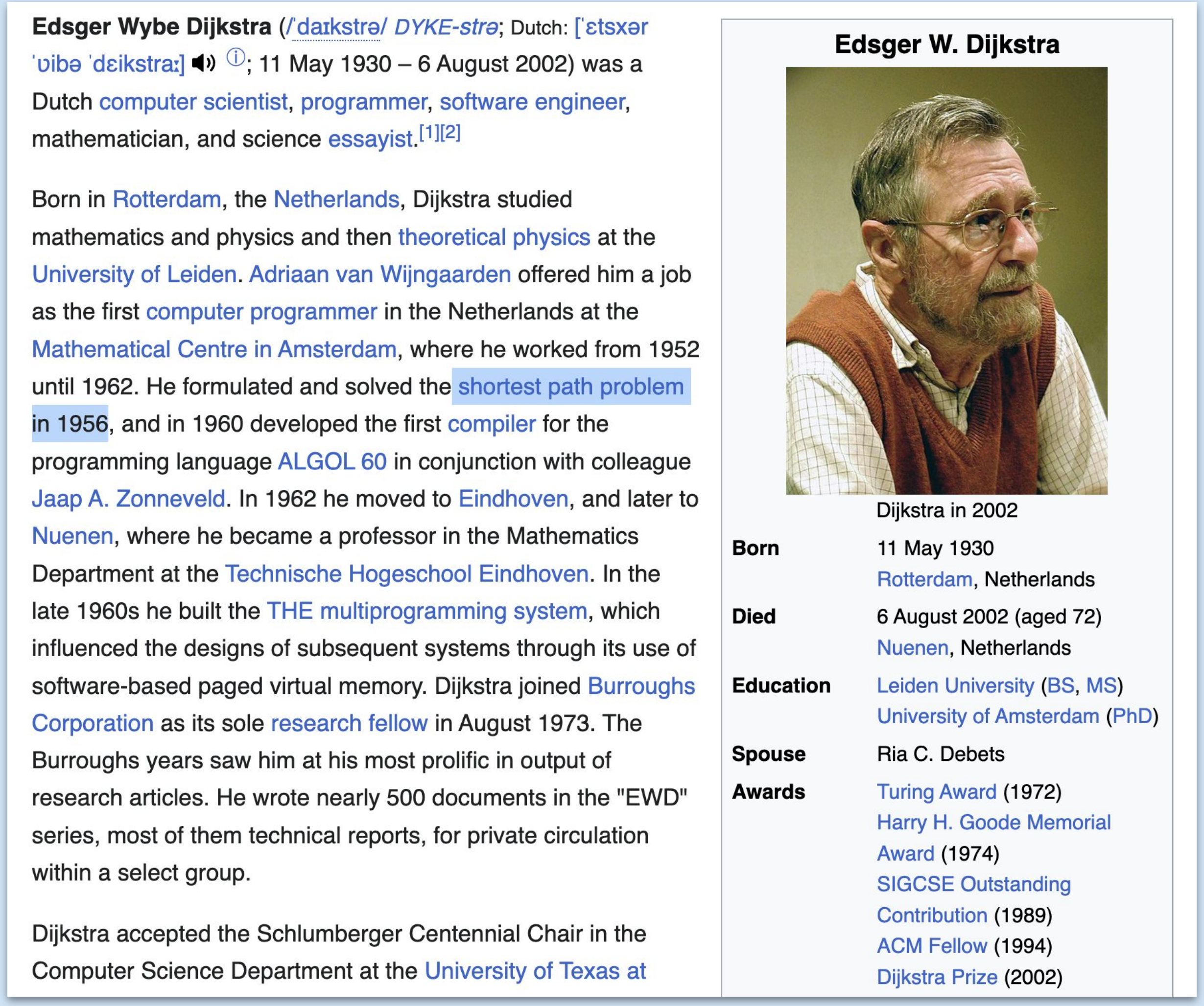
Classical Algorithms

Shortest Path

ivanovitch.silva@ufrn.br
@ivanovitchm



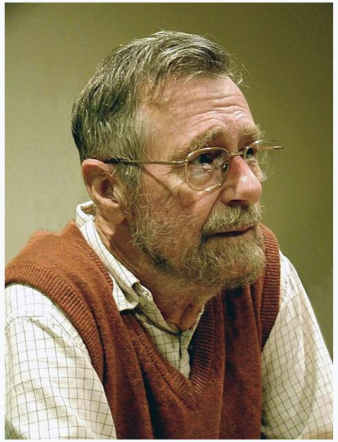


Edsger Wybe Dijkstra (/ˈdaɪkstrə/ *DYKE-strə*; Dutch: [ˈɛtsxər ˈʋibə ˈdɛikstraː] 🔊 ⓘ; 11 May 1930 – 6 August 2002) was a Dutch computer scientist, programmer, software engineer, mathematician, and science essayist.[1][2]

