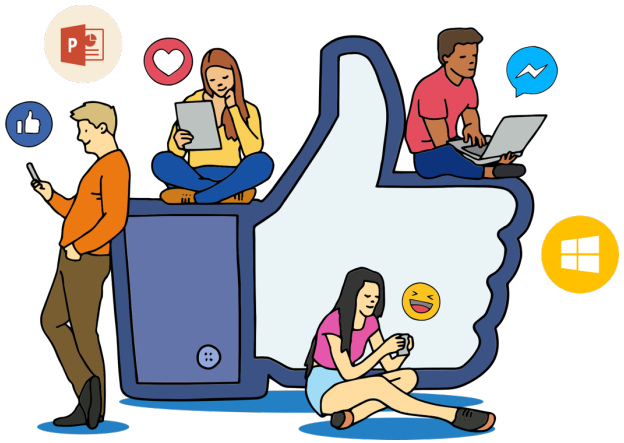
INFS7450 SOCIAL MEDIA ANALYTICS
Tutorial Week 4



Outlines

- Introduction to Project 1
  - Tasks
  - Requirements
  - Deliverables
  - Kaggle
- Knowledge Extension to Lecture 4
  - In-class quiz: computing node betweenness centrality
  - Shortest path algorithm: Dijkstra
- Coding Demo
  - Implementation of
    - PageRank
    - HITS
    - Edge betweenness
    - Node betweenness
- Q&A

Project 1— Introduction

Project 1 – Fast Computation of User Centrality Measures

Semester 1, 2024

Marks:	15 marks (15%)
Submission Due:	17 Apr 2024 16:00 (Brisbane Time)
Deliverables:	See deliverables part
How to submit:	Electronic submission via Blackboard

Goal: This project aims to implement a number of efficient algorithms to compute various centrality measures for user nodes such as PageRank and Betweenness. Students are required to finish this project individually.

Dataset: In this project, you will be working with publicly available Facebook social network data. The Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value. The data contains 4039 nodes, and 88234 edges in total. Each line of the data represents an undirected link starting from one node to another.

The dataset is available from UQ blackboard. See /Assessment/INFS7450 Project One.

```
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 10
0 11
0 12
0 13
0 14
```

Project 1—Task

Tasks:

1. Calculate the Betweenness Centrality for nodes in the Facebook dataset. (**8 marks**)
   Overview: write code to load the Facebook social network data and construct an undirected and unweighted graph. Based on the constructed graph, you are required to write a program to calculate the betweenness centralities for the graph vertices.
   Input: The provided Facebook social network data.
   Output: The top-10 nodes with the highest betweenness centralities.

2. Calculate PageRank Centrality for nodes in the Facebook dataset. (**7 marks**)
   Overview: write code to load the Facebook social network data and construct an undirected and unweighted graph. Based on the constructed graph, you are required to write a program to calculate the PageRank (with α=0.85,β=0.15) centralities for the graph vertices.
   Input: The provided Facebook social network data.
   Output: The top-10 nodes with the highest PageRank centralities.

Project 1-- Requirement

Requirements:

1. You may use third-party libraries, such as NetworkX to read, load and manipulate the Facebook network dataset. **However, you must write your own code to implement the function of node centrality calculation rather than using the third-part or built-in functions.** (You can use any functions in NetworkX other than the functions for centrality calculation.)
2. You can refer to the codes provided in the tutorial, but are not allowed to directly reuse or copy them.
3. You are not allowed to look at the code of any other student. **All submitted codes and reports will be subject to electronic plagiarism.**
4. **Failure to properly cite AI contributions is a form of academic misconduct.**

Project 1—Programming Languages

Programming Languages:

Python and NetworkX are recommended. However, you have your own choices of preferred programming languages including, but not limited to, Python, MATLAB, Java, C, C++, etc.

Project 1-- Deliverables

1. A report (.pdf). See the given appendix for an example template.
   Submit your report in the PDF format, not in the word format!

Report (Project One)

Student Name:
Student ID:
Student Email:

IMPORTANT: Submit your report in PDF. (please delete this line before submitting your report)

Here is the first paragraph, you can briefly tell us what you have done in this report.

Task 1

The content may include the following key points:

- In each task, briefly tell us how you calculate them, theory, formula, implementation, or other details you think necessary,
- then report your answer directly in tables, lists, figures or other kinds of forms that can clearly deliver your results.
- In the end, illustrate your opinions, analysis or findings.
- Maximum one page for each task (excluding the results)

Task 2

Same as required in Task 1.

Summary

Summarize your work, highlights and further discussions if applicable.

Reference

[1]. If you use/mention/quote some references, resources, or other tools, you can list them here.

Project 1--Deliverables

2. A source code file. For python users, organize your code in this way:

If you are using other programming languages, please organize your code in one file, and let me know how to run your code only via a simple click.

Only submit the source code in one file, don't submit the data file.

```python
import networkx as nx
import numpy as np


def construct_graph(graph_data):
    """
    Here is your code for constructing the graph;
    """
    pass

def between_centrality(graph):
    """
    Here is your code for calculating the node between centralities.
    """
    pass

def pagerank_centrality(graph):
    """
    Here is your code for calculating the node pagerank centralities.
    """
    pass

def main():
    """
    Here is your main function
    """
    pass

if __name__=="__main__":
    """
    Here you callback your code to load data, and generate the results

    Make sure I can run your code only via a click.
    """
```

Project 1--Deliverables

3. A result text file. The file must contain two lines of results. The first line is the top-10 nodes with the highest betweenness centralities. The second line is the top-10 nodes with the highest PageRank centralities. (each node id in each line should be separated by a space.) See the following picture as an example.

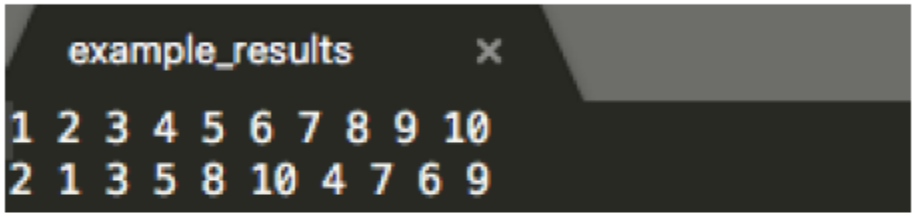


Figure 2. the example format of the results

Don't change the format! Your result file will be checked and marked via a script. If you change the format, your submitted results will not be accepted. Keep in mind the first line is the results of task 1 and the second line is the results of the task.
Don't mix the order.
Don't use other delimiters, only use a space.
Only report the top-10 node ids, don't fill in with node centrality scores!

Project 1--Submission

4. **Name all the submitted files by your student ID**. For example, 41234567.py for the source code, 41234567.txt for your submitted results, and 41234567.pdf for your report.
5. Submit one archive file with your student number as the file name (e.g.12345678.zip) with all the files mentioned above.

For example:
A student (with id 12345678) can submit his project as follows:
12345678.zip/
-------12345678.py
-------12345678.txt
-------12345678.pdf

Any submitted project which doesn't follow the above guidelines will be desk rejected without marking, which means you will get zero marks for the corresponding parts.

