Distances

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

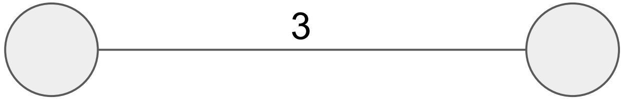
(Bad Idea) Split the edges into nodes based on their length.

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

(Bad Idea) Split the edges into nodes based on their length.



Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

(Bad Idea) Split the edges into nodes based on their length.

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

(Bad Idea) Split the edges into nodes based on their length.

Runtime is now dependent on edge weights. (yikes!)

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

(Bad Idea) Split the edges into nodes based on their length.

- Runtime is now dependent on edge weights. (yikes!)
- How would 0 or negative edges or fractional edge weights be handled?

Unit Distance Graph

How would we be able to find the shortest distance between a pair of nodes in a unit distance graph?

What if the graph was no longer unit distance?

(Bad Idea) Split the edges into nodes based on their length.

Runtime is now dependent on edge weights. (yikes!)

How would 0 or negative edges or fractional edge weights be handled?

Luckily there are three other algorithms that can be helpful.

Bellman-Ford-Moore

Most of the time called Bellman-Ford.

Bellman-Ford-Moore

Most of the time called Bellman-Ford.

Determines the distance between one node and all other nodes.

Bellman-Ford-Moore

Most of the time called Bellman-Ford.

Determines the distance between one node and all other nodes.

Finds distances by “relaxing” partial paths.

Bellman-Ford-Moore

Most of the time called Bellman-Ford.

Determines the distance between one node and all other nodes.

Finds distances by “relaxing” partial paths.

Requires an observation,

Bellman-Ford-Moore

Most of the time called Bellman-Ford.

Determines the distance between one node and all other nodes.

Finds distances by “relaxing” partial paths.

Requires an observation,

- A shortest path would consist of at most N vertices, assuming no negative cycles exist.

Bellman-Ford-Moore

Most of the time called Bellman-Ford.

Determines the distance between one node and all other nodes.

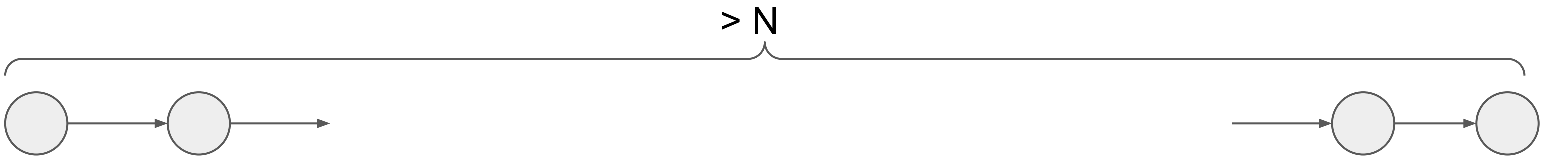
Finds distances by “relaxing” partial paths.

Requires an observation,

- A shortest path would consist of at most N vertices, **assuming no negative cycles exist**. Can be proven using Pigeonhole Principle and contradiction.

Proof

Assume a shortest path uses at least k (> N) vertices.



Proof

Assume a shortest path uses at least k (> N) vertices.

Then there exists two vertices that are identical.

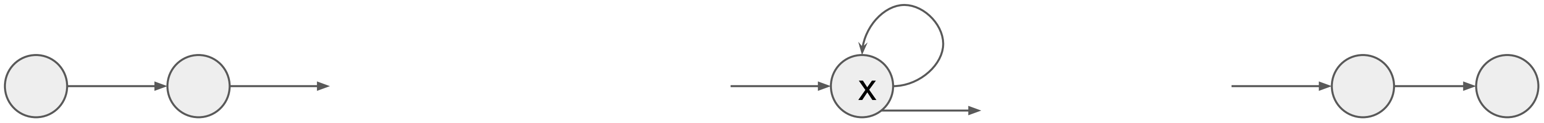
By Pigeonhole Principle.

Proof

Assume a shortest path uses at least k (> N) vertices.

Then there exists two vertices that are identical.

