THE GRAPH

SHORTEST PATH ALGORITHMS

THE GRAPH

SHORTEST PATH ALGORITHMS

GIVEN A GRAPH G WITH VERTICES V AND EDGES E

CHOOSE ANY VERTEX S - THE SOURCE.

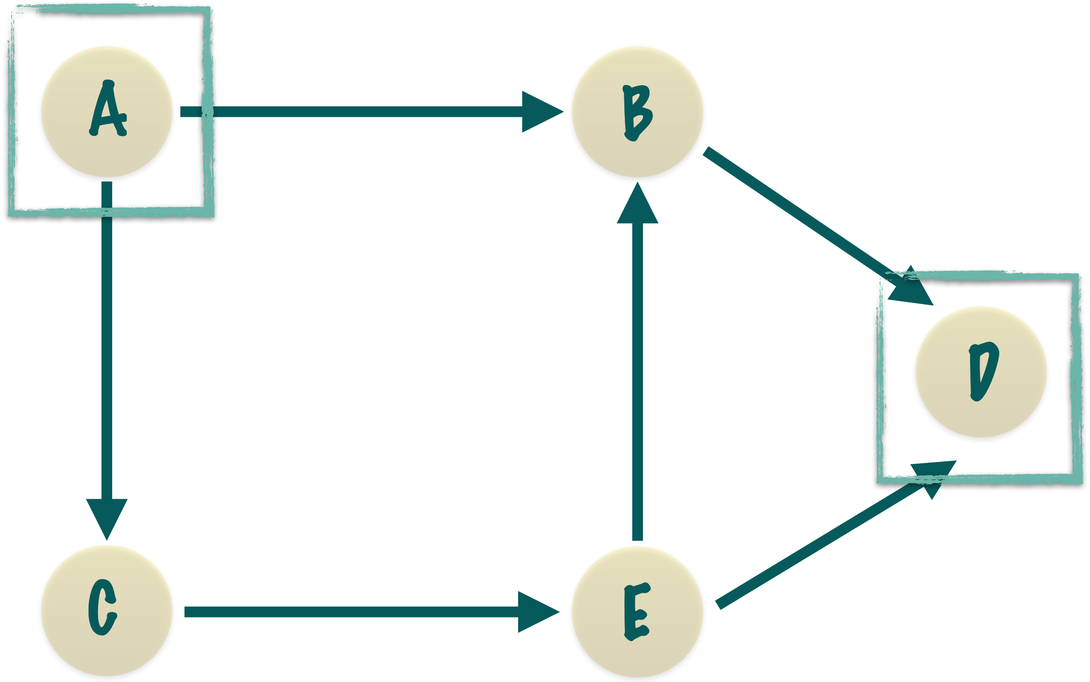
WHAT IS THE SHORTEST PATH FROM S TO
A SPECIFIC DESTINATION VERTEX D?