Born in Rotterdam, the Netherlands, Dijkstra studied mathematics and physics and then theoretical physics at the University of Leiden. Adriaan van Wijngaarden offered him a job as the first computer programmer in the Netherlands at the Mathematical Centre in Amsterdam, where he worked from 1952 until 1962. He formulated and solved the shortest path problem in 1956, and in 1960 developed the first compiler for the programming language ALGOL 60 in conjunction with colleague Jaap A. Zonneveld. In 1962 he moved to Eindhoven, and later to Nuenen, where he became a professor in the Mathematics Department at the Technische Hogeschool Eindhoven. In the late 1960s he built the THE multiprogramming system, which influenced the designs of subsequent systems through its use of software-based paged virtual memory. Dijkstra joined Burroughs Corporation as its sole research fellow in August 1973. The Burroughs years saw him at his most prolific in output of research articles. He wrote nearly 500 documents in the "EWD" series, most of them technical reports, for private circulation within a select group.

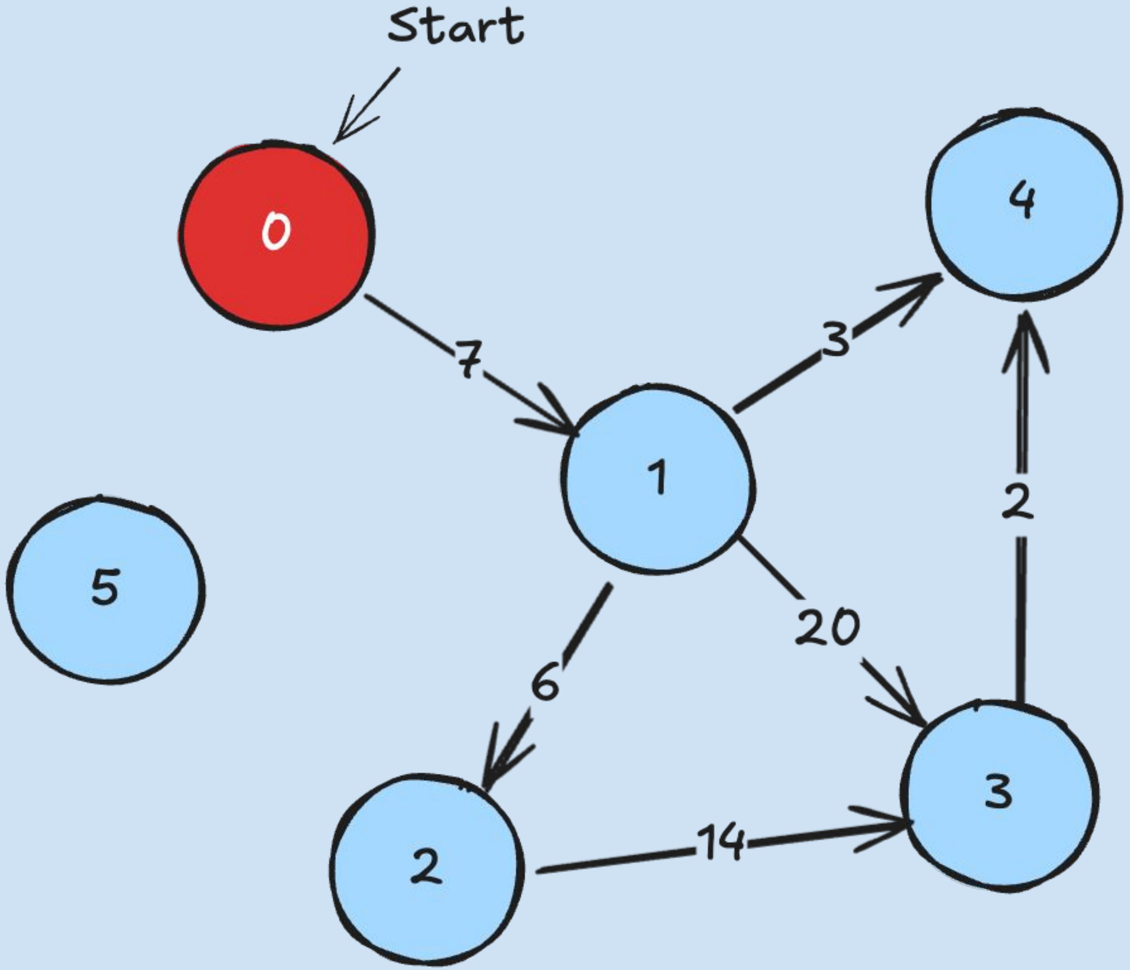
Dijkstra accepted the Schlumberger Centennial Chair in the Computer Science Department at the University of Texas at