The paths between these form a cycle.



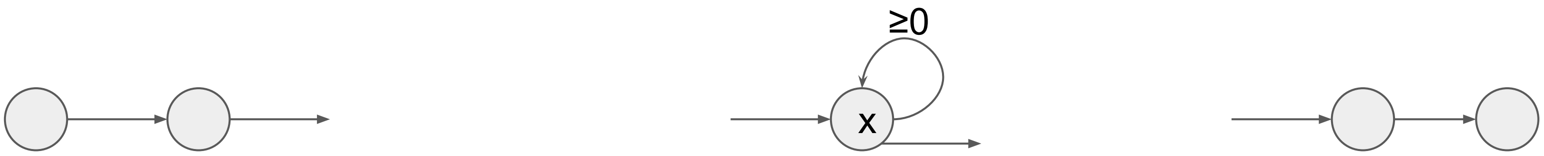
Proof

Assume a shortest path uses at least k (> N) vertices.

Then there exists two vertices that are identical.

The paths between these form a cycle.

The cycle is non-negative.



Proof

Assume a shortest path uses at least k (> N) vertices.

Then there exists two vertices that are identical.

The paths between these form a cycle.

The cycle is non-negative.

Removing does not worsen the shortest distance, but uses less than k nodes.

Proof

Assume a shortest path uses at least k (> N) vertices.

Then there exists two vertices that are identical.

The paths between these form a cycle.

The cycle is non-negative.

Removing does not worsen the shortest distance, but uses less than k nodes. ⊥

How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

How Bellman-Ford Works

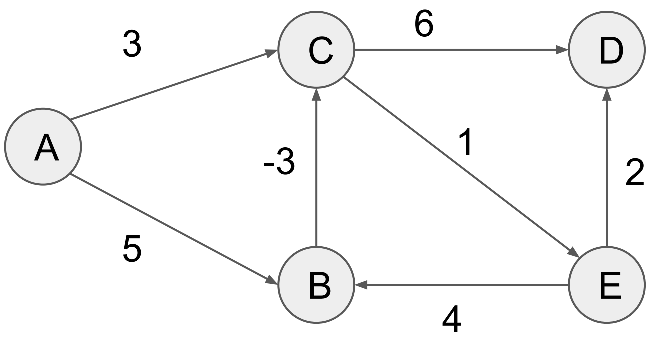
Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