IT IS THE PATH WITH
THE FEWEST HOPS TO
GET FROM S TO D

THE GRAPH

SHORTEST PATH ALGORITHMS

LET A BE THE SOURCE AND D THE DESTINATION

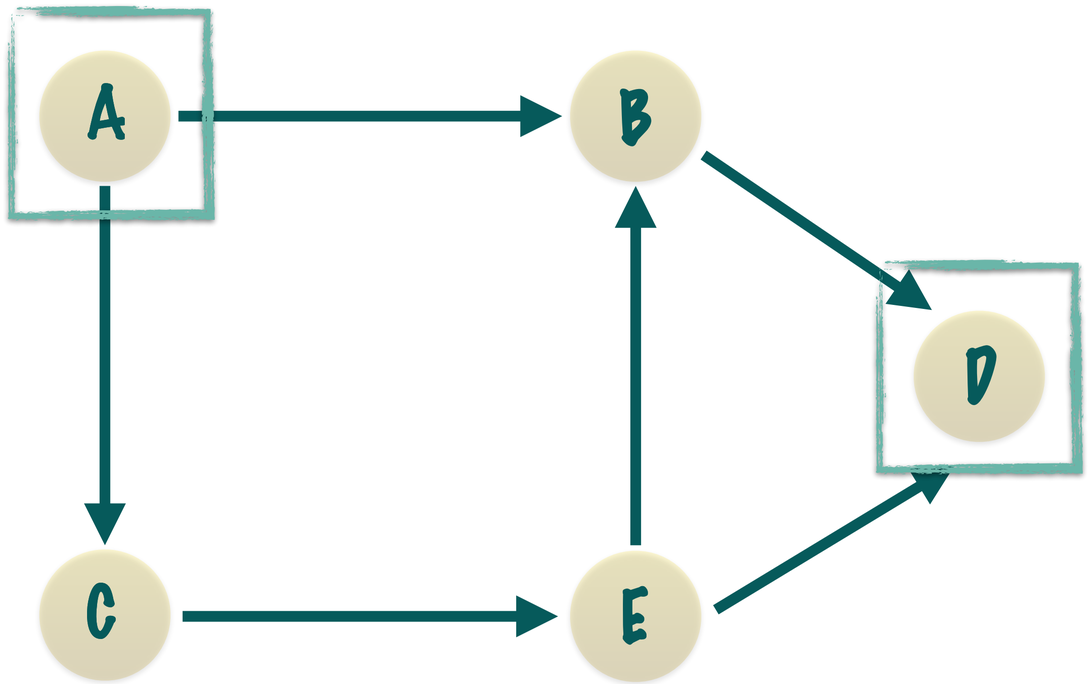


A -> B -> D ✓

A -> C -> E -> D ✗

A -> C -> E -> B -> D ✗

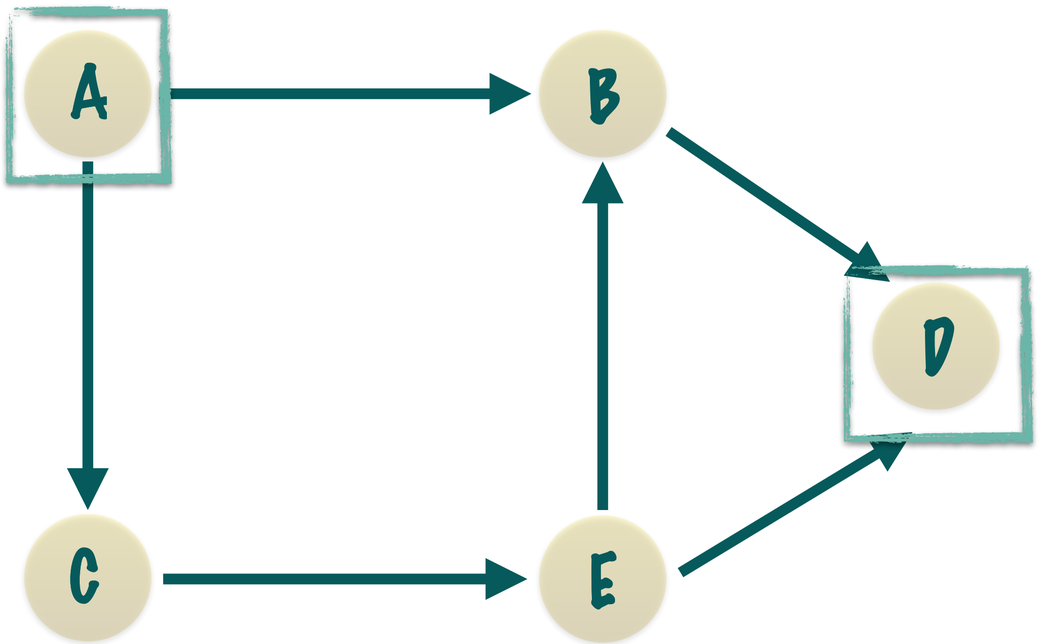
THE GRAPH SHORTEST PATH ALGORITHMS



	DISTANCE
A -> B -> D ✓	2
A -> C -> E -> D ✗	3
A -> C -> E -> B -> D ✗	4

THERE CAN BE MULTIPLE PATHS TO THE SAME VERTEX WITH DIFFERENT DISTANCES

THE GRAPH SHORTEST PATH ALGORITHMS



	DISTANCE
A -> B -> D	2
A -> C -> E -> D	3
A -> C -> E -> B -> D	4

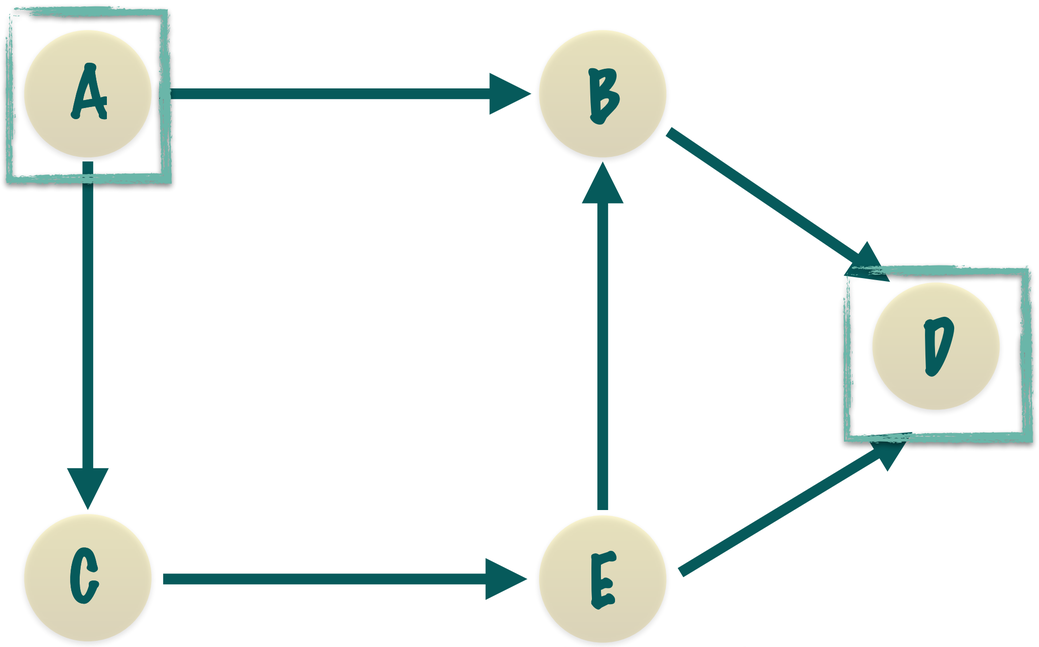
THERE CAN BE MULTIPLE PATHS TO THE SAME VERTEX WITH DIFFERENT DISTANCES

GETTING THE SHORTEST PATH IS VERY SIMILAR TO BFS

WE NEED TO SET UP SOMETHING CALLED A DISTANCE TABLE

A TABLE OF ALL VERTICES IN A GRAPH

THE GRAPH SHORTEST PATH ALGORITHMS



DISTANCE

A -> B -> D 2

A -> C -> E -> D 3

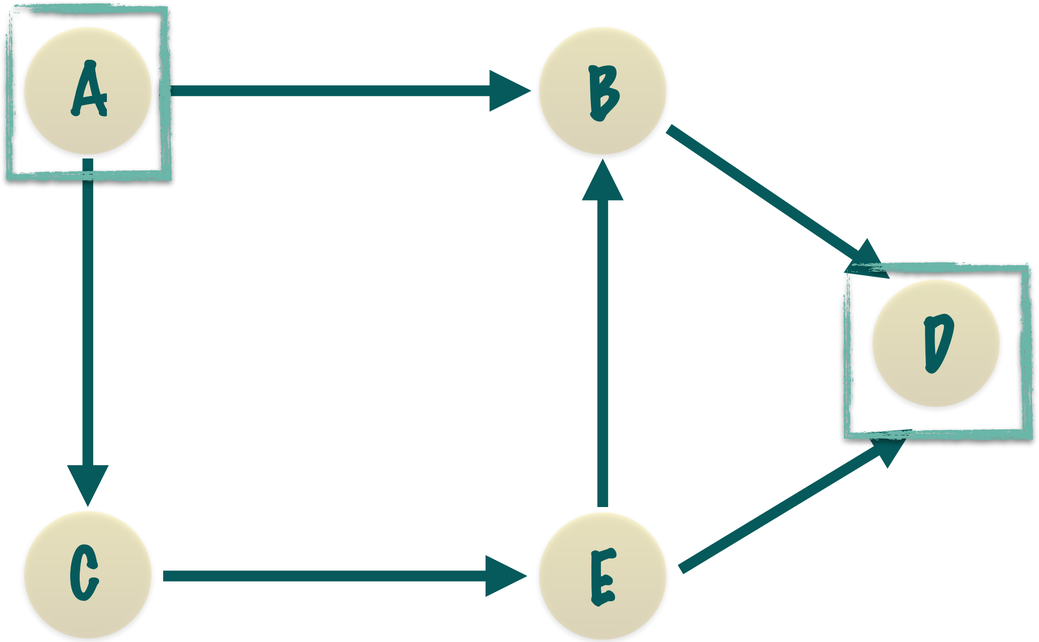
A -> C -> E -> B -> D 4

WE NEED TO SET UP SOMETHING CALLED A DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	-1	
C	-1	
D	-1	
E	-1	

HAS AN ENTRY FOR ALL VERTICES IN THE GRAPH

THE GRAPH SHORTEST PATH ALGORITHMS



DISTANCE

A -> B -> D 2

A -> C -> E -> D 3

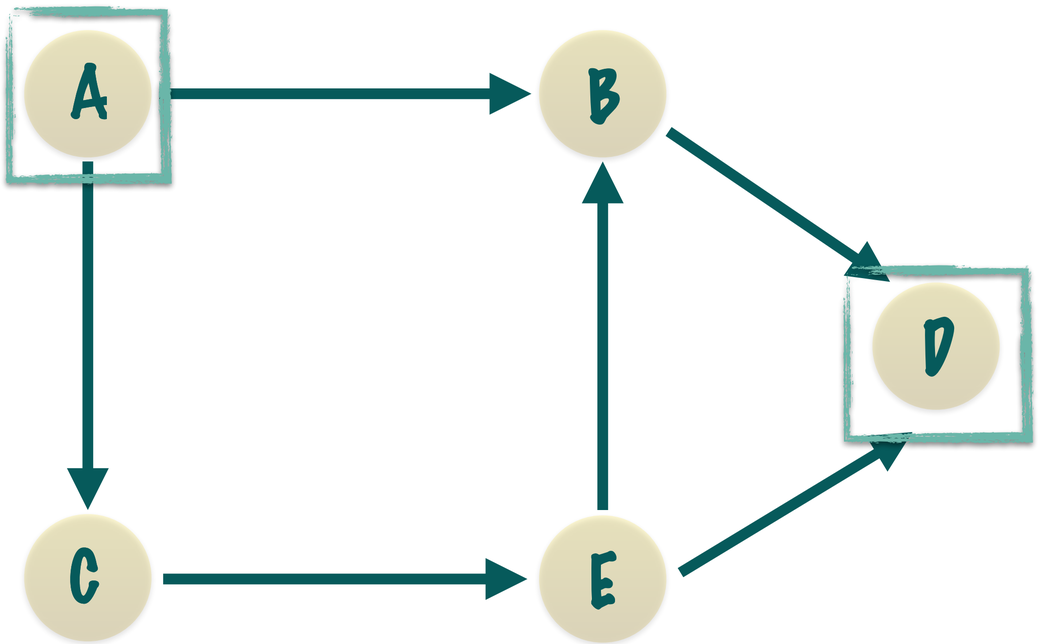
A -> C -> E -> B -> D 4

WE NEED TO SET UP SOMETHING CALLED A DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	-1	
C	-1	
D	-1	
E	-1	