Project 1—Marking Criteria

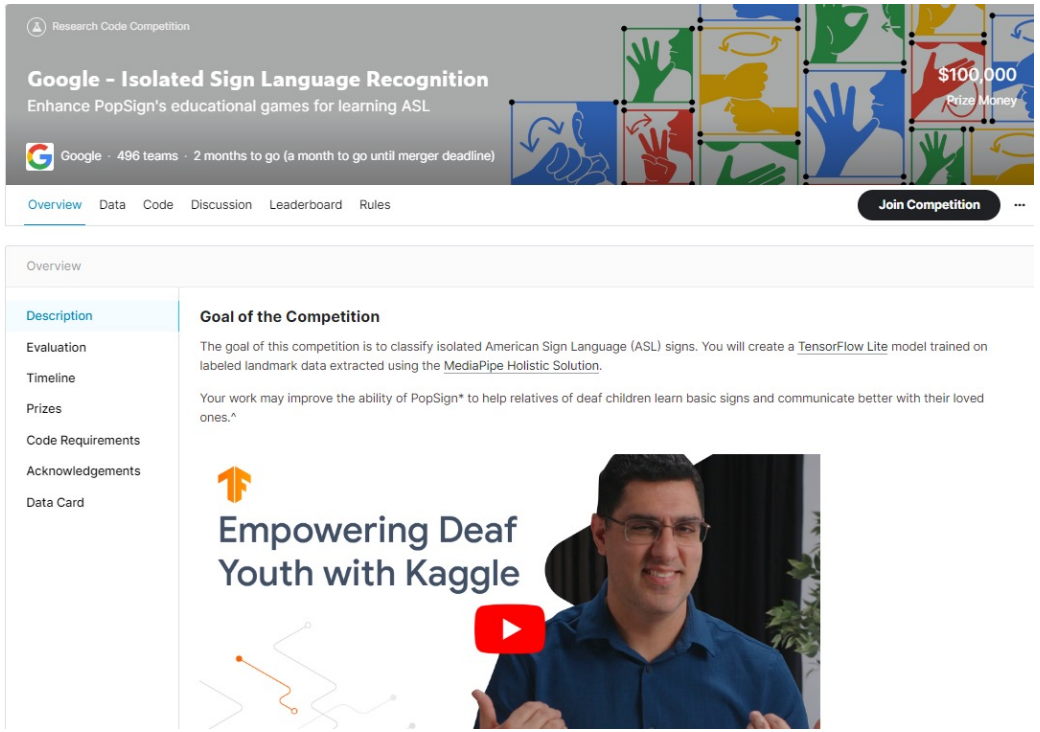
Marking criteria (Total marks: 15):

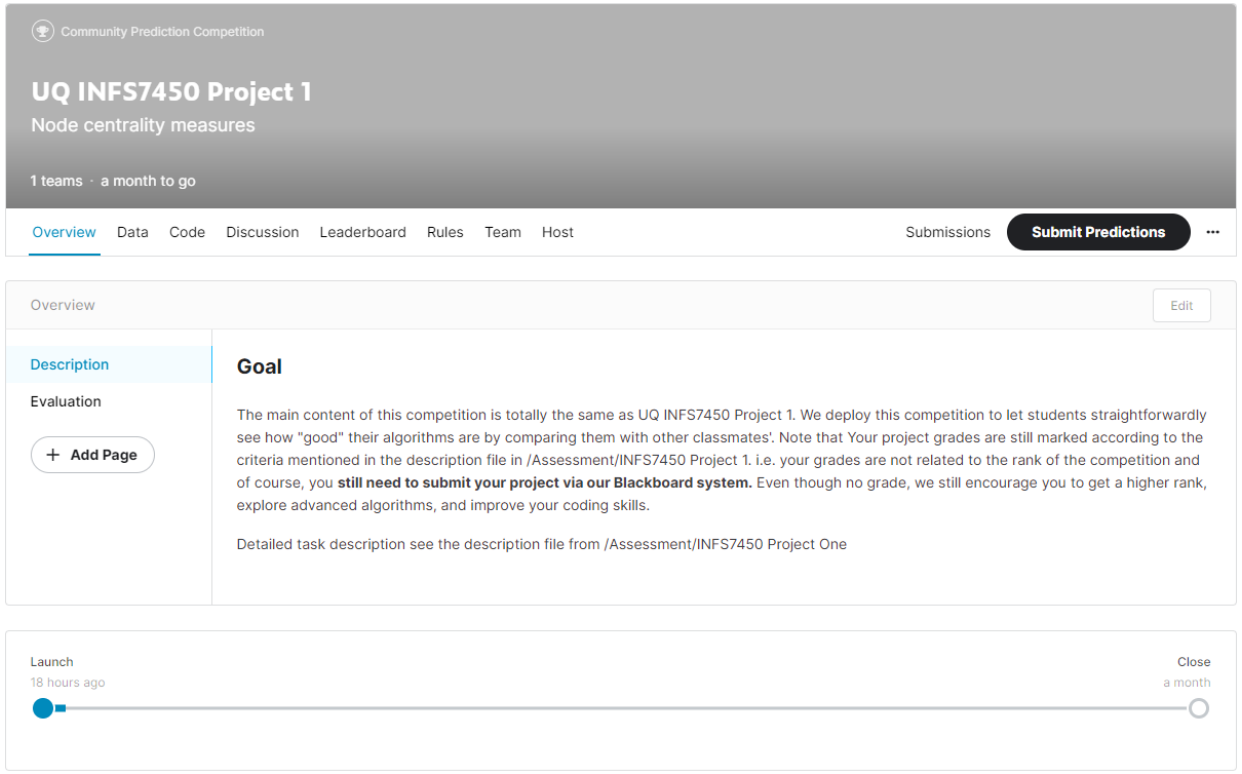
- Task 1: 8 marks = 3 marks (code) + 3 marks (results) + 2 marks (report)
- Task 2: 7 marks = 2 marks (code) + 3 marks (results) + 2 marks (report)
- Your results should be reproducible and your codes should be readable. If your codes cannot be executed or generate the results as reported, the corresponding marks for the code and results will be deducted.
- We will evaluate your submitted results by calculating the Jaccard Similarity between the submitted results and the ground truth. That means your mark for each task will be calculated by:

Result Mark = Jaccard Similarity (Submitted Results, Ground Truth) * 3

Project 1— Kaggle

- Kaggle is an online platform that provides many competitions about data science and machine learning;
- You can even win a prize of $50,000 if you achieve a good score; (It's Google's competition, not ours 😂)
- You can submit your results on Kaggle and see how "good" your algorithms are, and compete with other students.
- Submitting results to Kaggle is not compulsory, as your final scores are based only on the files you submit to the blackboard.





Project 1—Kaggle

- It is worth noting that the format of the results to be submitted to Kaggle is different from that on the blackboard.
- You don't need to change your algorithm, you just need to label each node as the following rules:

The only difference is that you need to give each node a label according to the following rules:

Suppose that your model predicts the set of nodes with the top-10 highest betweenness centraliries as

$$A = [a_1, a_2, \ldots, a_{10}]$$

.

Your model predicts the set of nodes with the top-10 highest PageRank centralities as

$$B = [b_1, b_2, \ldots, b_{10}]$$

.

The label $L(v_i)$ of node v_i is as follows:

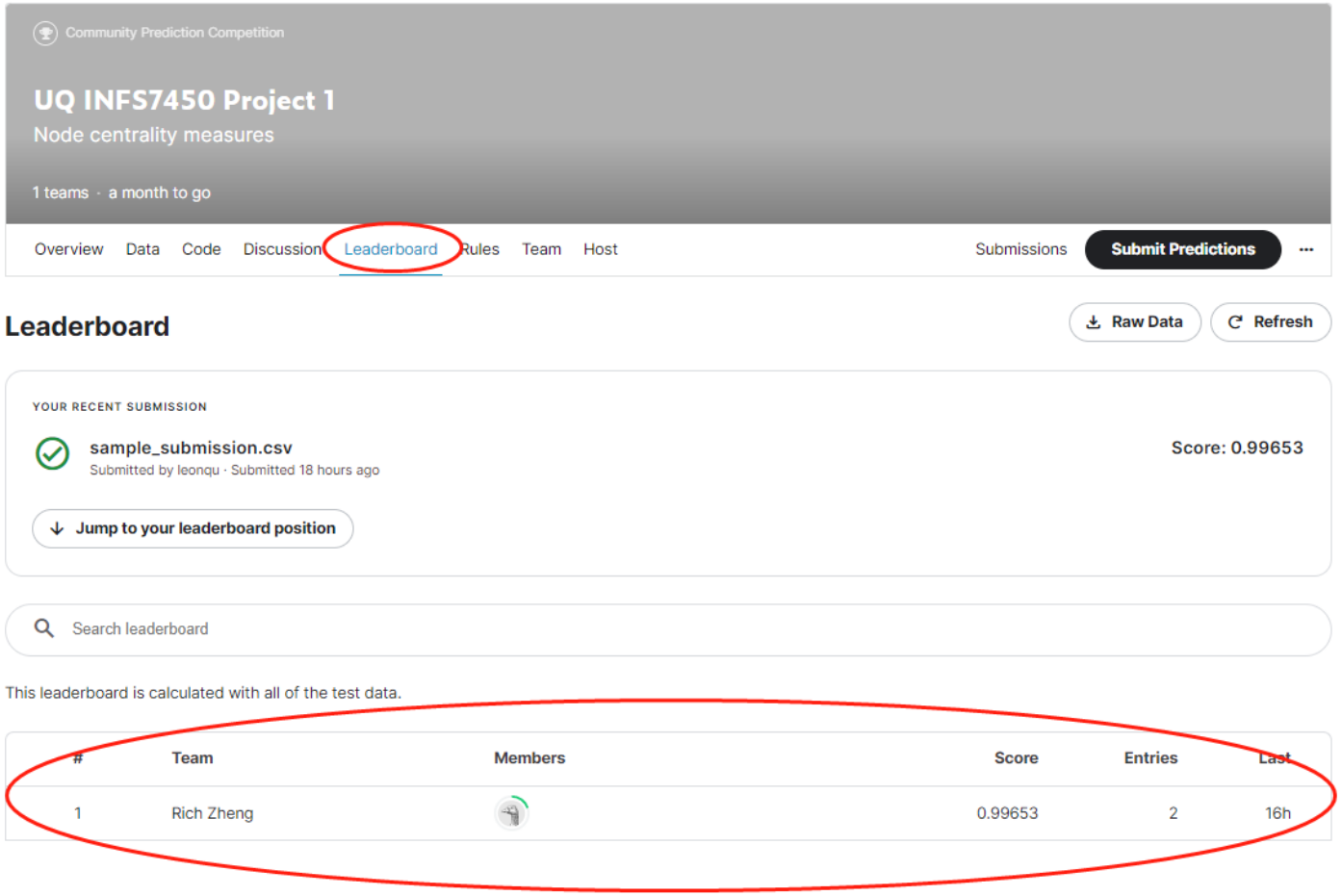
$$L(v_i) = \begin{cases} \mathbf{0}, & v_i \notin A \text{ and } v_i \notin B \\ \mathbf{1}, & v_i \in A \\ \mathbf{2}, & v_i \in B \\ \mathbf{3}, & v_i \in A \text{ and } v_i \in B \end{cases}$$

.

	A	B
1	Id	Label
2	0	1
3	1	1
4	2	1
5	3	1
6	4	0
7	5	1
8	6	2
9	7	3
10	8	3
11	9	0
12	10	0
13	11	1
14	12	1
15	13	1
16	14	1
17	15	1
18	16	1
19	17	1

Project 1—Kaggle

- You can view the rankings under Leaderboard.
- A very high score can be achieved by predicting all results to 0. (Thinking for yourself.)



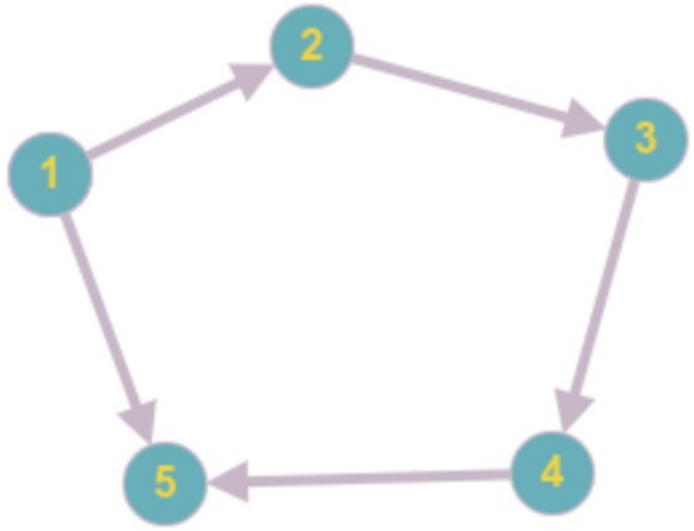
Section 2: Lecture Knowledge Extension

Section 2:

- In-class quiz: computing node betweenness centrality
- Shortest path algorithm
  - Dijkstra

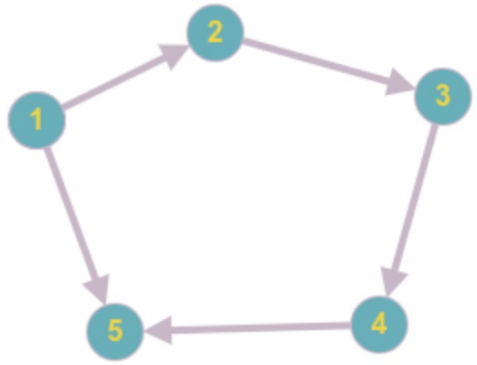
In-class Quiz (5 ~ 10 mins)

For the below graph, calculate the betweenness centrality of node 2



In-class Quiz

For the below graph, calculate the betweenness centrality of node 2



- **The betweenness centrality for node x is the probability that a shortest path passes through x**

$$C_b(v_i) = \sum_{s \neq t \neq v_i} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$$

σ_{st} The number of shortest paths from vertex s to t – *a.k.a.* information pathways

$\sigma_{st}(v_i)$ The number of **shortest paths** from s to t that pass through v_i

$$C_b(v_2) = \frac{1}{1} + \frac{1}{1} + \frac{0}{1} + \frac{0}{1} + \frac{0}{1} + \frac{0}{1} = 2$$

(s,t)	Number of shortest paths	Number of shortest paths that pass node 2
(1,3)	1 (1→2→3)	1 (1→2→3)
(1,4)	1 (1→ 2 → 3 → 4)	1 (1→ 2 → 3 → 4)
(1,5)	1 (1→ 5)	0
(3,1)	0	
(3,4)	1 (3 → 4)	0
(3,5)	1 (3 → 5)	0
(4,1)	0	
(4,3)	0	
(4,5)	1 (4 → 5)	0
(5,1)	0	
(5,3)	0	
(5,4)	0	