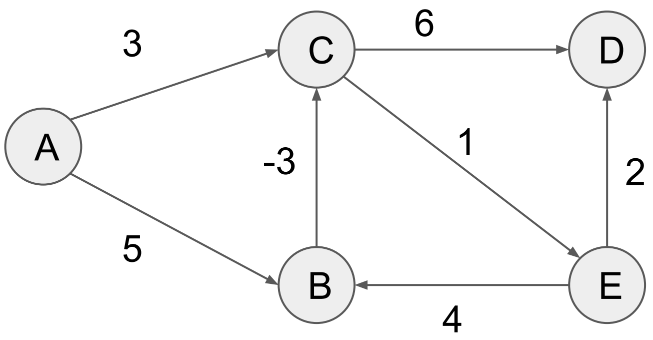


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	∞	∞	∞	∞

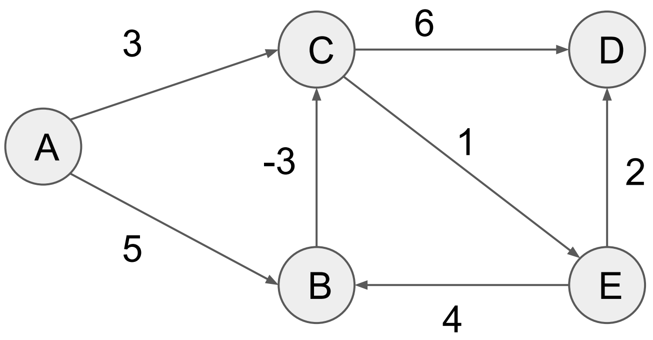


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	∞->5	∞->3	∞	∞

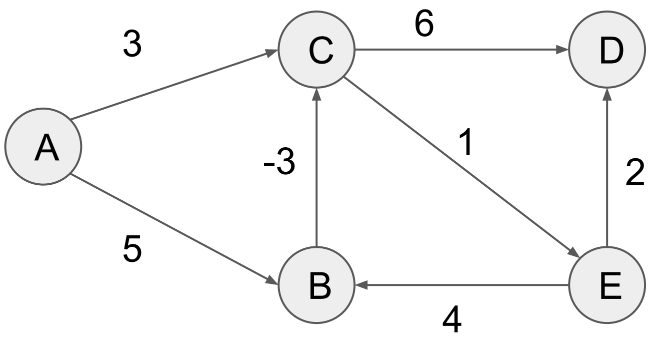


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	3	∞	∞

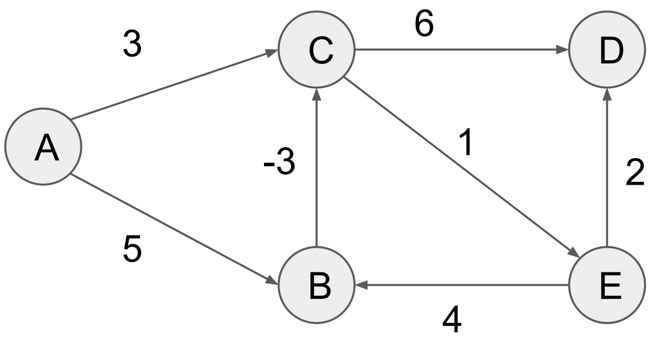


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	3->2	∞->9	∞->4

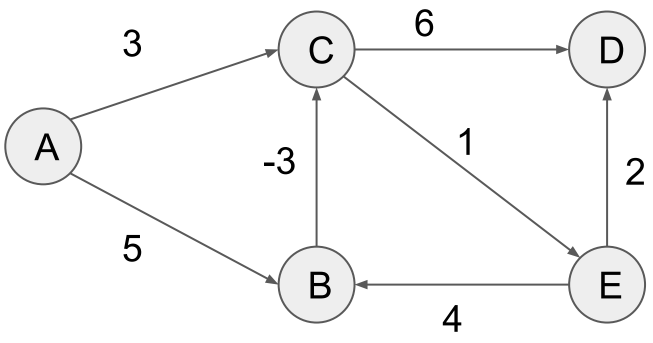


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	2	9	4

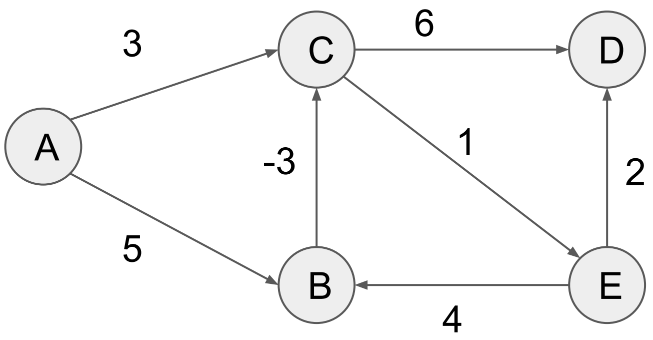


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	2	9->(8 or 6)	4->3

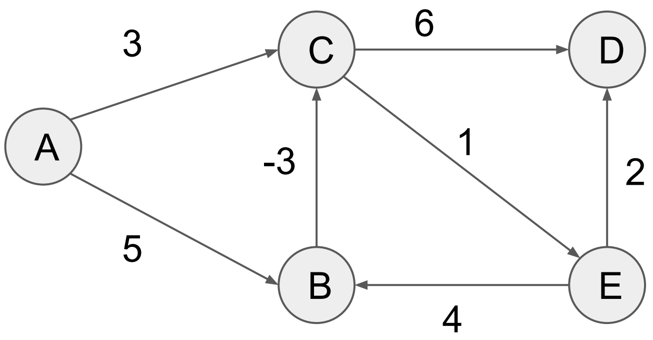


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	2	6	3

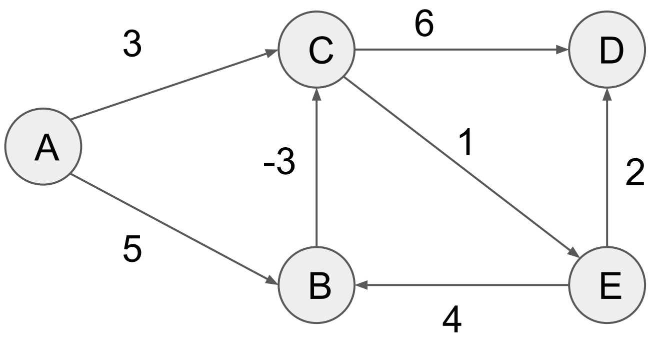


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	2	6->5	3

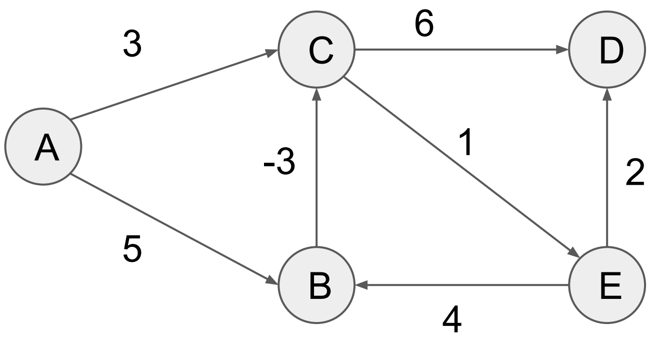


How Bellman-Ford Works

Assume every other node takes a very long distance to reach (∞).

Update theses distances N times using the edges in the graph.

Node	A	B	C	D	E
Distance	0	5	2	5	3



Bellman-Ford Analysis

How long would an update take?

Bellman-Ford Analysis

How long would an update take?

How many updates are there?

Bellman-Ford Analysis

How long would an update take?

How many updates are there?

What is the total runtime?

Negative Cycle Detection

Bellman-Ford can be used to detect if a negative cycle exists.

Negative Cycle Detection

Bellman-Ford can be used to detect if a negative cycle exists.

After updating N times…

Negative Cycle Detection