THE DISTANCE COLUMN WILL HOLD THE DISTANCE OF THAT VERTEX FROM THE SOURCE A

THE GRAPH SHORTEST PATH ALGORITHMS



DISTANCE

A -> B -> D 2

A -> C -> E -> D 3

A -> C -> E -> B -> D 4

WE NEED TO SET UP SOMETHING CALLED A DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	-1	
C	-1	
D	-1	
E	-1	

THIS COLUMN IS THE LAST VERTEX IN THE PATH FROM THE SOURCE A TO THAT VERTEX

THE GRAPH SHORTEST PATH ALGORITHMS

DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	-1	
C	-1	
D	-1	
E	-1	

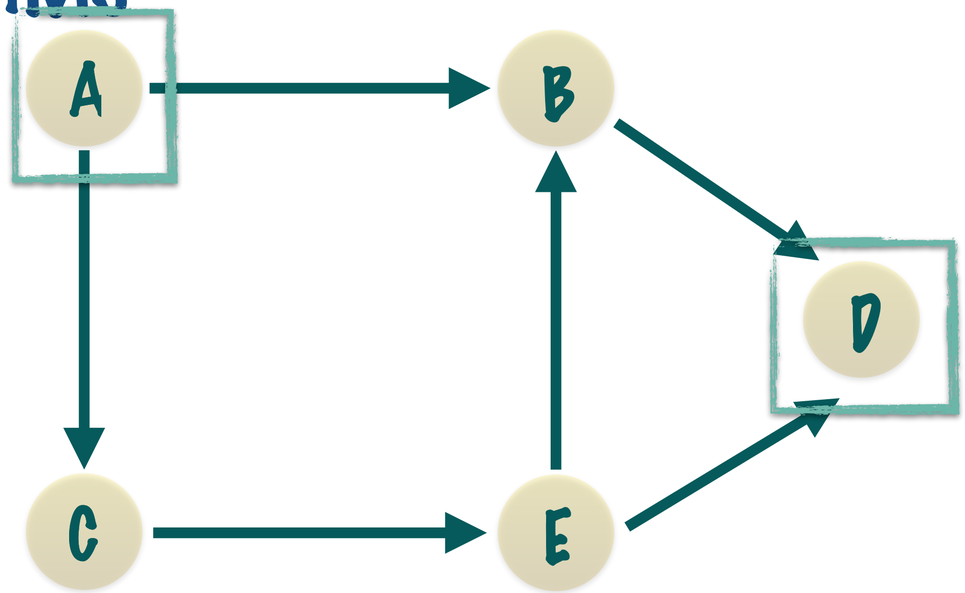
THE DISTANCE COLUMN WILL HOLD THE DISTANCE OF THAT VERTEX FROM THE SOURCE A

THIS COLUMN IS THE LAST VERTEX IN THE PATH FROM THE SOURCE A TO THAT VERTEX

DISTANCE [VERTEX] = DISTANCE [LAST VERTEX] + 1

IN UNWEIGHTED GRAPH, EACH ADJACENT VERTEX OF A SELECTED VERTEX IS AT A DISTANCE OF 1 EDGE FROM THAT VERTEX

THE GRAPH SHORTEST PATH ALGORITHMS



DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A		A
B		E
C		A
D		B
E		C

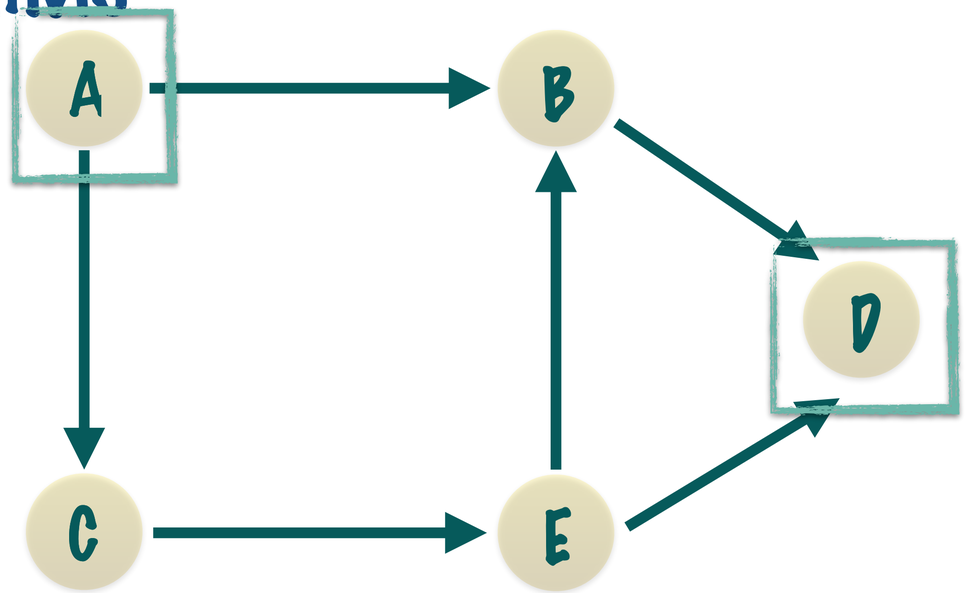
SUPPOSE THIS IS THE SHORTEST PATH FROM A TO D:

A -> C -> E -> B -> D

THE GRAPH SHORTEST PATH ALGORITHMS

DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	3	E
C	1	A
D	4	B
E	2	C



A -> C -> E -> B -> D

THE GRAPH SHORTEST PATH ALGORITHMS

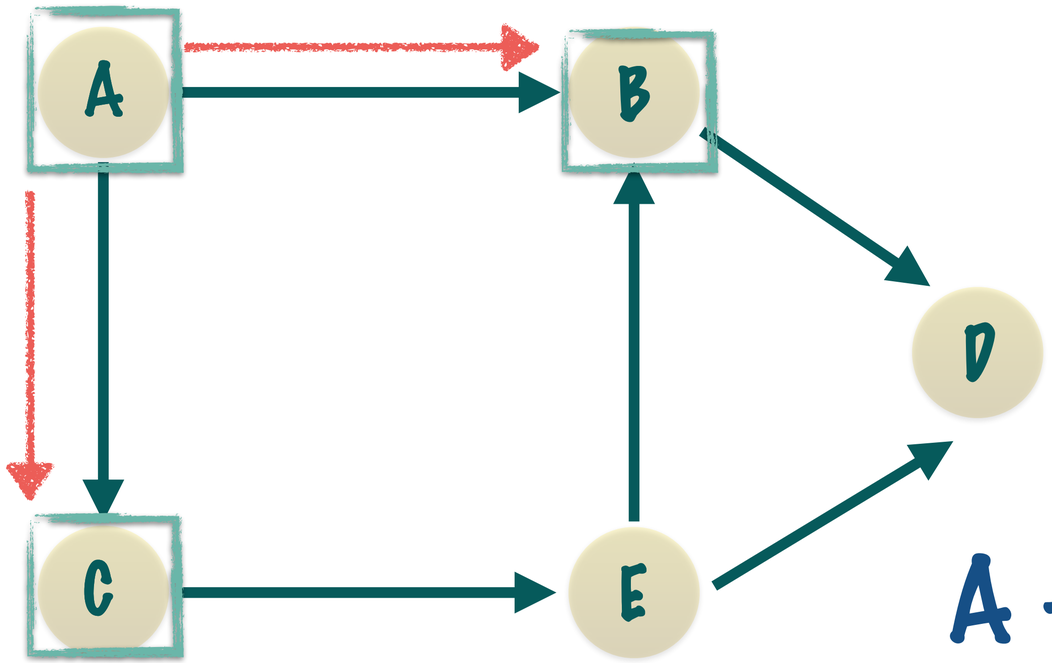
VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	-1	
C	-1	
D	-1	
E	-1	

WE INITIALIZE THIS TABLE BY SETTING -1 IN THE DISTANCE COLUMN FOR ALL VERTICES OTHER THAN THE SOURCE VERTEX

THE DISTANCE OF A FROM ITSELF IS 0

WE UPDATE THIS TABLE AS WE EXPLORE THE GRAPH USING BFS

THE GRAPH SHORTEST PATH ALGORITHMS



B AND C ARE NEIGHBOURS OF A. THERE EXISTS A DIRECT PATH TO BOTH B AND C.

