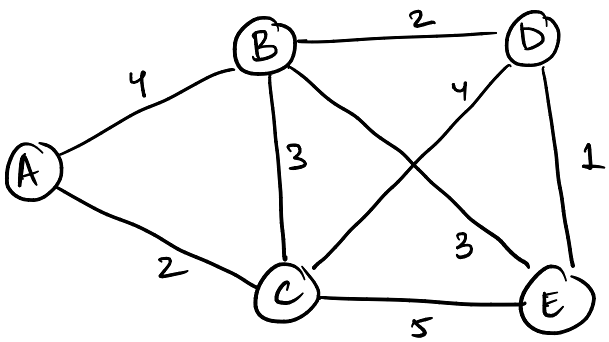
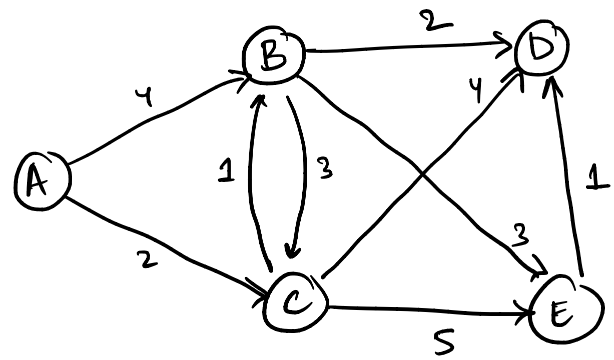
Adjacency Matrix



	A	B	C	D	E
A	0	4	2	0	0
B	4	0	3	2	3
C	0	3	0	4	5
D	0	2	4	0	1
E	0	3	5	1	0

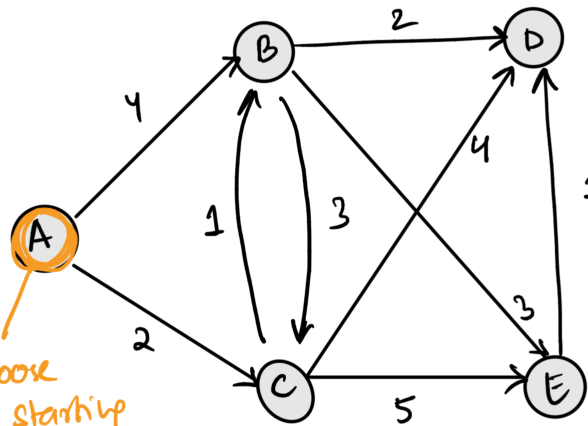


END

START

	A	B	C	D	E
A	0	4	2		
B	0	0	3	2	3
C		1	0	4	5
D				0	
E				1	0

Dijkstra's Algorithm — shortest path from one node to every other node



① choose as starting node.

PRIMS
KRUSKALS
} Minimum spanning trees

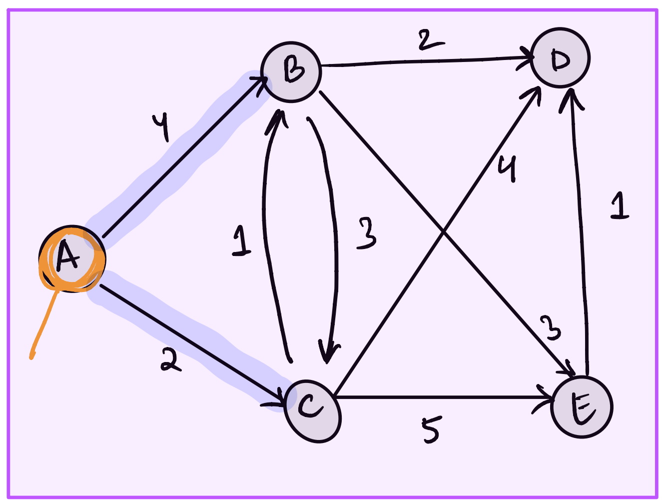
We only care about updating distances to unvisited nodes.

Unvisited Nodes [A, B, C, D, E]

② Distances from starting node.

A — 0
B — ∞
C — ∞
D — ∞
E — ∞

} haven't visited yet.