Bellman-Ford can be used to detect if a negative cycle exists.

After updating N times… Update again.

Negative Cycle Detection

Bellman-Ford can be used to detect if a negative cycle exists.

After updating N times… Update again.

If any value changes, a negative cycle must exist!

Bellman-Ford Summary

Works with negative edge weights.

Can find the shortest distance from some node to all others.

Can detect negative cycles.

Reasonable runtime: $\Theta(|V||E|)$

A Little Bit Faster

There is a lot of needless operations.

A Little Bit Faster

There is a lot of needless operations.

Why do we update the full graph?

A Little Bit Faster

There is a lot of needless operations.

Why do we update the full graph?

We can modify how we “relax” the distances to slightly improve runtime.

A Little Bit Faster

There is a lot of needless operations.

Why do we update the full graph?

We can modify how we “relax” the distances to slightly improve runtime.

Assuming no negative edge weights.

A Little Bit Faster

There is a lot of needless operations.

Why do we update the full graph?

We can modify how we “relax” the distances to slightly improve runtime.

Assuming no negative edge weights.

Dijkstra’s Algorithm.

Dijkstra’s algorithm

Assume every other node takes a very long distance to reach (∞).

Dijkstra’s algorithm

Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Dijkstra’s algorithm

Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

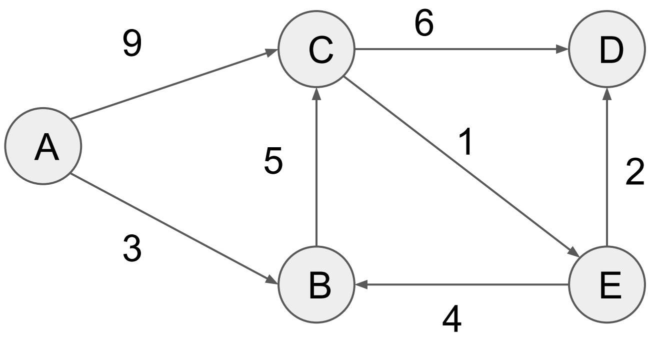
Dijkstra’s algorithm

Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.