A -> B

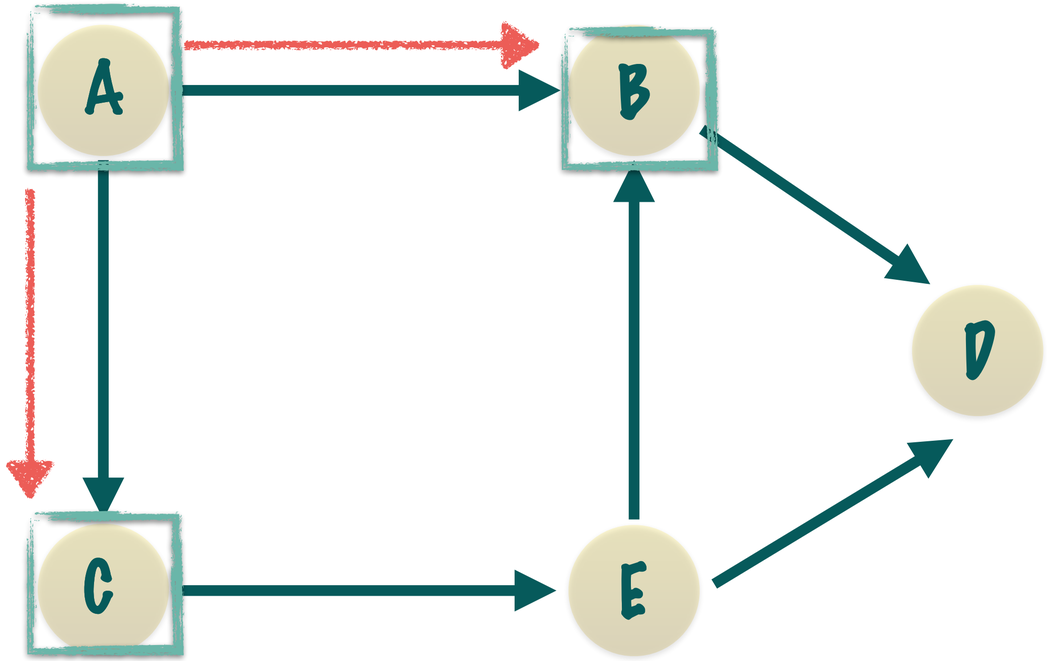
A -> C

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	-1	-
E	-1	-

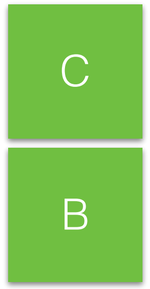
THE GRAPH SHORTEST PATH ALGORITHMS

A -> B

A -> C



VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	-1	-
E	-1	-



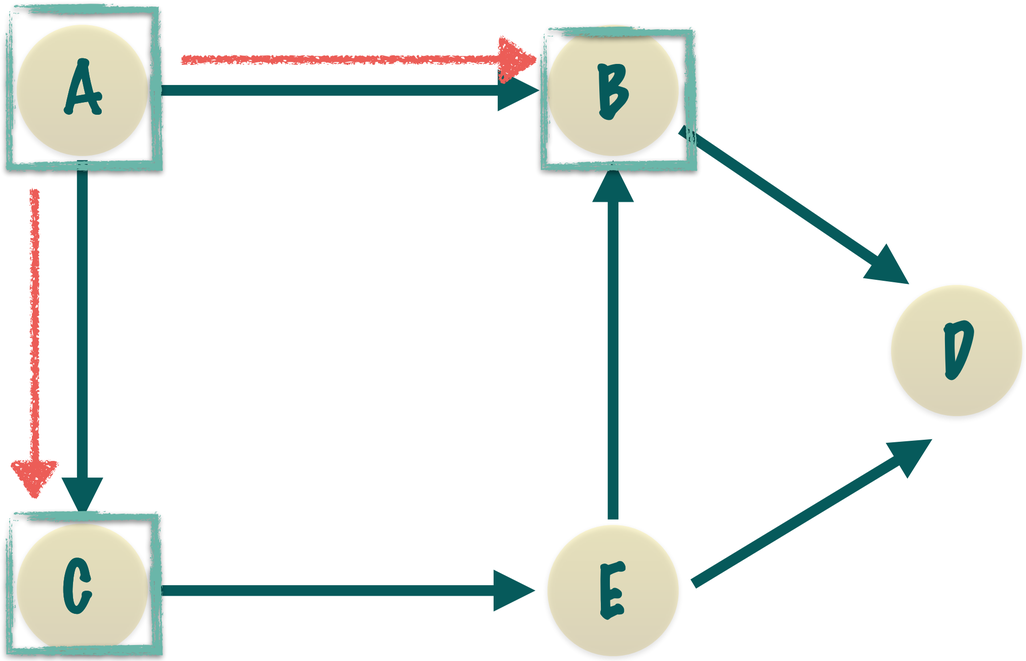
ADD B AND C TO THE QUEUE

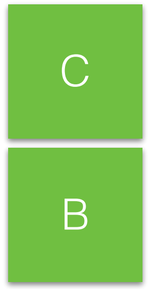
THE GRAPH SHORTEST PATH ALGORITHMS

PROCESSING C

A -> B

A -> C

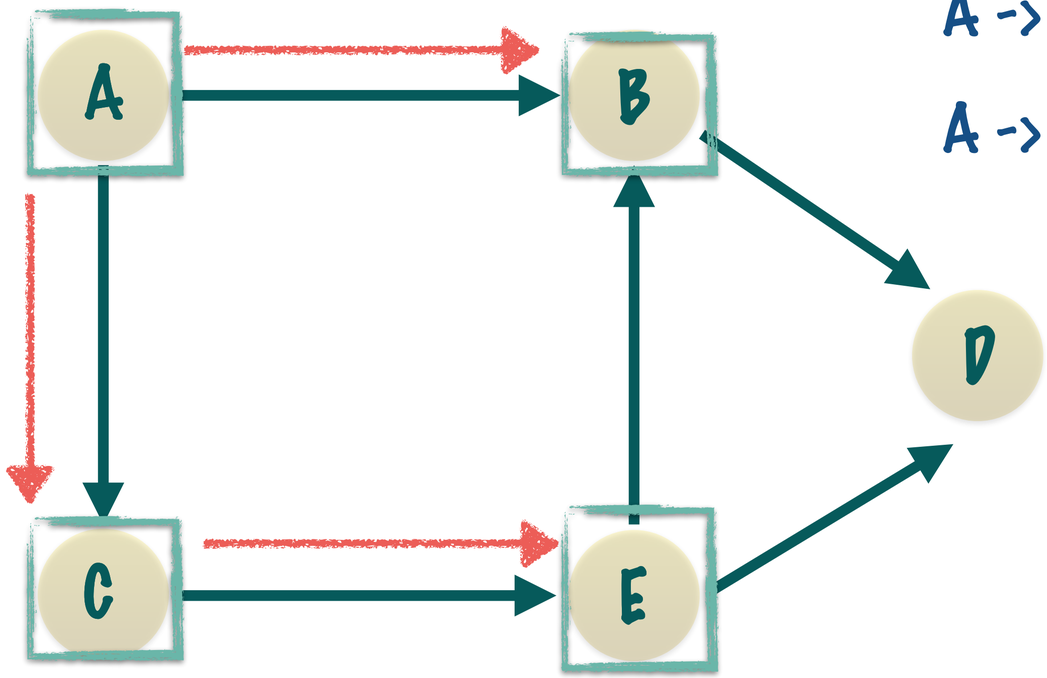




REMOVE C FROM THE QUEUE TO PROCESS

THE GRAPH SHORTEST PATH ALGORITHMS

PROCESSING C



A -> B

A -> C

A -> C -> E