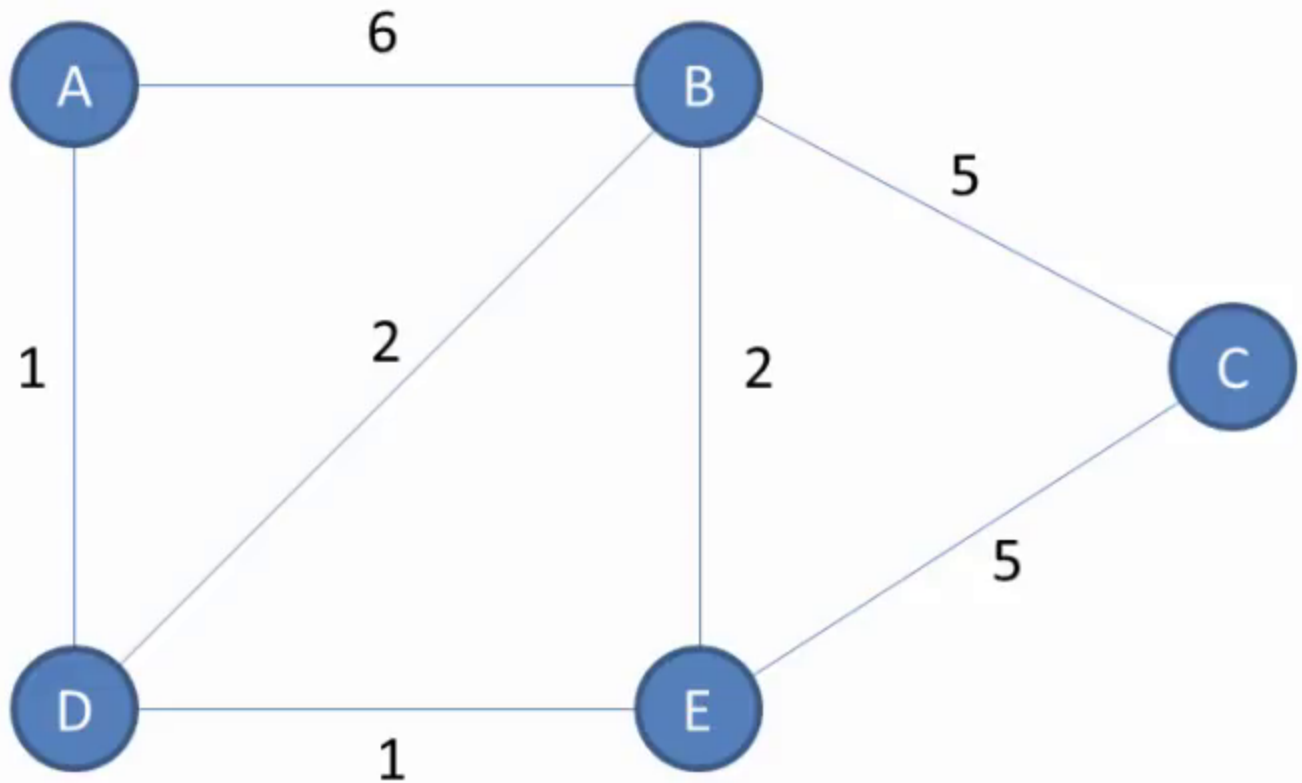
Computing Node Betweenness

- In betweenness centrality, we compute shortest paths between all pairs of nodes to compute the betweenness value.

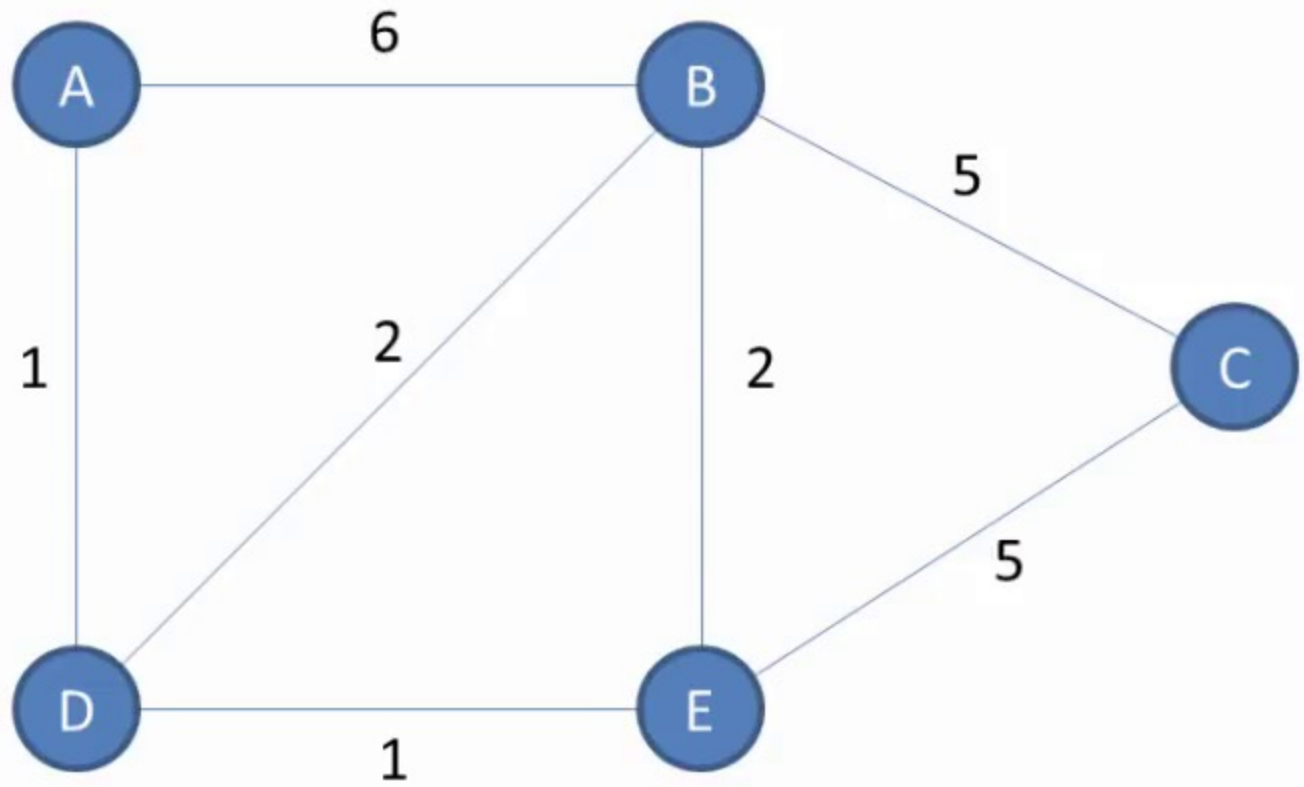
- **Trivial Solution**:
  - Use Dijkstra and run it $O(n)$ times
  - We get an $O(n^3)$ solution

- Better Solution?