Dijkstra’s algorithm

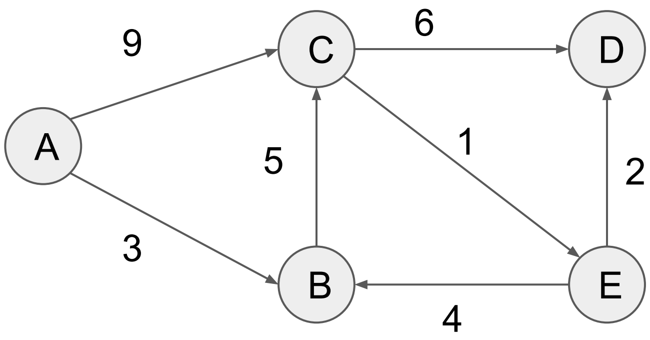
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	∞	∞	∞	∞



Dijkstra’s algorithm

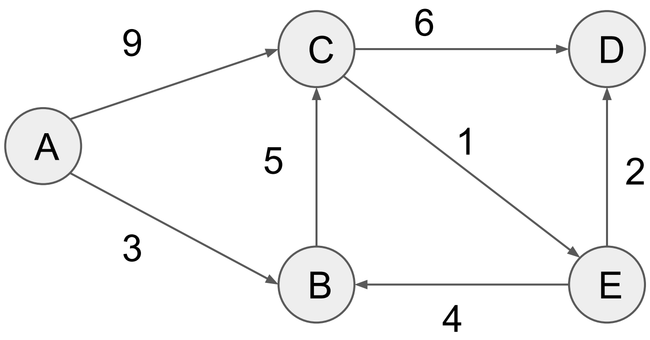
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	∞	∞	∞	∞



Dijkstra's algorithm

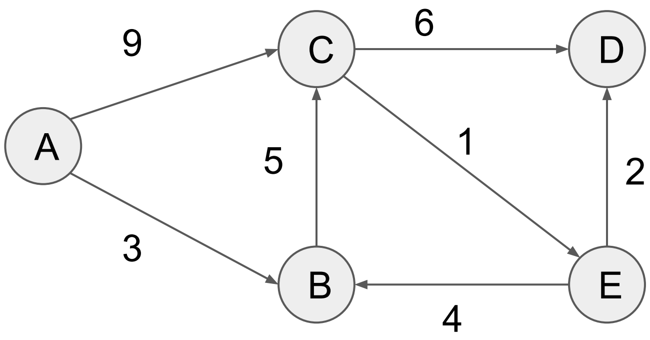
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	**B**	**C**	D	E
Distance	0	**3**	**9**	∞	∞



Dijkstra’s algorithm

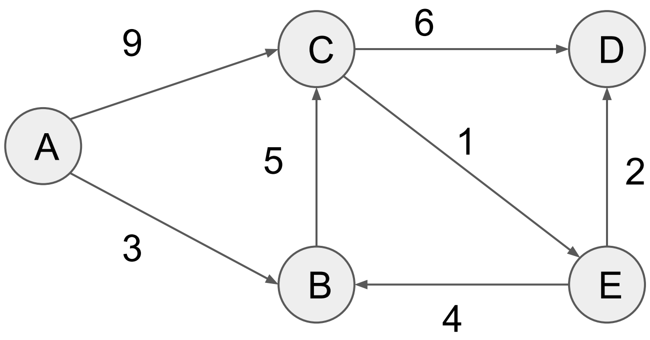
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	9	∞	∞