③ Examine edges leaving A

unvisited

A — 0 — 0
B — ∞ — 4
C — ∞ — 2
D — ∞ — ∞
E — ∞ — ∞

} pick smallest edge of vertices that haven't been chosen. i.e. C

i.e. unvisited.

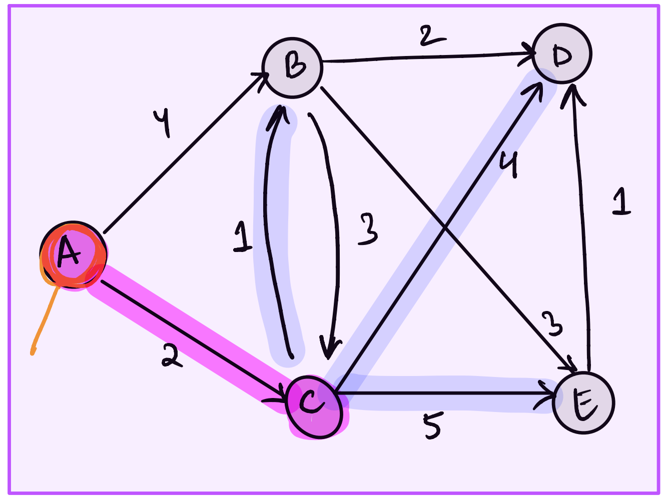
] still unreachable

C is chosen.

Unvisited Nodes [~~A~~, B, ~~C~~, D, E]

4. Examine edges leaving C

A	0	0	0
B	∞	4	2+1 = 3
C	∞	2	2 ← already visited
D	∞	∞	2+4 = 6
E	∞	∞	2+5 = 7



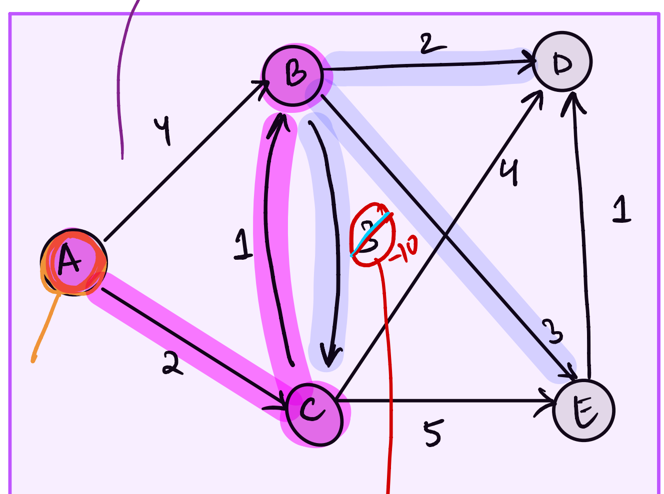
B is chosen.

Unvisited Nodes [~~A~~, ~~B~~, ~~C~~, D, E]

5. Examine edges leaving B

A	0	0	0	0
B	∞	4	3	3
C	∞	2	2	2
D	∞	∞	6	3+2 = 5
E	∞	∞	7	3+3 = 6

can not use this edge for further distance calculation.



Q what if this was -10?

D is chosen.

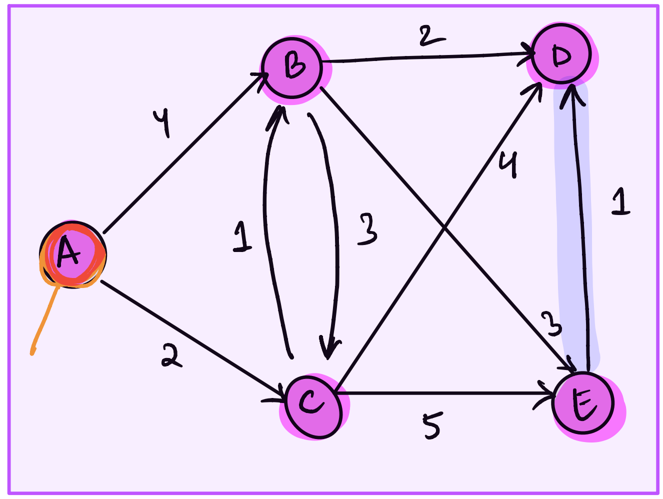
Unvisited Nodes [~~A~~, ~~B~~, ~~C~~, ~~D~~, E]

6. Examine edges leaving D → None!

No updates to the table.