Edsger W. Dijkstra



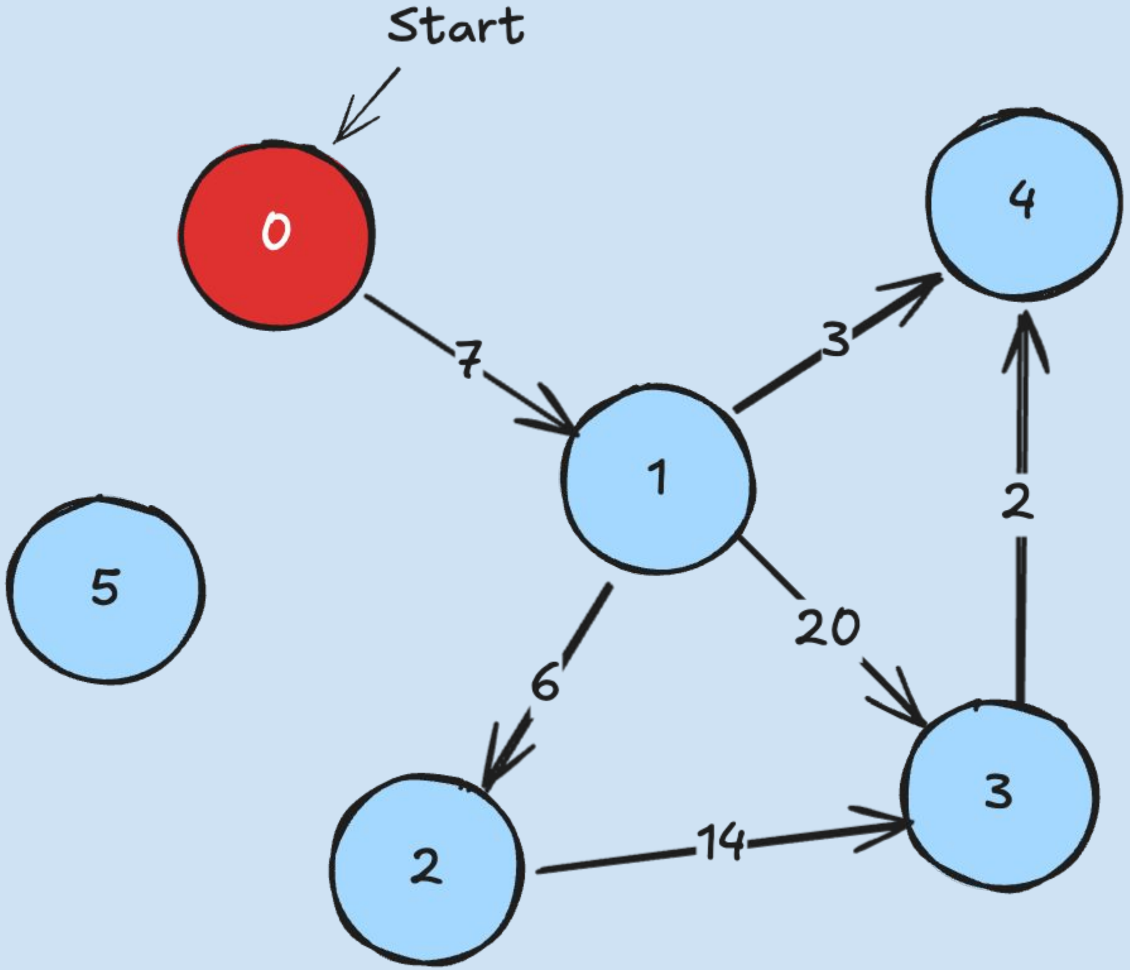
Dijkstra in 2002

Born	11 May 1930 Rotterdam, Netherlands
Died	6 August 2002 (aged 72) Nuenen, Netherlands
Education	Leiden University (BS, MS) University of Amsterdam (PhD)
Spouse	Ria C. Debets
Awards	Turing Award (1972) Harry H. Goode Memorial Award (1974) SIGCSE Outstanding Contribution (1989) ACM Fellow (1994) Dijkstra Prize (2002)