Dijkstra’s algorithm

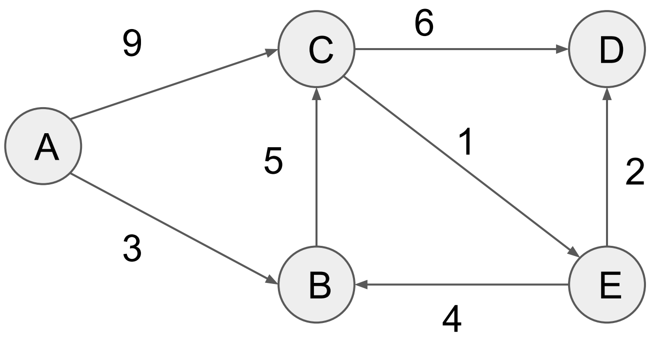
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	9	∞	∞



Dijkstra's algorithm

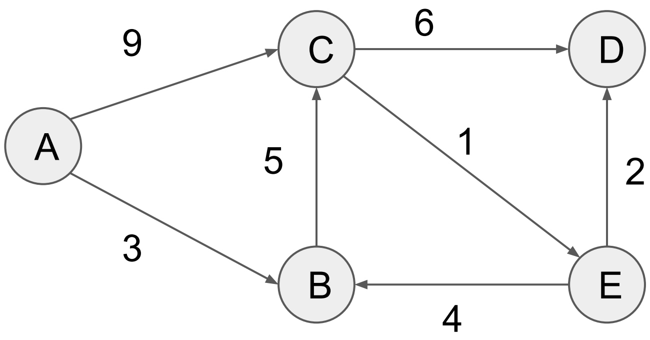
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	**C**	D	E
Distance	0	3	**8**	∞	∞



Dijkstra's algorithm

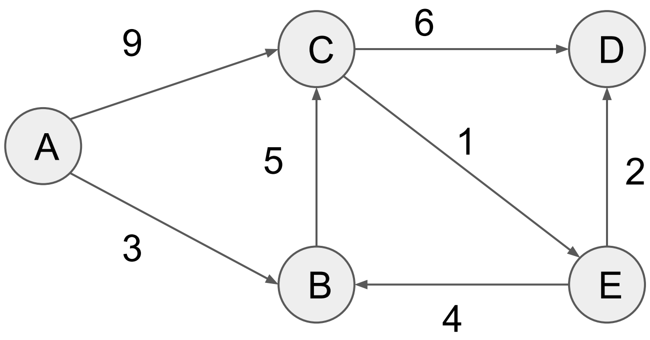
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	∞	∞



Dijkstra's algorithm

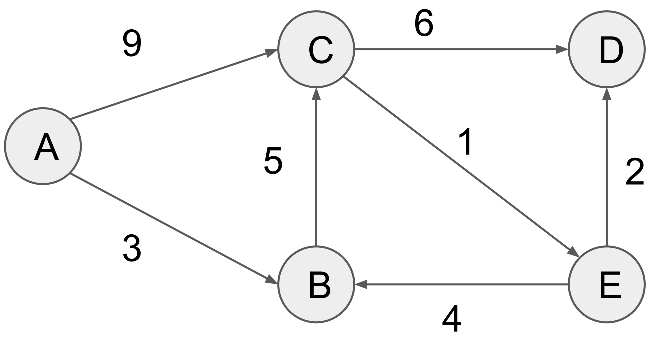
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	∞	∞



Dijkstra’s algorithm

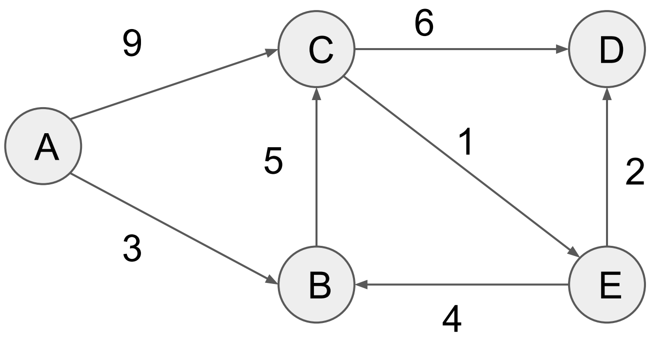
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	**D**	**E**
Distance	0	3	8	**14**	**9**