7 Examine edges leaving E.

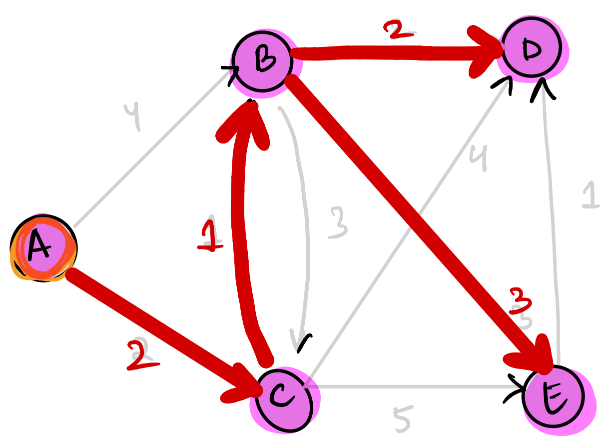
A	0	0	0	0	0
B	∞	4	3	3	3
C	∞	2	2	2	2
D	∞	∞	6	5	5
E	∞	∞	7	6	6



We have already visited D → no updates!

Unvisited Nodes [~~A~~, ~~B~~, ~~C~~, ~~D~~, ~~E~~]

All nodes have been visited. & no more updates.



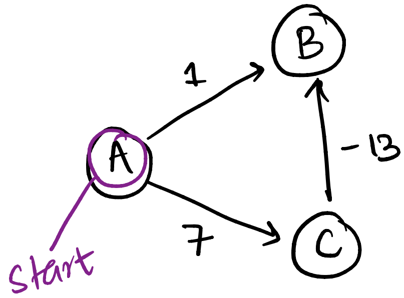
Shortest path from a to the other nodes.

Algo fails w negative weights

"The problem with negative weights arises from the fact that Dijkstra's algorithm assumes that once a node is added to the set of visited nodes, its distance is finalized and will not change. However, in the presence of negative weights, this assumption can lead to incorrect results."

Once a node is visited, we can't modify its shortest distance from the starting node.

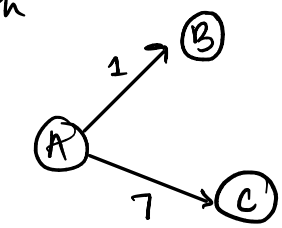
Eg.



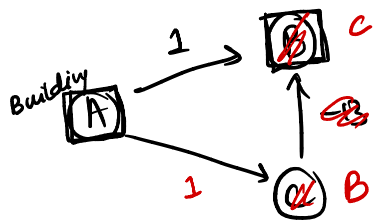
B chosen. — no outward edges

Next best — C — No updates since B already visited.

shortest path

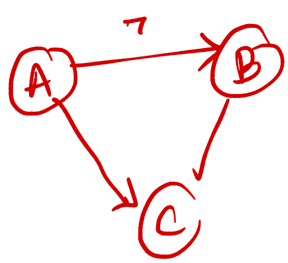


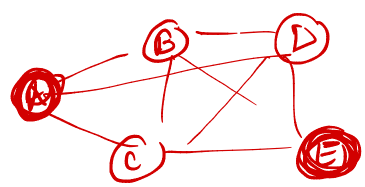
What if start node is C?



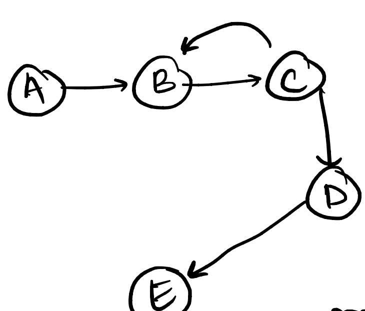
Shortest paths?