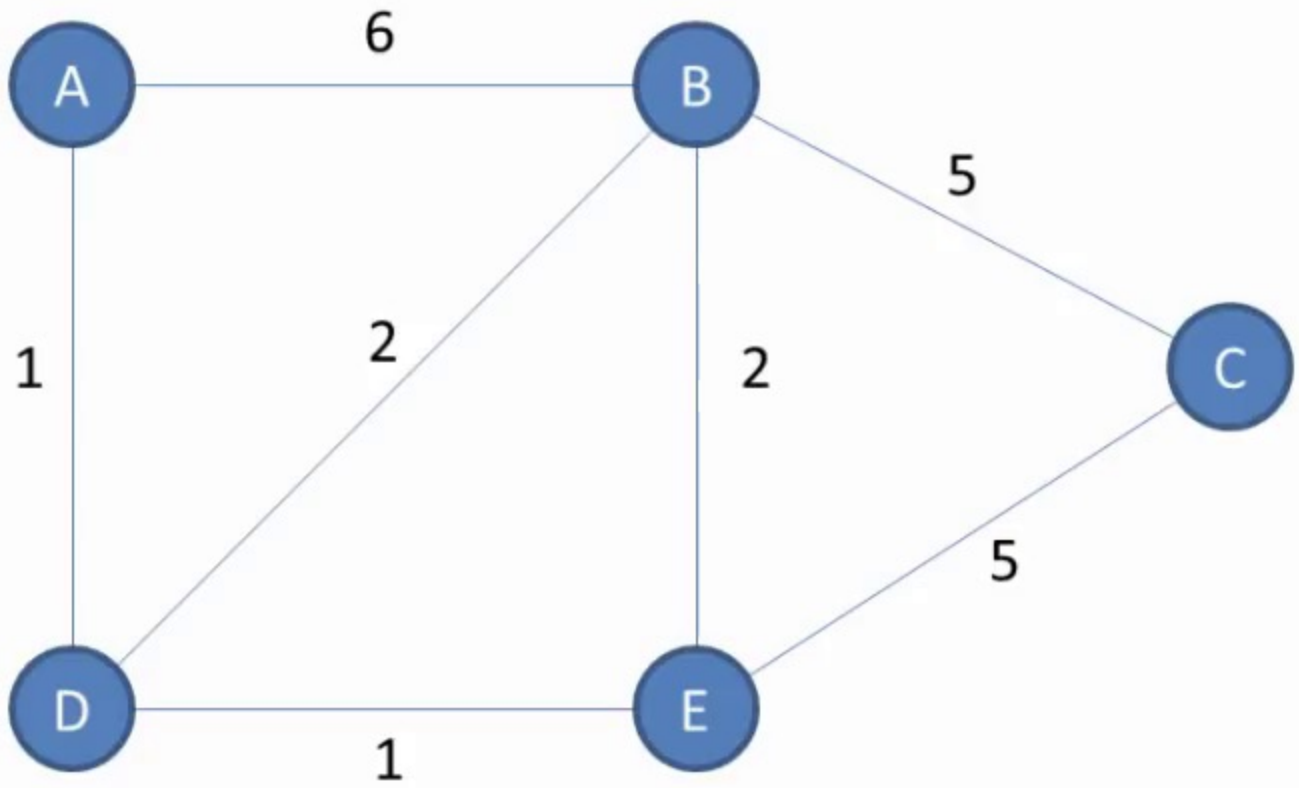
Dijkstra's Shortest Path Algorithm


A
B
C
D
E
6
5
1
2
2
5
1

Find the shortest path from vertex A to every other vertex

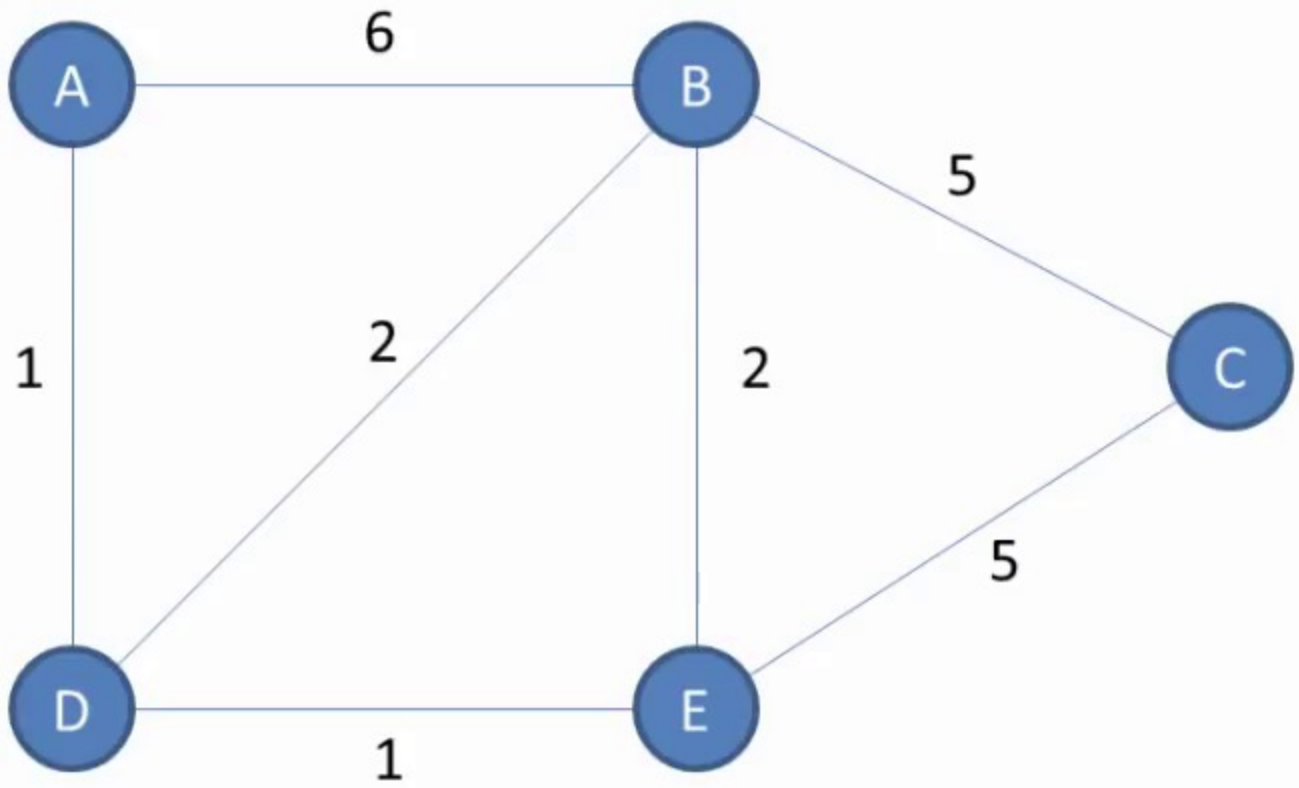


Find the shortest path from vertex A to every other vertex



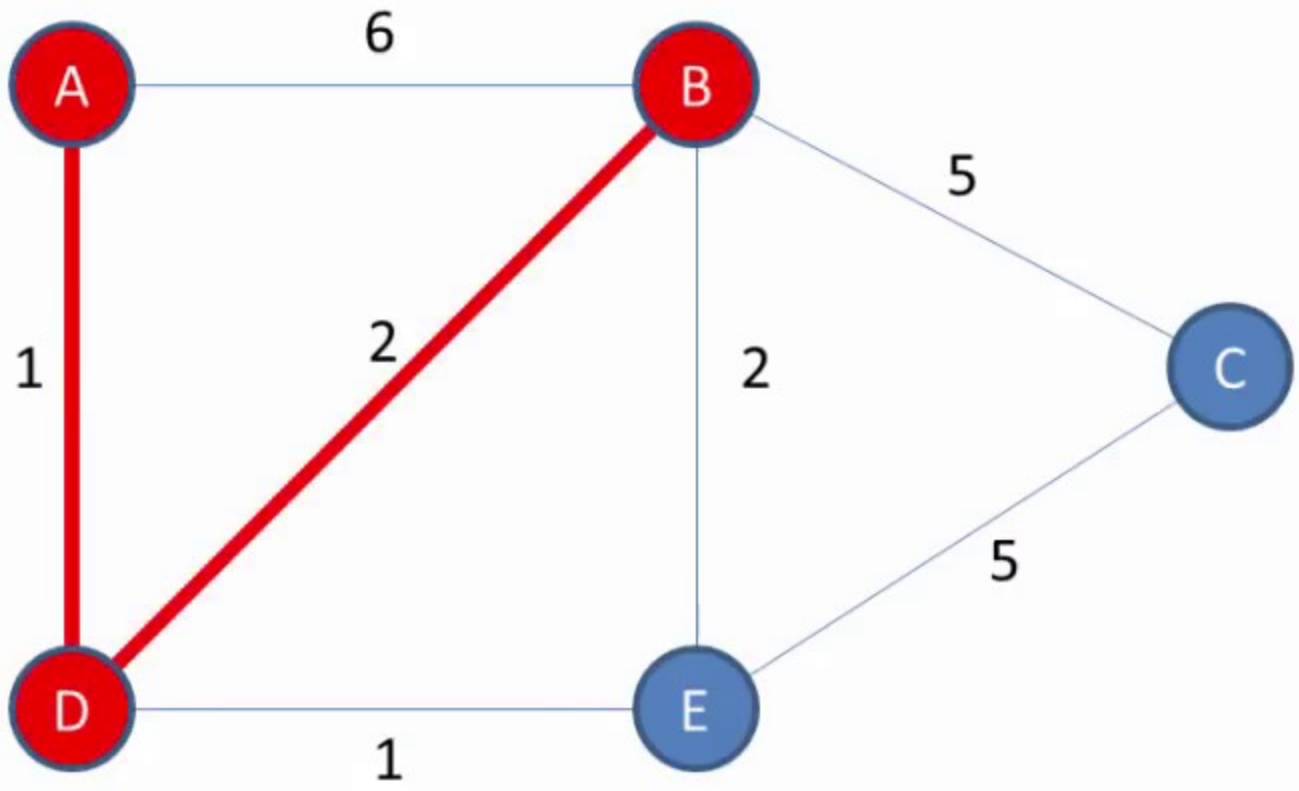
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We have the shortest distance from vertex A to every other vertex