Dijkstra's algorithm

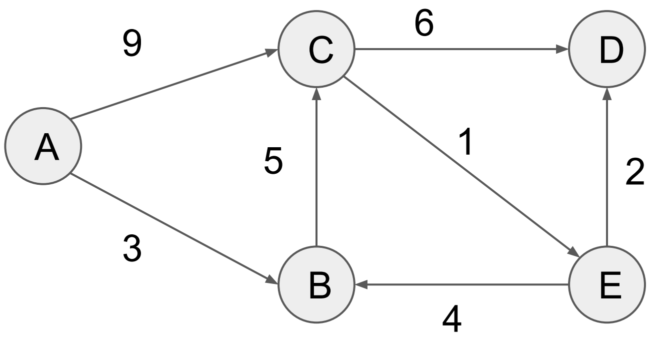
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	14	9



Dijkstra’s algorithm

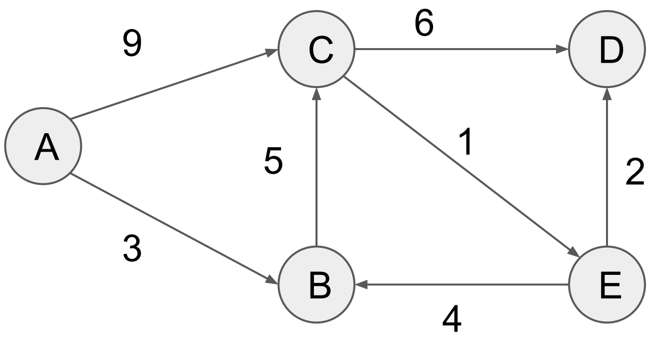
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	14	9



Dijkstra’s algorithm

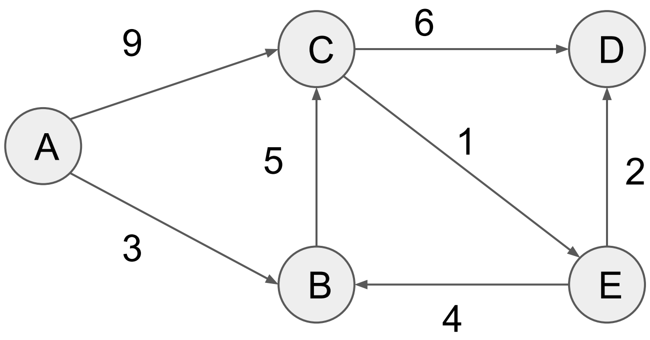
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	**D**	E
Distance	0	3	8	**11**	9



Dijkstra's algorithm

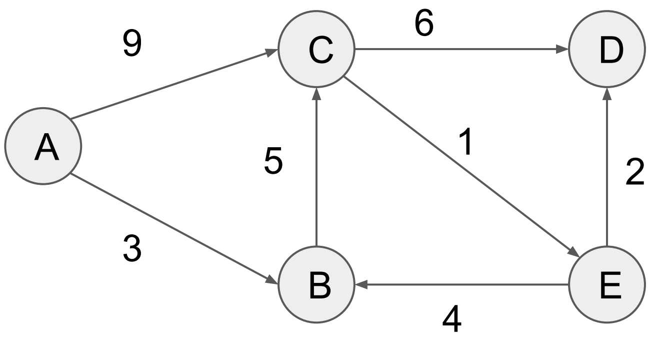
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	11	9



Dijkstra’s algorithm

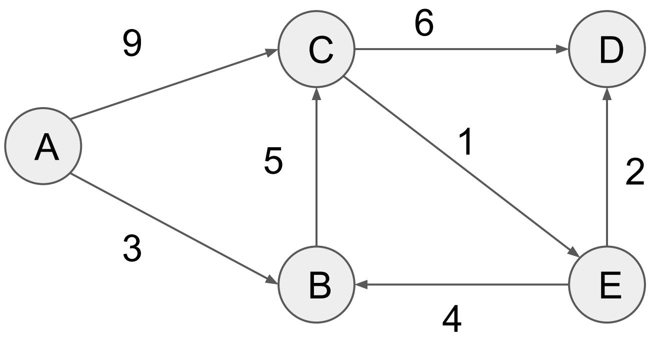
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	11	9



Dijkstra's algorithm

Assume every other node takes a very long distance to reach (∞).