A	B
∞	-13





N



repeating a vertex.

can't have -ive cycles.
+ive are effectively bad / useless since you're adding distance

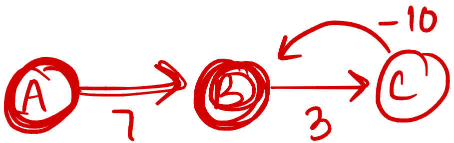
Once a vertex's minimum distance from the starting vertex is determined, the algorithm never reconsiders or backtracks this decision. Dijkstra "greedily" commits to this path as part of the shortest path, even if a shorter path could possibly be found by considering a longer path initially.

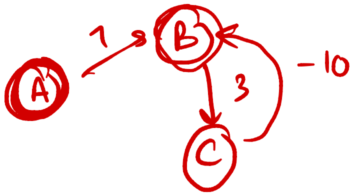
Bellman Ford

Negative Weights ✅
Detect negative cycles ✅

Relaxing an edge

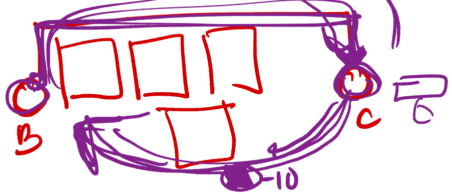
In the context of graph algorithms, particularly in algorithms like Bellman-Ford for finding the shortest path, to "relax" an edge means to check if the current known distance to reach one endpoint of the edge is greater than the distance to reach the other endpoint of the edge plus the edge's weight. If it is, then the current known distance is updated to this smaller value. Essentially, relaxing an edge means to update the shortest path estimate if a better path is found.





Key Idea

For the graph having N vertices, all the edges should be relaxed N-1 times to compute the single source shortest path.



Negative cycle detection:

In order to detect whether a negative cycle exists or not, relax all the edge one more time and if the shortest distance for any node reduces then we can say that a negative cycle exists. If we relax the edges N times, and there is any change in the shortest distance of any node between the N-1th and Nth relaxation than a negative cycle exists, otherwise not exist.

Why N-1?

Why N-1?

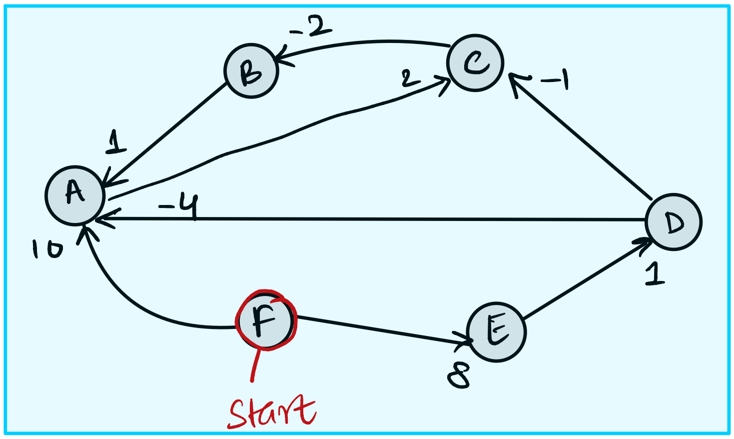
In a graph, a path is a sequence of edges which connects a sequence of vertices. A simple path is a path that does not include any vertex more than once, except possibly the first and last vertices if the path forms a cycle. Since a graph with N vertices has N distinct points, the longest simple path that can exist without revisiting any vertex (thereby not forming a closed loop) is one that visits each vertex exactly once.

This path would have N−1 edges, as each edge connects two vertices.

Relaxing edges N-1 times in the Bellman-Ford algorithm <u>guarantees that the algorithm has explored all possible paths of length up to N-1</u>, which is the maximum possible length of a shortest path in a graph with N vertices.

The reduction of distance during the N'th relaxation indicates revisiting a vertex–detecting negative cycles!

Bellman Ford



$V = 6$

$\therefore$ max iterations $= 5$

① Initialise distances from F

A	∞
B	∞
C	∞
D	∞
E	∞
F	0

For each iteration, look at a node one by one — this will gaurantee that you have seen all the edges.