FROM C WE CAN ONLY GET TO E

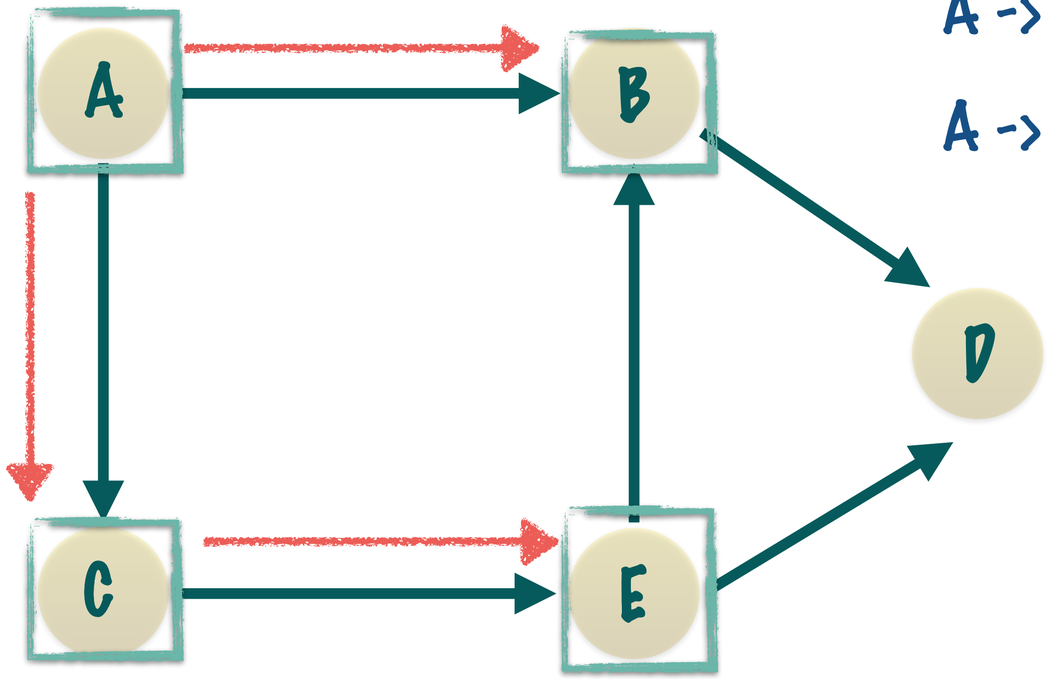
C'S NEIGHBOURS WILL BE AT ADDITIONAL DISTANCE OF 1 EDGE FROM SOURCE A

DISTANCE [E] = DISTANCE [C] + 1

B

THE GRAPH SHORTEST PATH ALGORITHMS

PROCESSING C



A -> B

A -> C

A -> C -> E

UPDATE THE DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	-1	-
E	2	C

DISTANCE [E] = DISTANCE [C] + 1

B

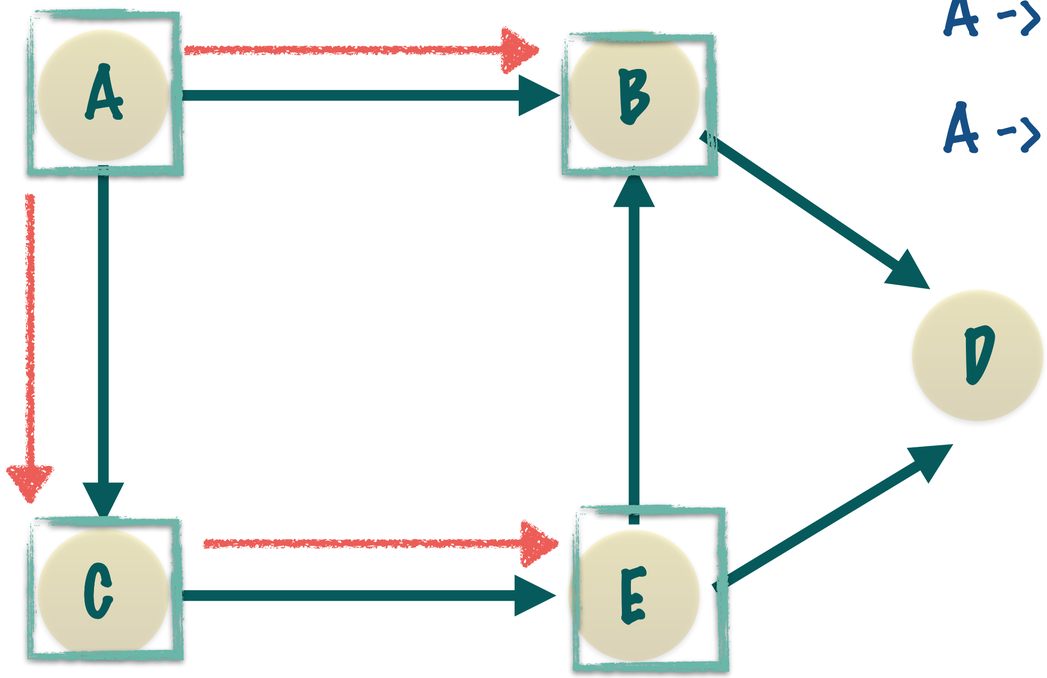
THE GRAPH SHORTEST PATH ALGORITHMS

PROCESSING C

A -> B

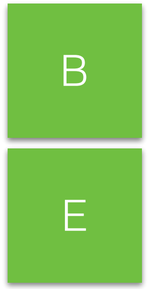
A -> C

A -> C -> E



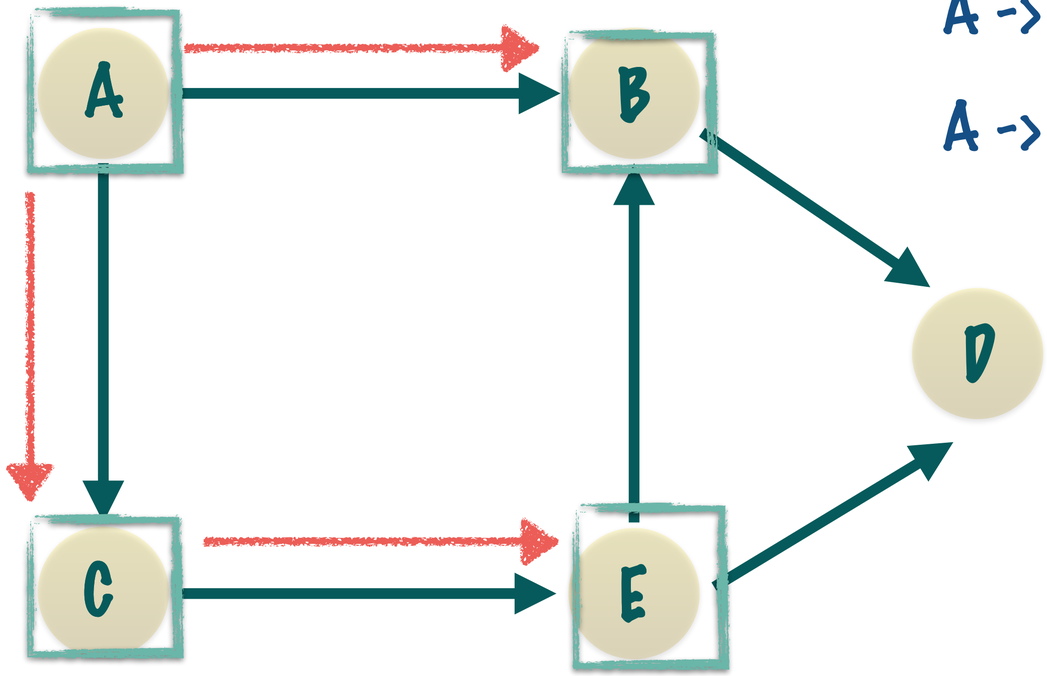
VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	-1	-
E	2	C