Dijkstra's algorithm is a technique used to solve the shortest path problem in **weighted** and **directed graphs**. It calculates the shortest distance from the starting node (referred to as "start") to all other nodes in the graph.

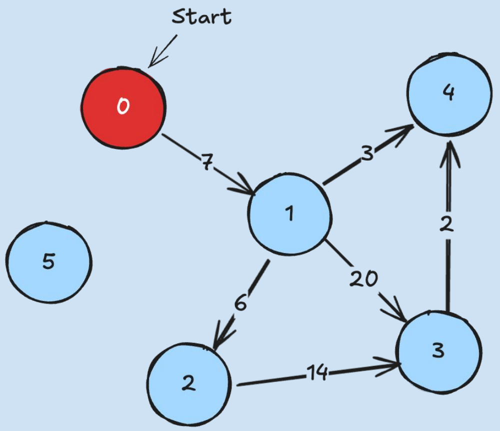
It does not work with negative cycles (i.e., edges with negative weights).

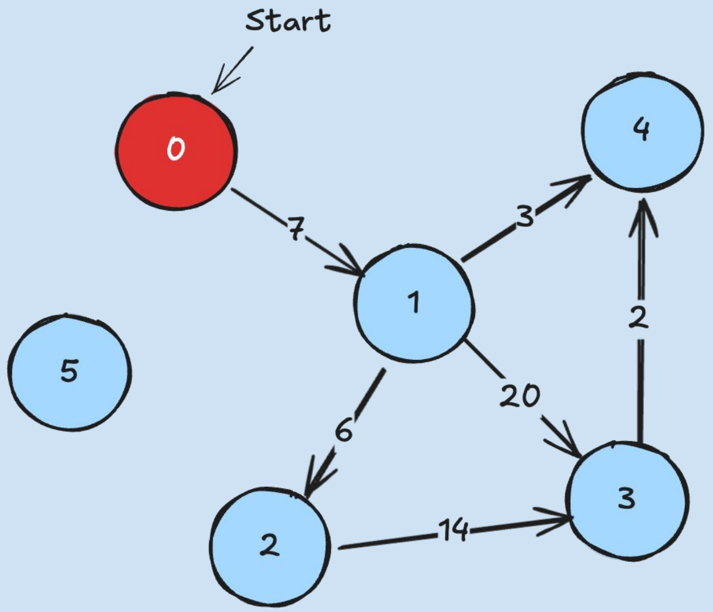


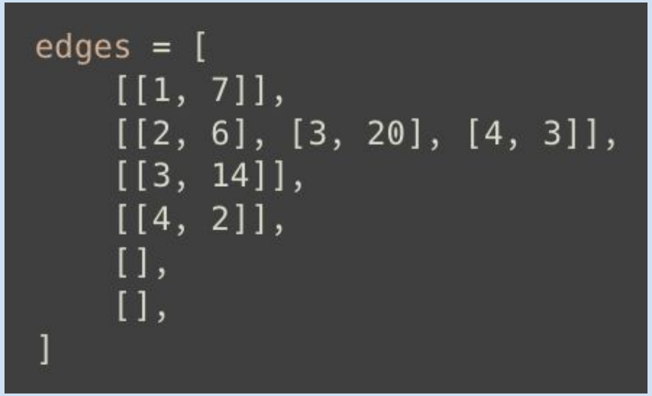
```
edges = [
    [[1, 7]],
    [[2, 6], [3, 20], [4, 3]],
    [[3, 14]],
    [[4, 2]],
    [],
    [],
]
```

How It Works?