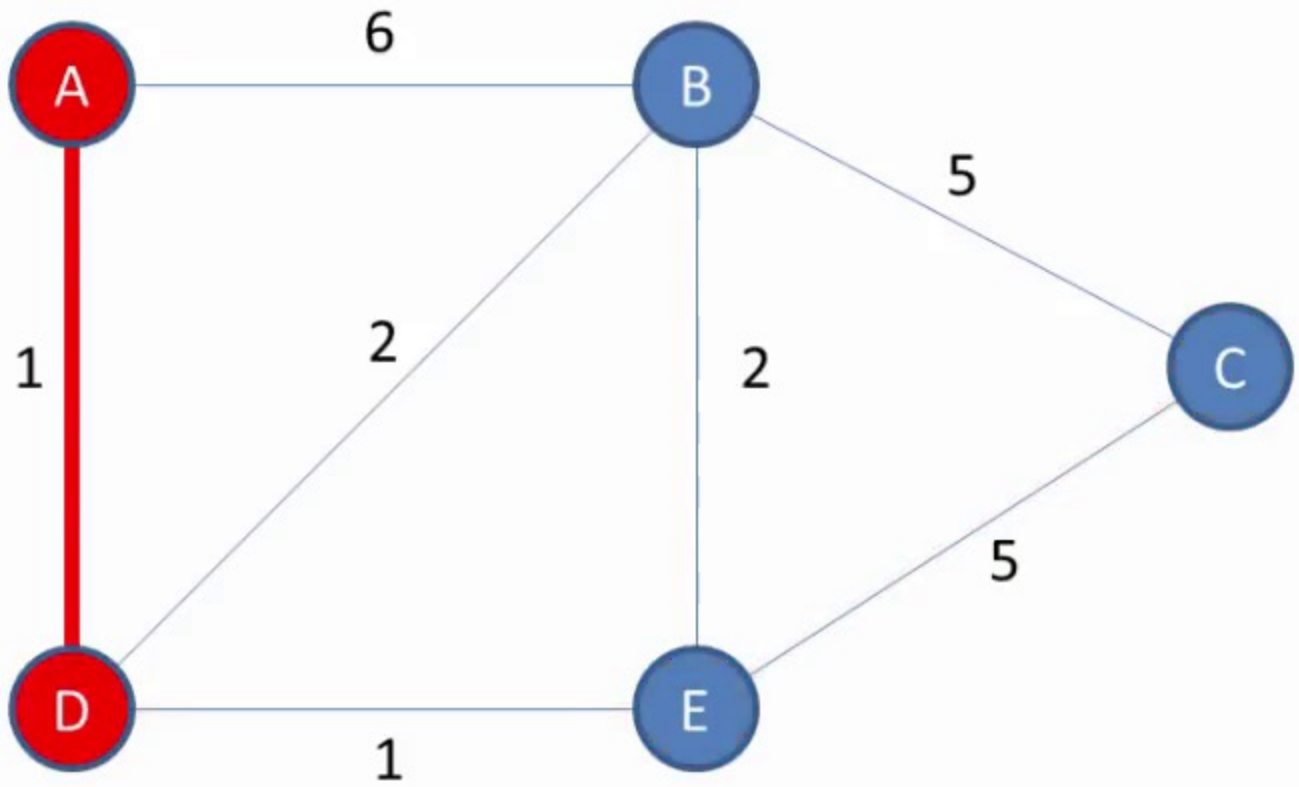
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to B



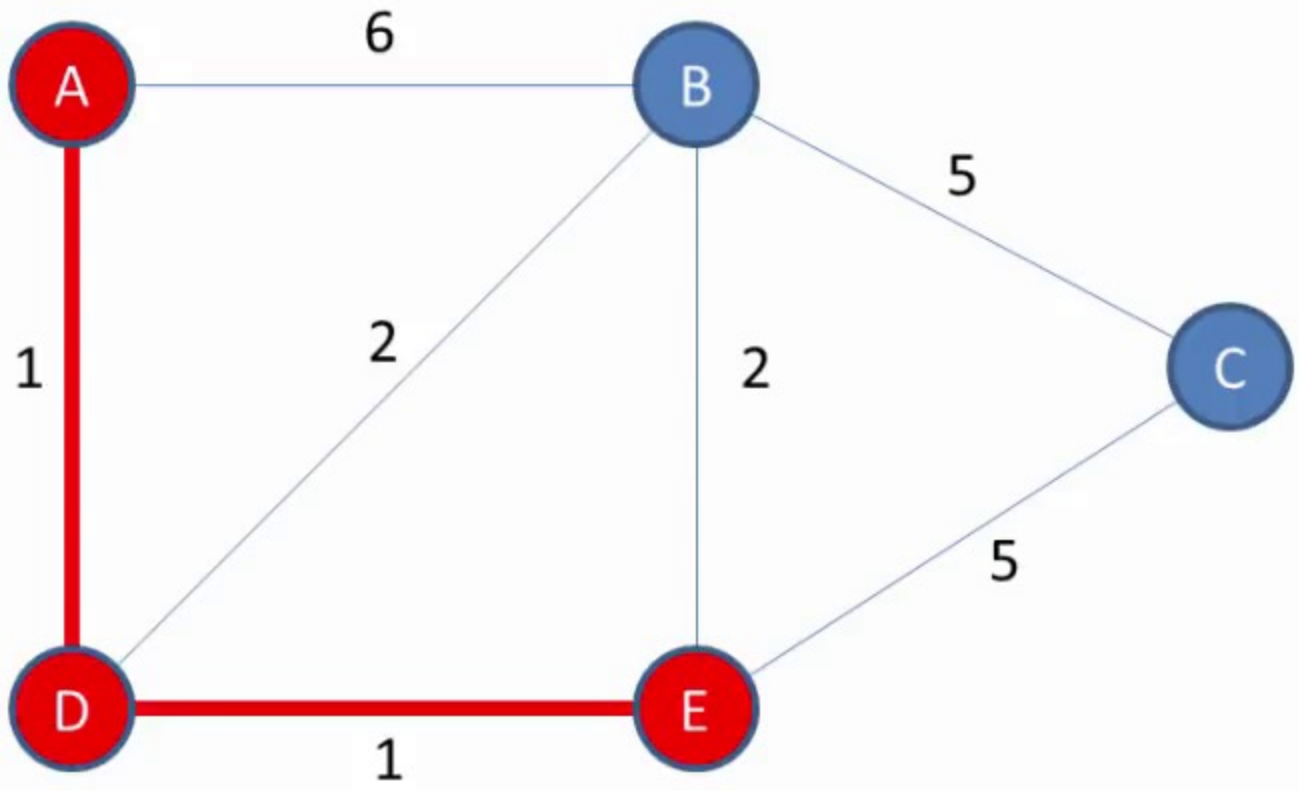
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to D