ENQUEUE E



THE GRAPH SHORTEST PATH ALGORITHMS

PROCESSING B

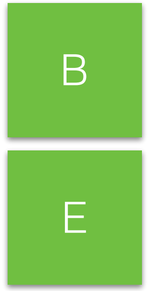


A -> B

A -> C

A -> C -> E

REMOVE B FROM THE QUEUE



THE GRAPH SHORTEST PATH ALGORITHMS

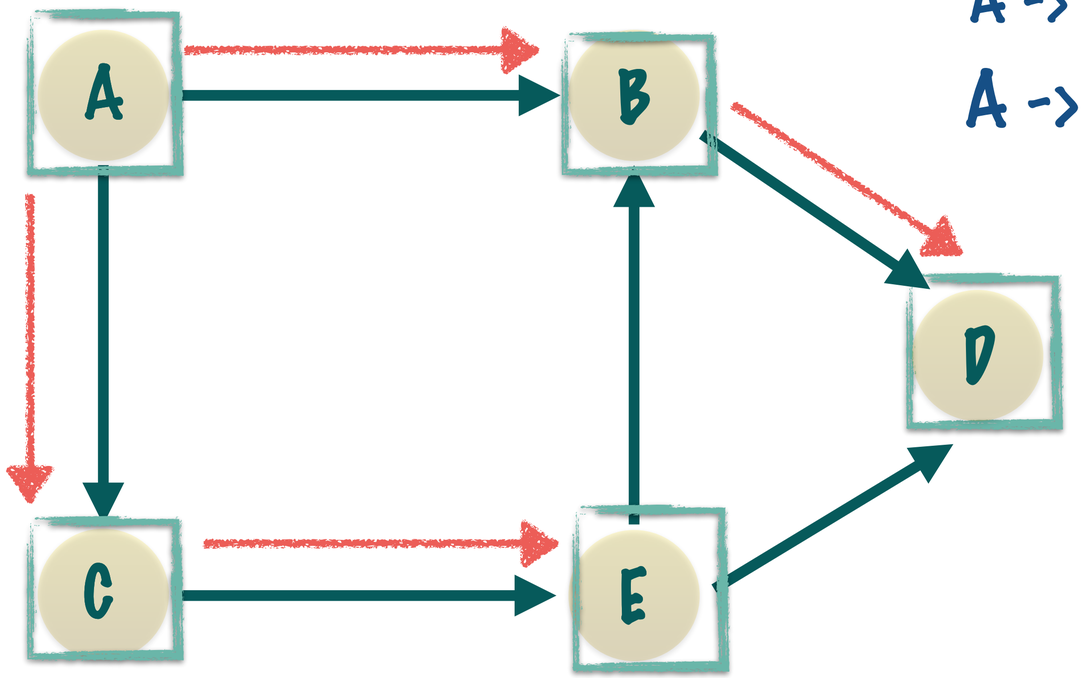
PROCESSING B

A -> B

A -> C

A -> C -> E

A -> B -> D



FROM B WE CAN ONLY GET TO D

B'S NEIGHBOURS WILL BE AT AN ADDITIONAL DISTANCE OF 1 EDGE FROM SOURCE A

DISTANCE [D] = DISTANCE [B] + 1

E

THE GRAPH SHORTEST PATH ALGORITHMS

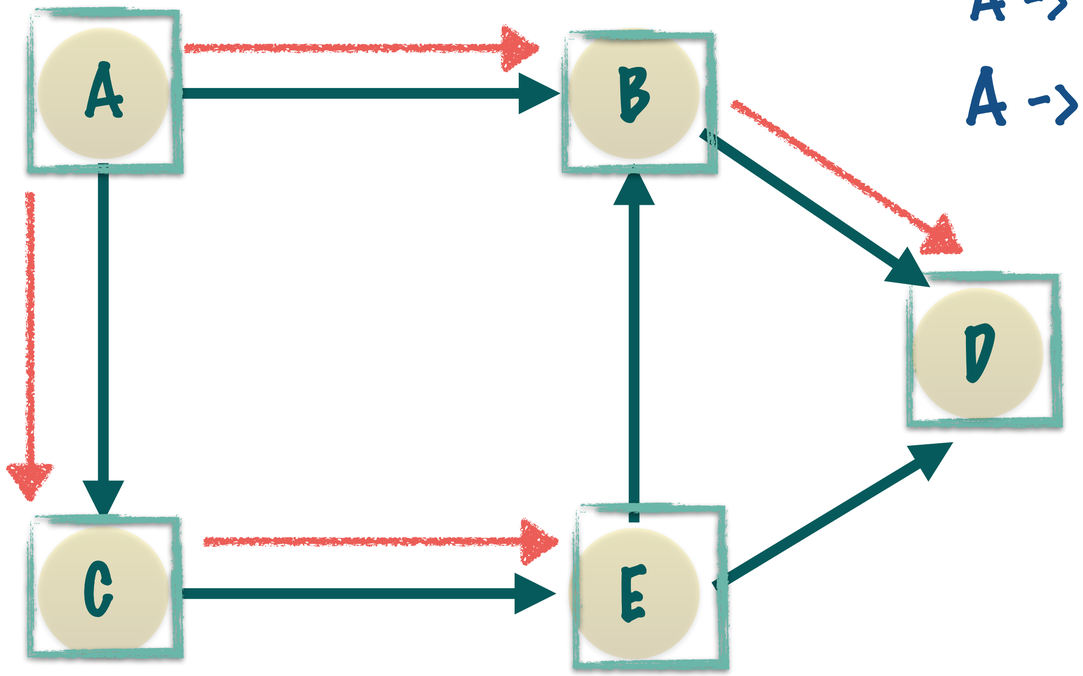
PROCESSING B

A -> B

A -> C

A -> C -> E

A -> B -> D



UPDATE THE DISTANCE TABLE

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

DISTANCE [D] = DISTANCE [B] + 1

E

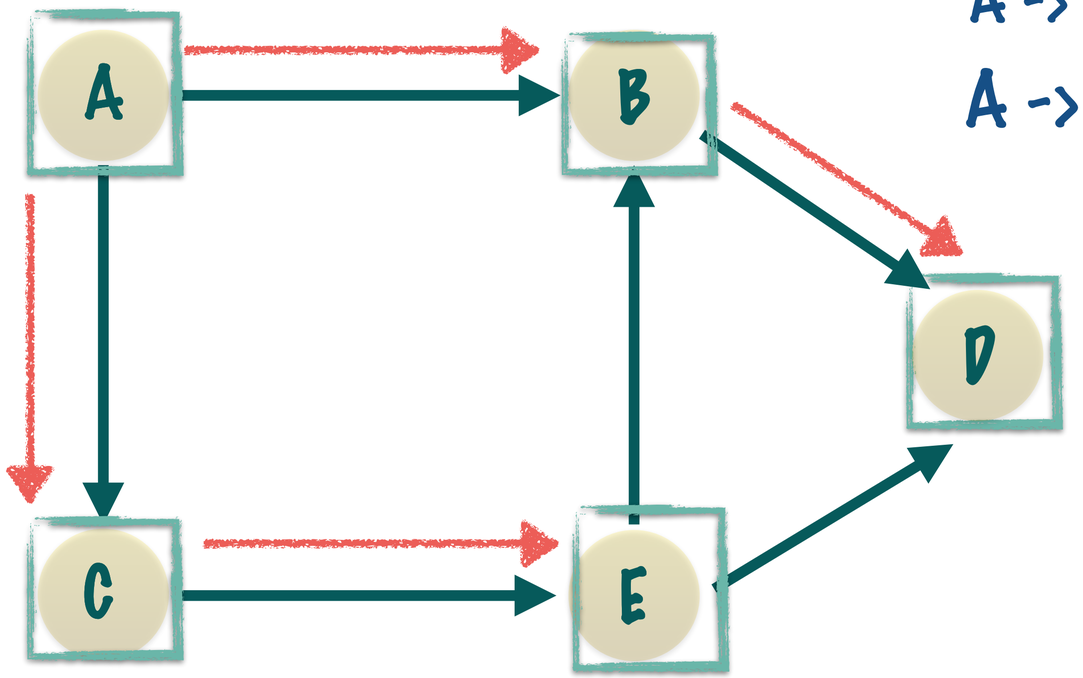
THE GRAPH SHORTEST PATH ALGORITHMS

A -> B

A -> C

A -> C -> E

A -> B -> D



VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

DO NOT ENQUEUE D AS IT HAS NO NEIGHBOURS!