- **Initialization**:
  - Set the distance of the starting node to itself as 0.
  - Set the distance of all other nodes to "infinity" (inf).
- **Choose the Next Node**:
  - Always select the node with the smallest known distance that hasn't been processed yet.
- **Update Distances**:
  - Check the neighbors of the current node.
  - Update the distances of neighbors if a shorter path is found.
- **Repeat**:
  - Mark the current node as "visited" and repeat the process for the next node with the smallest distance.
- **Finalize**:
  - Once all nodes are visited, the algorithm returns the shortest distances or calculated paths.







```
edges = [
    [[1, 7]],
    [[2, 6], [3, 20], [4, 3]],
    [[3, 14]],
    [[4, 2]],
    [],
    [],
]
```

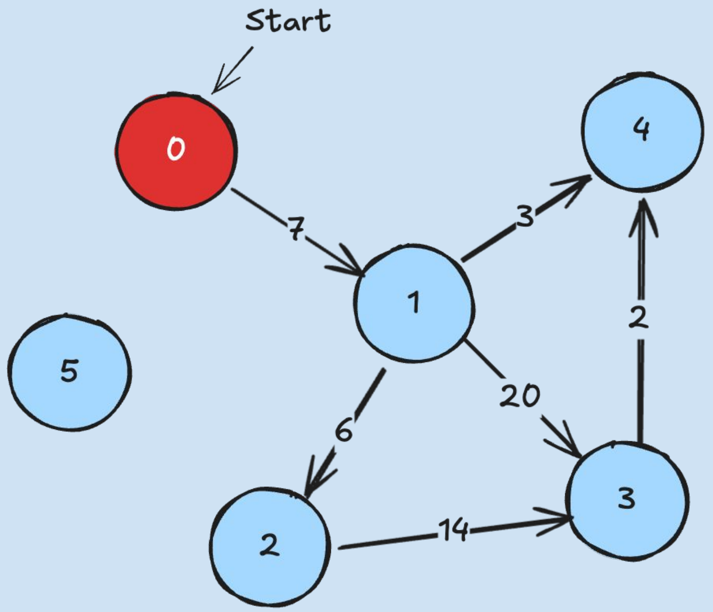
Main function: dijkstrasAlgorithm(start, edges) return list

[0, 7, 13, 27, 10, -1]

Aux function: getVertexWithMinDistance(distances, visited)
return vertex (not visited), currentMinDistance

Aux variables: minDistances = [inf, inf, inf, inf, inf, inf]

visited = set()



Iteration #1

Node Selection

- Starting node: start = 0.
- Current distance: minDistances[0] = 0.

```
minDistances = [0, inf, inf, inf, inf, inf]
```

Neighbor Updates

- Node 0 has an edge to node 1 with weight 7.
- Calculate the new distance to node 1:

```
newPathDistance = minDistances[0] + 7 = 0 + 7 = 7
```

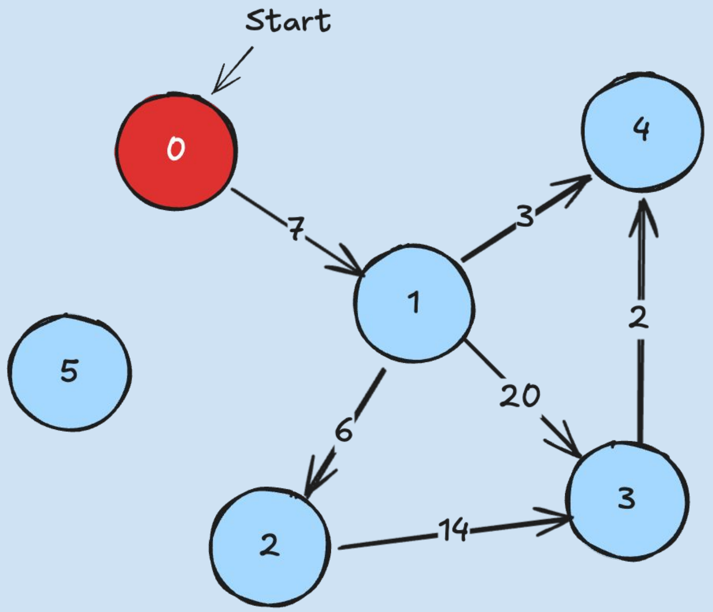
- Update `minDistances[1]`:

```
minDistances = [0, 7, inf, inf, inf, inf]
```

Update `visited`

- Add node `0` to the set:

```
visited = {0}
```



Iteration #2

Node Selection

- Node 1 is selected (smallest distance among unvisited nodes, 7).