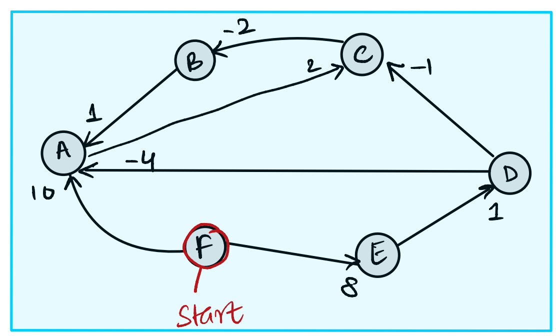
② Iteration 1

Distance from F

Look at node*	∞ A	∞ B	∞ C	∞ D	∞ E
F	F→A 10	F→B ∞	F→C $\cancel{\infty}$	F→D ∞	F→E 8
A	F→A 10	F→B ∞	F→C $10+2 = 12$	F→D ∞	F→E 8
B	10	$\cancel{\infty}$	12	∞	8
C	10	$12-2 = 10$	12	∞	8
D	-	-	-	$\cancel{\infty}$	-
E	10	10	12	$8+1 = 9$	8

havent reached B yet.

in sequence



* consider outgoing edges.

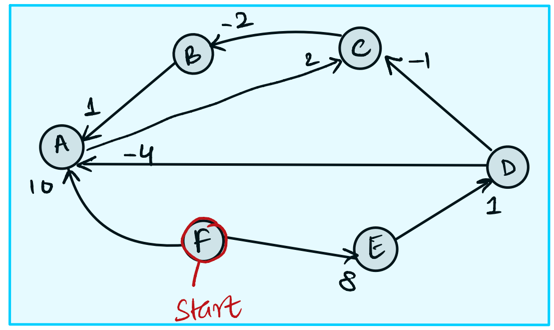
③ Iteration 2

Distance from F

Look at node*	A	B	C	D	E
	10	10	12	9	8
F	10	10	12	9	8
A	10	10	12	9	8
B	10	10	12	9	8
C	~~10~~	10	~~12~~	9	8
D	9−4 = 5	10	9−1 = 8	9	8
E	5	10	8	9	8

in sequence

no improvement in shortest path.



* consider outgoing edges in checking for what to update

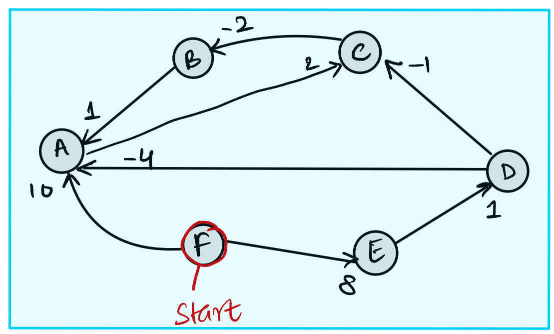
④ Iteration 3

Distance from F

Look at node*	A	B	C	D	E
	5	10	8	9	8
F	5	10	~~8~~	9	8
A	5	10	5+2 =7	9	8
B	5	10	7	9	8
C	5	7−2 =5	7	9	8
D	5	5	7	9	8
E	5	5	7	9	8

in sequence

no improvement in shortest path.



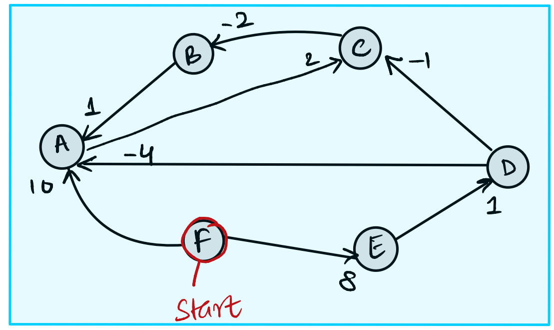
* consider outgoing edges in checking for what to update

⑤ Iteration 4

Distance from F

Look at node*	A^{5}	B^{5}	C^{7}	D^{9}	E^{8}
F	5	5	7	9	8
A	5	5	7	9	8
B	5	5	7	9	8
C	5	5	7	9	8
D	5	5	7	9	8
E	5	5	7	9	8

in sequence



* consider outgoing edges in checking for what to update

No values change. (At most |V|−1 iterations)

Can stop early

∴ shortest distances from F =

A	B	C	D	E
5	5	7	9	8