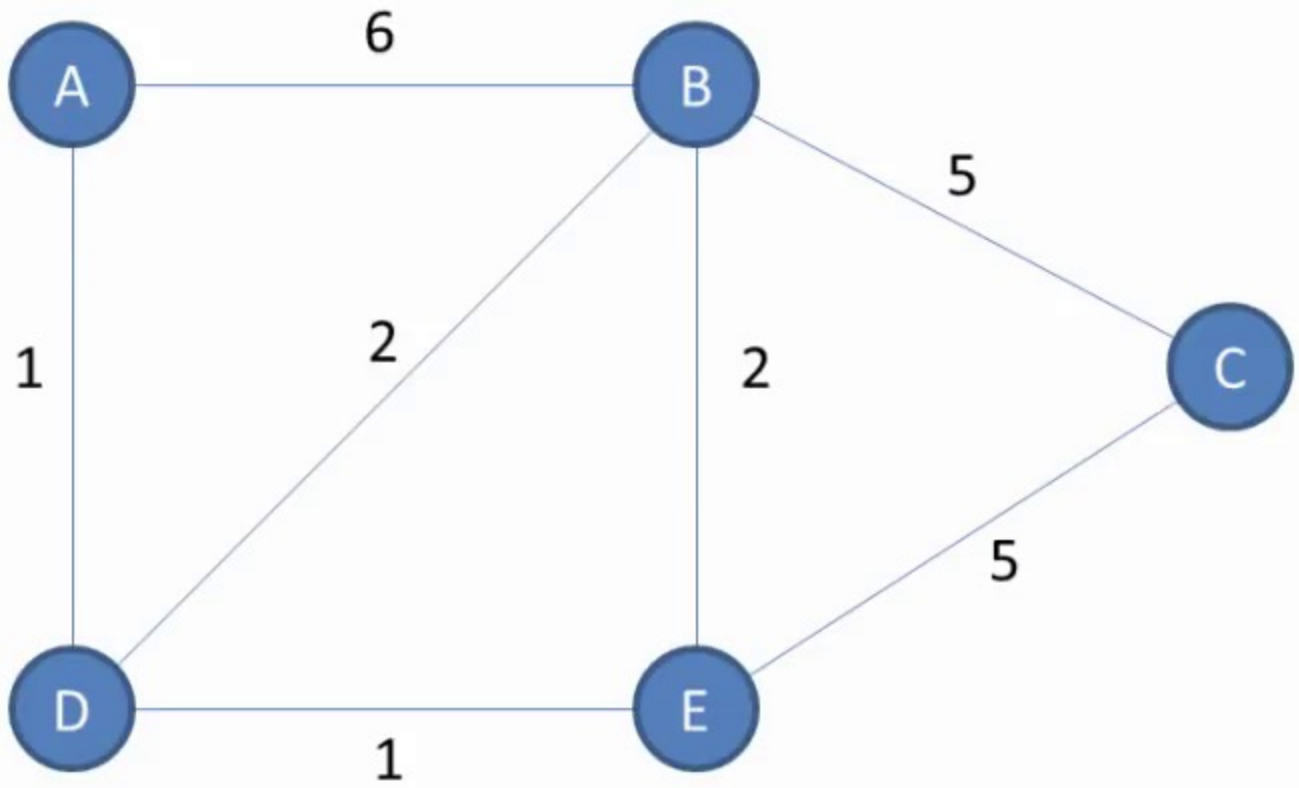
vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

A to E



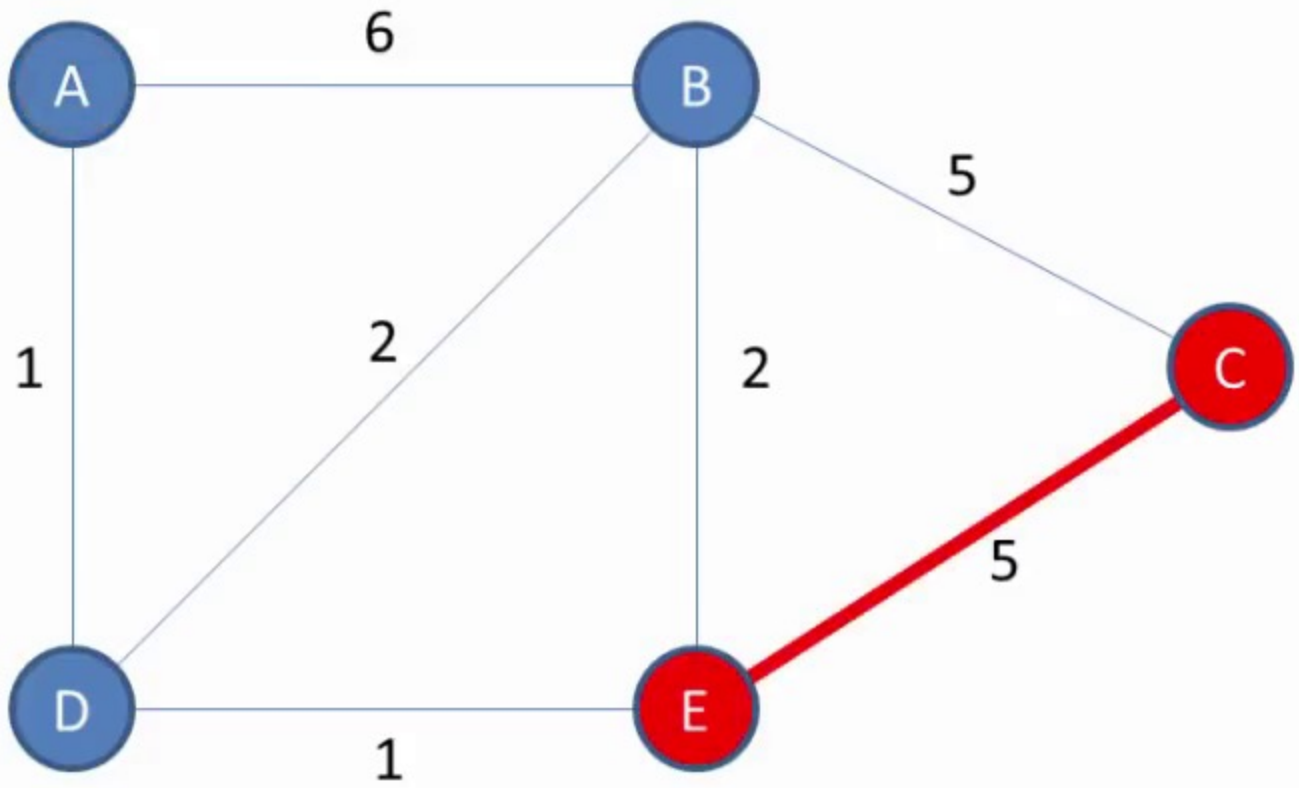
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We also have the shortest sequence of vertices from A to every other vertex



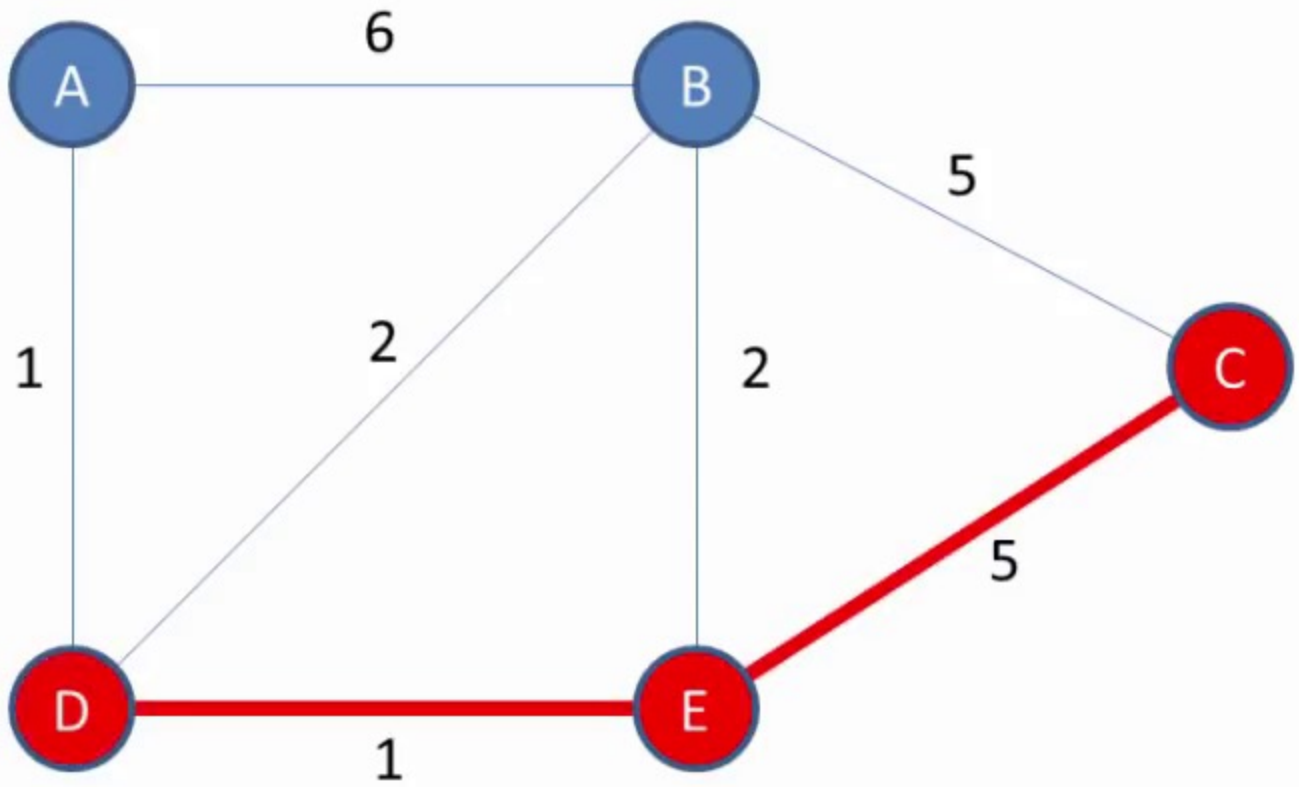
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at C via E Path = E → C