E

THE GRAPH SHORTEST PATH ALGORITHMS

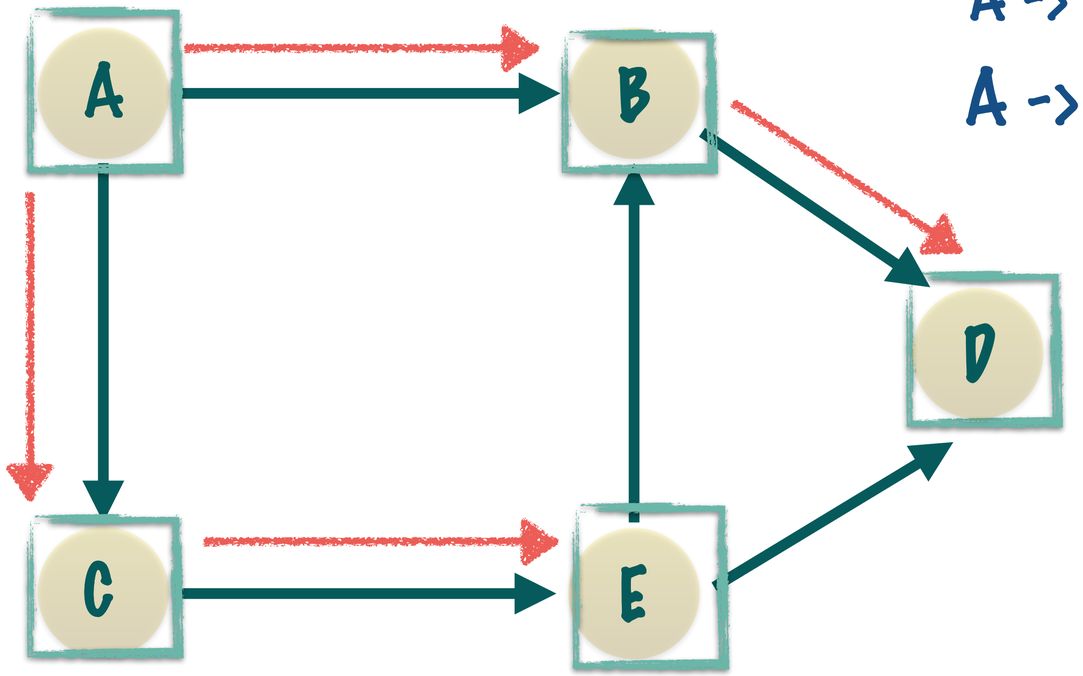
PROCESSING E

A -> B

A -> C

A -> C -> E

A -> B -> D



VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

REMOVE E FROM THE QUEUE

E

THE GRAPH SHORTEST PATH ALGORITHMS

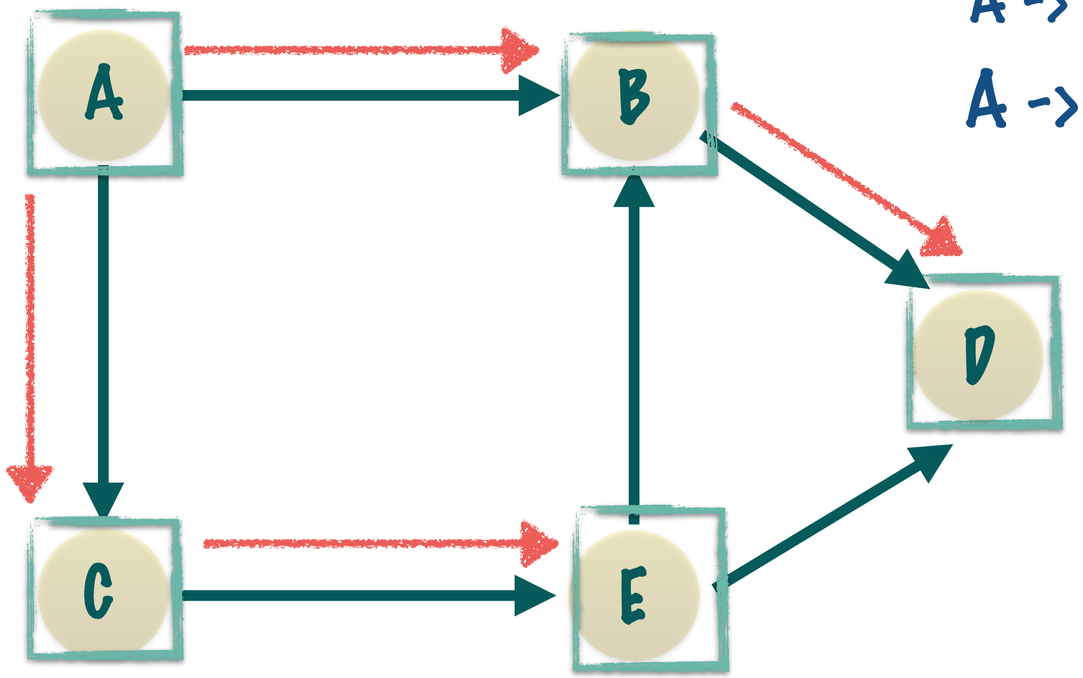
PROCESSING E

A -> B

A -> C

A -> C -> E

A -> B -> D



CHECK E'S NEIGHBOURS BUT ALL HAVE BEEN COVERED

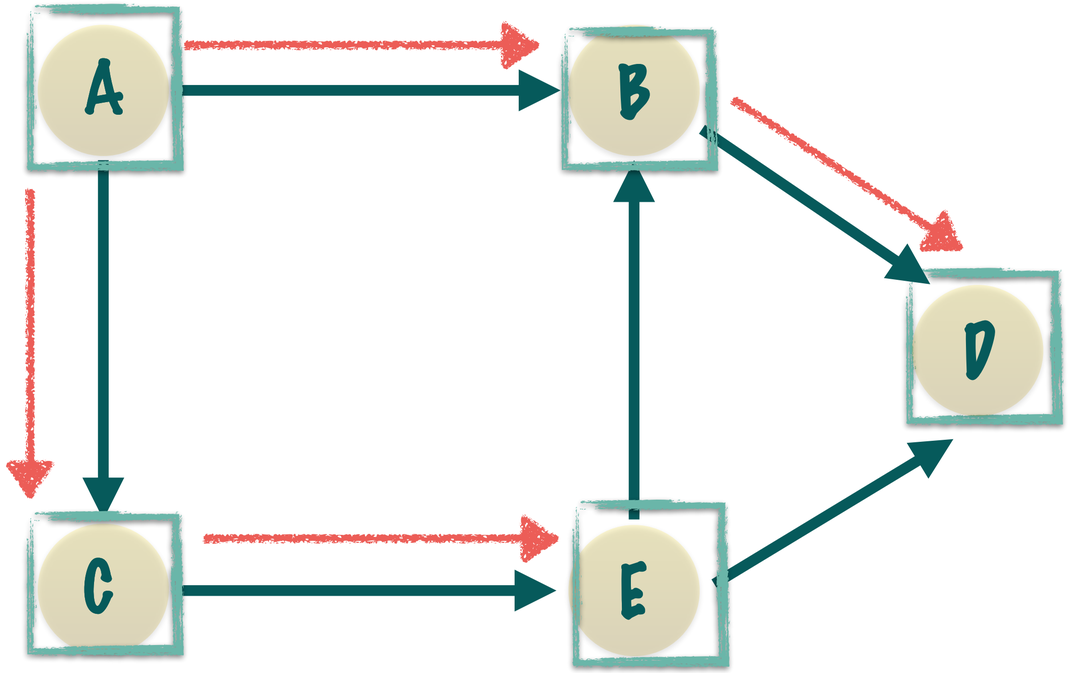
VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

WE VISIT EVERY NODE EXACTLY ONCE!