minDistances = [0, 7, inf, inf, inf, inf]

Neighbor Updates

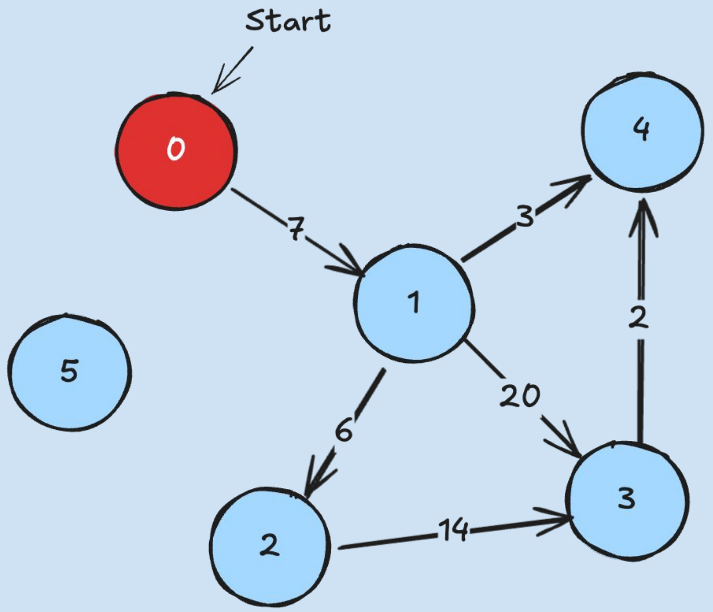
- Node 1 has edges to:
  - Node 2 with weight 6:
    newPathDistance = 7 + 6 = 13
  - minDistances[2] = 13
- Node 3 with weight 20:
  - newPathDistance = 7 + 20 = 27
  - minDistances[3] = 27
- Node 4 with weight 3:
  - newPathDistance = 7 + 3 = 10
  - minDistances[4] = 10

Update minDistances

minDistances = [0, 7, 13, 27, 10, inf]

Update visited

visited = {0,1}



Node Selection

- Node 4 is selected (smallest distance among unvisited nodes, 10).

minDistances = [0, 7, 13, 27, 10, inf]

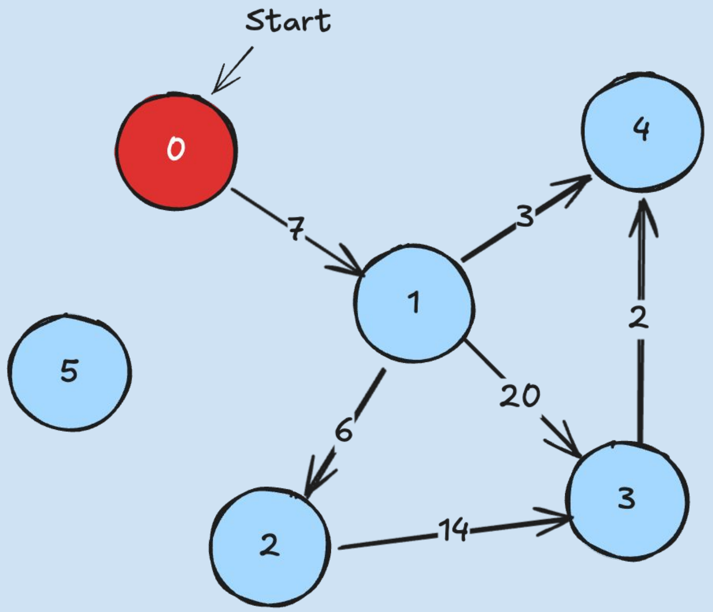
Neighbor Updates

- Node 4 has no outgoing edges, so no distances are updated.

Update visited

visited = {0,1,4}

Iteration #3



Iteration #4

Node Selection

- Node 2 is selected (smallest distance among unvisited nodes, 10).