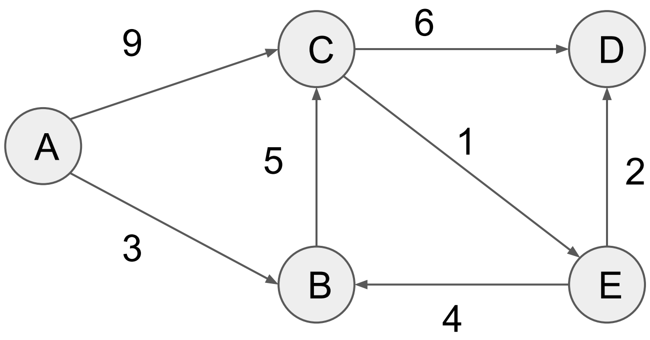
Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	11	9

No change.



Dijkstra's algorithm

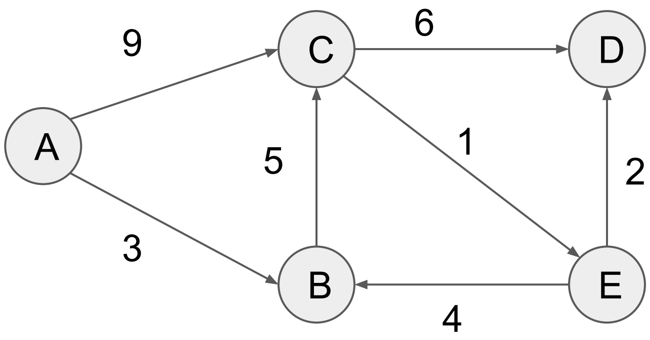
Assume every other node takes a very long distance to reach (∞).

Find the closest node not used.

Update the distances based on said node.

Repeat until all nodes are used.

Node	A	B	C	D	E
Distance	0	3	8	11	9



Dijkstra’s Analysis/Implementation Details

How many times do updates occur?

Dijkstra's Analysis/Implementation Details

How many times do updates occur?

|V| times

Dijkstra's Analysis/Implementation Details

How many times do updates occur?

    |V| times

How do we find the closest node to perform an update on?

Dijkstra’s Analysis/Implementation Details

How many times do updates occur?

|V| times

How do we find the closest node to perform an update on?

Priority Queue

Dijkstra's Analysis/Implementation Details

How many times do updates occur?

|V| times

How do we find the closest node to perform an update on?

Priority Queue

What is the most number of items that can exist in the priority queue?

Dijkstra’s Analysis/Implementation Details

How many times do updates occur?

 |V| times

How do we find the closest node to perform an update on?

 Priority Queue

What is the most number of items that can exist in the priority queue?

 |E|

Dijkstra’s Analysis/Implementation Details

How many times do updates occur?

|V| times

How do we find the closest node to perform an update on?

Priority Queue

What is the most number of items that can exist in the priority queue?

|E|

What would be the total number of operations to insert/remove the values?

Dijkstra’s Analysis/Implementation Details

How many times do updates occur?

  |V| times

How do we find the closest node to perform an update on?

  Priority Queue

What is the most number of items that can exist in the priority queue?

  |E|

What would be the total number of operations to insert/remove the values?

  |E|log(|E|)

Pseudo-code

```
Make a distance array of N infs
Set distance of source to 0
Add to an empty priority queue the pair (0, source)
Make a visited array of N falses
While the priority queue has elements
   Let current pair be the top of the priority queue
   Remove the top of the priority queue
   If visited the second term of the current pair
      Go to the top of the While loop
   End If
   Let visited of the second term of the current pair to true
   For all edges e leaving the second term of the current pair
      If e improves the distance for the destination of e
         Update distance for destination of e
         Add to the priority queue (new distance, destination of e)
      End If
   End For
End While
```