IF ANY VERTEX IS STILL NOT COVERED, THEN THERE IS NO PATH TO IT FROM OUR SOURCE VERTEX!

THE GRAPH SHORTEST PATH ALGORITHMS

GENERATE ALL PATHS USING THE DISTANCE TABLE



VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

USE THE STACK DATA STRUCTURE TO BACKTRACK FROM THE DESTINATION TO THE SOURCE NODE

THE GRAPH SHORTEST PATH ALGORITHMS

D IS OUR DESTINATION - START BACKTRACKING FROM D

PUT D ON THE STACK

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

D

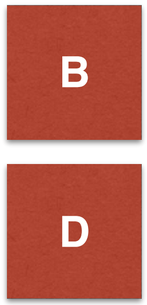
THE GRAPH SHORTEST PATH ALGORITHMS

PUT D ON THE STACK

HOW DID WE GET TO D? FROM B

PUT B ON THE STACK

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C



THE GRAPH SHORTEST PATH ALGORITHMS

PUT D ON THE STACK

HOW DID WE GET TO D? FROM B

PUT B ON THE STACK

HOW DID WE GET TO B? FROM A

PUT A ON THE STACK

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

A IS THE SOURCE!

THE GRAPH SHORTEST PATH ALGORITHMS

A IS THE SOURCE!

POP ELEMENTS FROM THE STACK
TO GET THE SHORTEST PATH

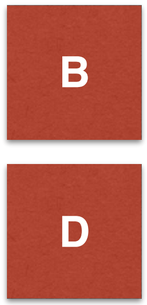
VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

THE GRAPH SHORTEST PATH ALGORITHMS

A IS THE SOURCE!

POP ELEMENTS FROM THE STACK
TO GET THE SHORTEST PATH

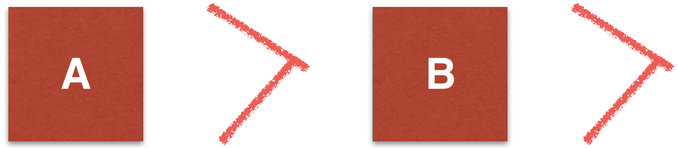
VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C



THE GRAPH SHORTEST PATH ALGORITHMS

A IS THE SOURCE!

POP ELEMENTS FROM THE STACK
TO GET THE SHORTEST PATH



VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

THE GRAPH SHORTEST PATH ALGORITHMS

A IS THE SOURCE!

POP ELEMENTS FROM THE STACK
TO GET THE SHORTEST PATH

VERTEX	DISTANCE	LAST VERTEX
A	0	A
B	1	A
C	1	A
D	2	B
E	2	C

THE GRAPH SHORTEST PATH ALGORITHMS

RUNNING TIME IS : O(V+E)
[IF ADJACENCY LISTS ARE USED]

RUNNING TIME IS : O(V*V)
[IF ADJACENCY MATRIX IS USED]

DISTANCE TABLE DATA STRUCTURE

```java
/**
 * A class which holds the distance information of any vertex.
 * The distance specified is the distance from the source node
 * and the last vertex is the last vertex just before the current
 * one while traversing from the source node.
 */
public static class DistanceInfo {

    private int distance;
    private int lastVertex;

    public DistanceInfo() {
        distance = -1;
        lastVertex = -1;
    }

    public int getDistance() {
        return distance;
    }

    public int getLastVertex() {
        return lastVertex;
    }

    public void setDistance(int distance) {
        this.distance = distance;
    }

    public void setLastVertex(int lastVertex) {
        this.lastVertex = lastVertex;
    }
}
```

FOR EVERY VERTEX STORE

1. THE DISTANCE TO THE VERTEX FROM THE SOURCE
2. THE LAST VERTEX IN THE PATH FROM THE SOURCE

GETTERS AND SETTERS FOR THE DISTANCE AND THE LAST VERTEX

BUILD THE DISTANCE TABLE

```
private static Map<Integer, DistanceInfo> buildDistanceTable(Graph graph, int source) {
    Map<Integer, DistanceInfo> distanceTable = new HashMap<>();
    for (int j = 0; j < graph.getNumVertices(); j++) {
        distanceTable.put(j, new DistanceInfo());
    }

    distanceTable.get(source).setDistance(0);
    distanceTable.get(source).setLastVertex(source);

    LinkedList<Integer> queue = new LinkedList<>();
    queue.add(source);

    while (!queue.isEmpty()) {
        int currentVertex = queue.pollFirst();
        for (int i : graph.getAdjacentVertices(currentVertex)) {
            int currentDistance = distanceTable.get(i).getDistance();
            if (currentDistance == -1) {
                currentDistance = 1 + distanceTable.get(currentVertex).getDistance();
                distanceTable.get(i).setDistance(currentDistance);
                distanceTable.get(i).setLastVertex(currentVertex);
                // Enqueue the neighbour only if it has other adjacent vertices.
                if (!graph.getAdjacentVertices(1).isEmpty()) {
                    queue.add(i);
                }
            }
        }

    }
    return distanceTable;
}
```

SET AN ENTRY IN THE DISTANCE TABLE FOR EVERY VERTEX IN THE GRAPH

INITIALIZE THE DISTANCE TO THE SOURCE AND THE LAST VERTEX IN THE PATH TO SOURCE

ADD THE SOURCE TO THE QUEUE

EXPLORE THE NEIGHBORING VERTICES OF EVERY VERTEX ADDED TO THE QUEUE

IF THE VERTEX IS SEEN FOR THE FIRST TIME THEN UPDATE IT'S ENTRY IN THE DISTANCE TABLE

IF A VERTEX HAS NEIGHBORS ADD IT TO THE QUEUE SO NEIGHBOURS CAN BE EXPLORED

SHORTEST PATH

```java
public static void shortestPath(Graph graph, int source, int destination) {
    Map<Integer, DistanceInfo> distanceTable = buildDistanceTable(graph, source);

    Stack<Integer> stack = new Stack<>();
    stack.push(destination);

    int previousVertex = distanceTable.get(destination).getLastVertex();
    while (previousVertex != -1 && previousVertex != source) {
        stack.push(previousVertex);
        previousVertex = distanceTable.get(previousVertex).getLastVertex();
    }

    if (previousVertex == -1) {
        System.out.println("There is no path from node: " + source
                + " to node: " + destination);
    }
    else {
        System.out.print("Smallest path is " + source);
        while (!stack.isEmpty()) {
            System.out.print(" -> " +stack.pop());
        }
        System.out.println(" Shortest Path Unweighted DONE!");
    }
}
```

BUILD THE DISTANCE TABLE FOR THE ENTIRE GRAPH

BACKTRACK USING A STACK, START FROM THE DESTINATION NODE

BACKTRACK BY GETTING THE LAST VERTEX OF EVERY NODE AND ADDING IT TO THE STACK

IF NO VALID LAST VERTEX WAS FOUND IN THE DISTANCE TABLE, THERE WAS NO PATH FROM SOURCE TO DESTINATION