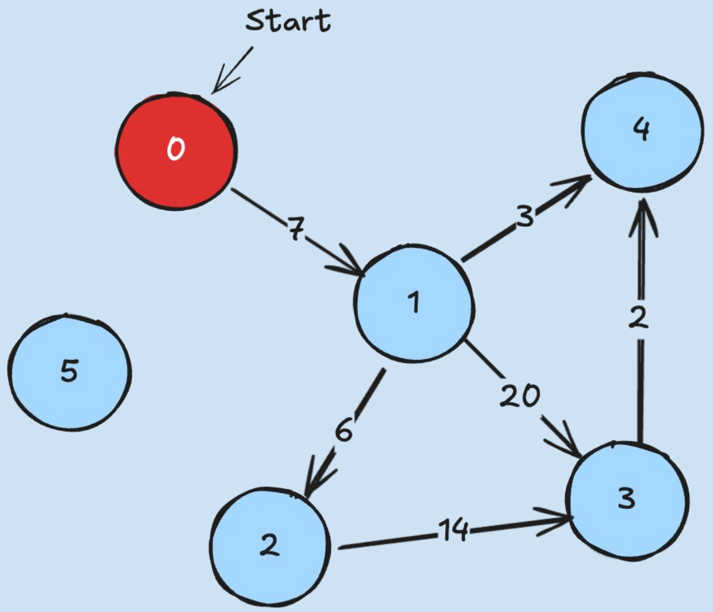
minDistances = [0, 7, 13, 27, 10, inf]

Neighbor Updates

- Node 2 has edge to:
  - Node 3 with weight 14:
    newPathDistance = 13 + 14 = 27
  - minDistances[3] is not updated, as the current value (27) is equal.

Update visited

visited = {0,1,4,2}



Node Selection

- Node 3 is selected (smallest distance among unvisited nodes, 27).

minDistances = [0, 7, 13, 27, 10, inf]

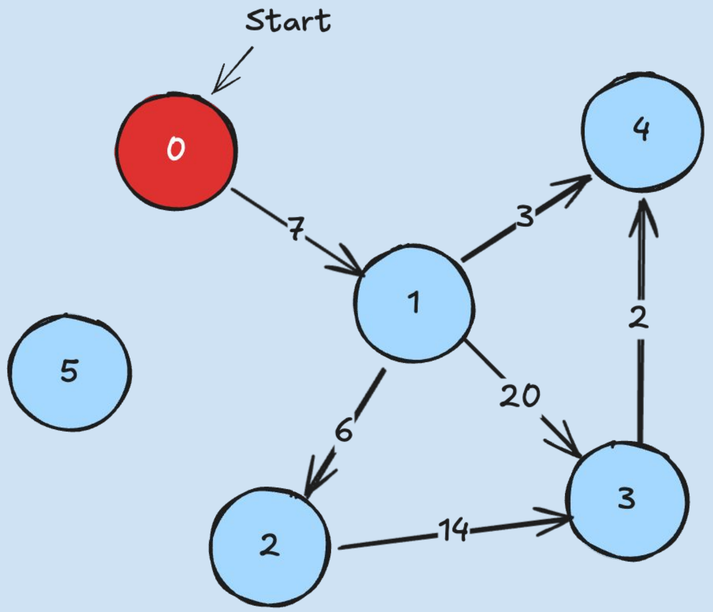
Neighbor Updates

- Node 3 has edge to:
  - Node 4
  - Node 4 has already been visited, so no update is needed.

Update visited

visited = {0,1,4,2,3}

Iteration #5



Final result

- Replace **inf** with **-1** for nodes that are unreachable.

```
minDistances = [0, 7, 13, 27, 10, -1]
```

This represents the shortest distances from the starting **node (0)** to all other nodes.

Iteration #6

How about the complexity?

$O(V^2+E)$ time

$O(V)$ space

1. **Finding the vertex with the minimum distance:**
   In each main iteration, the algorithm searches for the unvisited node with the smallest known distance. This search scans through all vertices (not yet visited) and can take up to O(V) time per iteration, where V is the total number of vertices.

2. **Repeating the main iteration for all vertices:**
   Since you need to select the minimum-distance vertex V times (once for each vertex), you repeat this O(V)-time search V times, resulting in O(V) * O(V) = O(V^2).

3. **Relaxing the edges:**
   After selecting a vertex, you iterate through its outgoing edges to possibly update the distances of its neighbors. Over the entire run of the algorithm, each edge is considered at least once, adding up to O(E), where E is the total number of edges.

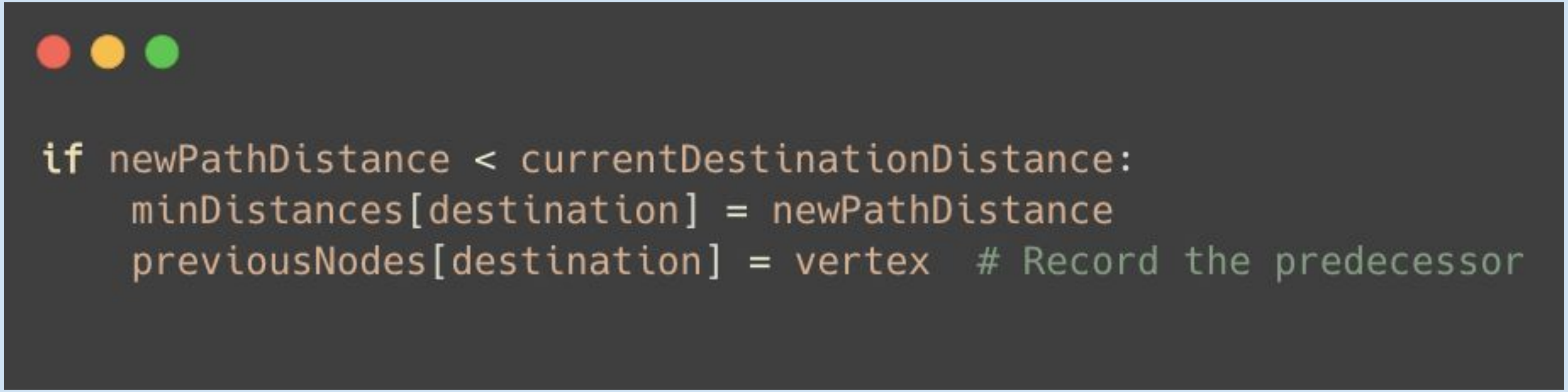
How can we improve it?

$O(V+E)*\log(V)$ time
$O(V)$ space

How to find the path with Dijkstra's Algorithm?

Finding the Path with Dijkstra's Algorithm

To find the **actual path** (sequence of nodes) from the starting node to any other node in the graph, you need to keep track of the **predecessors** of each node during the execution of the algorithm. This can be achieved by adding a list called previousNodes, which stores the previous node for each node in the shortest path.



```
if newPathDistance < currentDestinationDistance:
    minDistances[destination] = newPathDistance
    previousNodes[destination] = vertex  # Record the predecessor
```

Finding the Path with Dijkstra's Algorithm

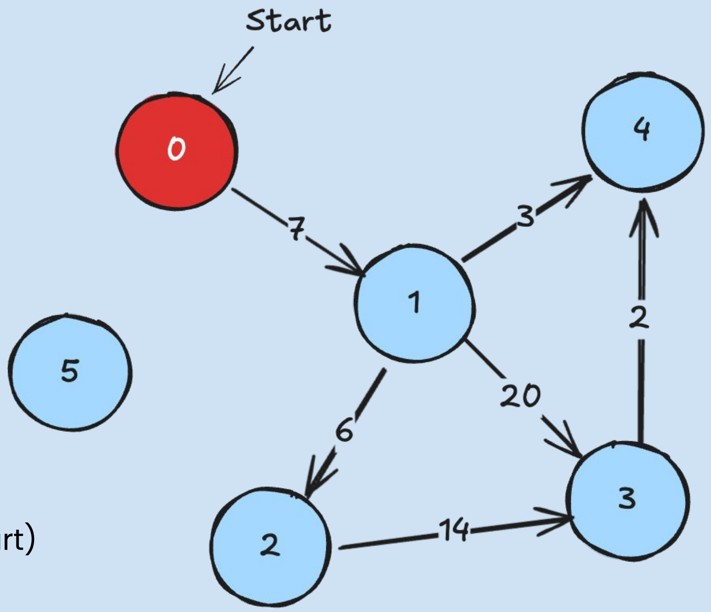
Example with Final Result

Algorithm Outputs:

- minDistances = [0, 7, 13, 27, 10, -1]
- previousNodes = [None, 0, 1, 2, 1, None]

Finding the Path from 0 to 3:

1. Start with node 3 and follow the predecessors:
    - currentNode = 3 → predecessor: 2
    - currentNode = 2 → predecessor: 1
    - currentNode = 1 → predecessor: 0
    - currentNode = 0 → predecessor: None (reached the start)
2. The path traced backward is [3, 2, 1, 0].
3. Reverse the path: [0, 1, 2, 3].


Start
0
1
2
3
4
5
7
3
2
20
6
14