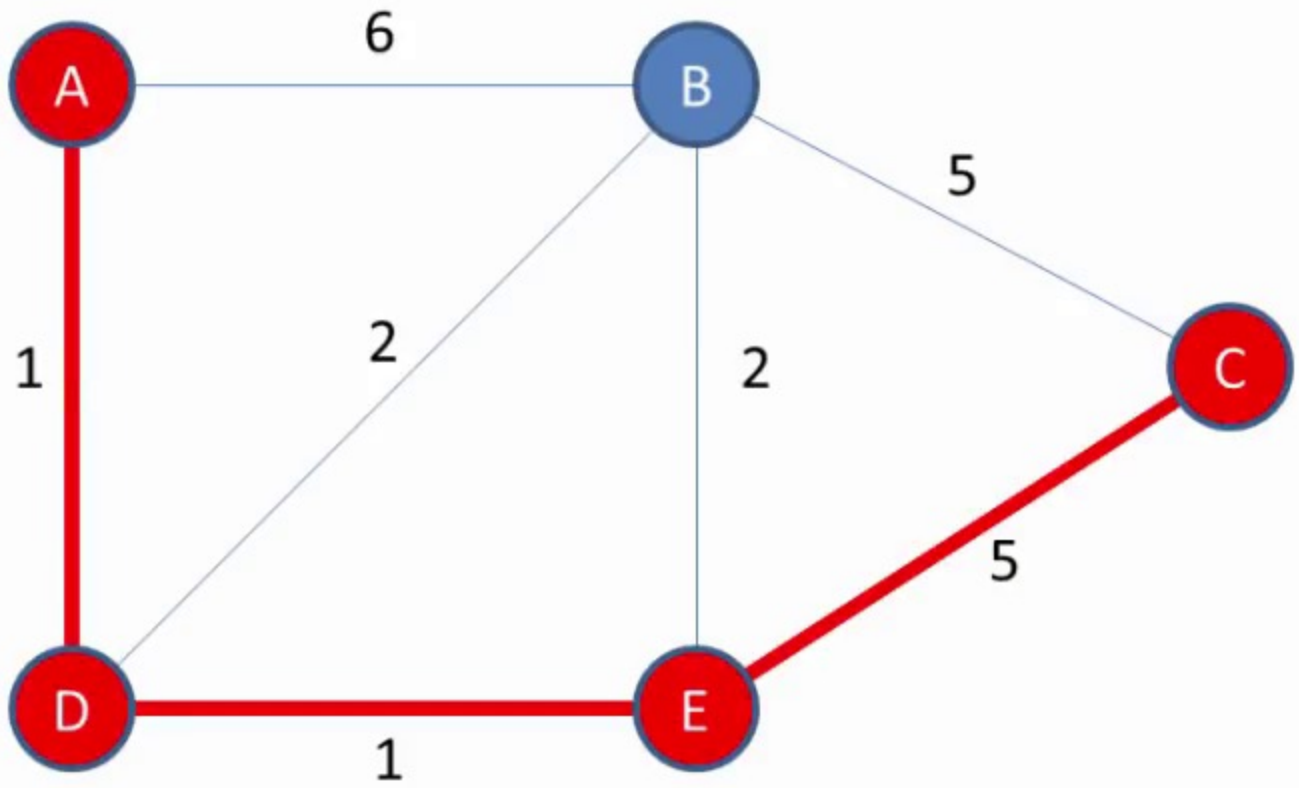
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at E via D Path = D → E → C



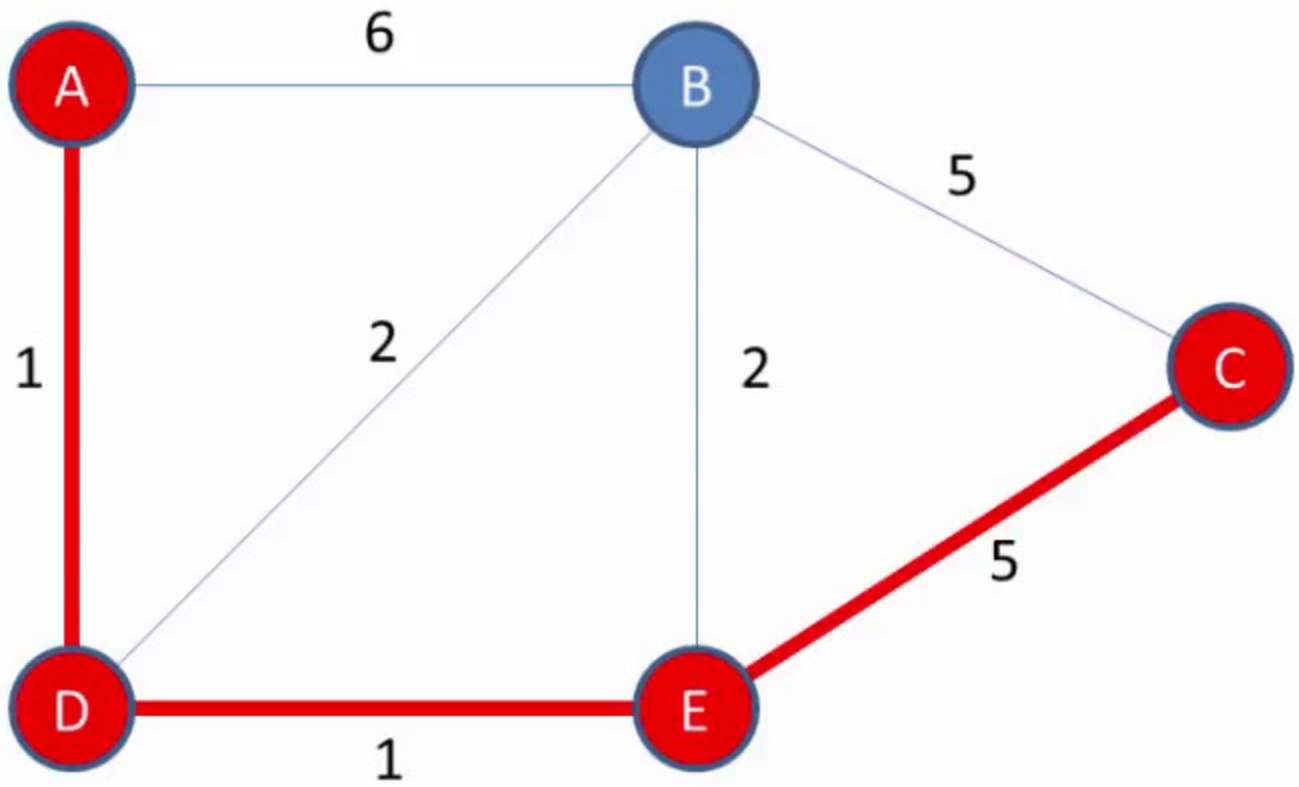
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

We arrived at D via A Path = A → D → E → C

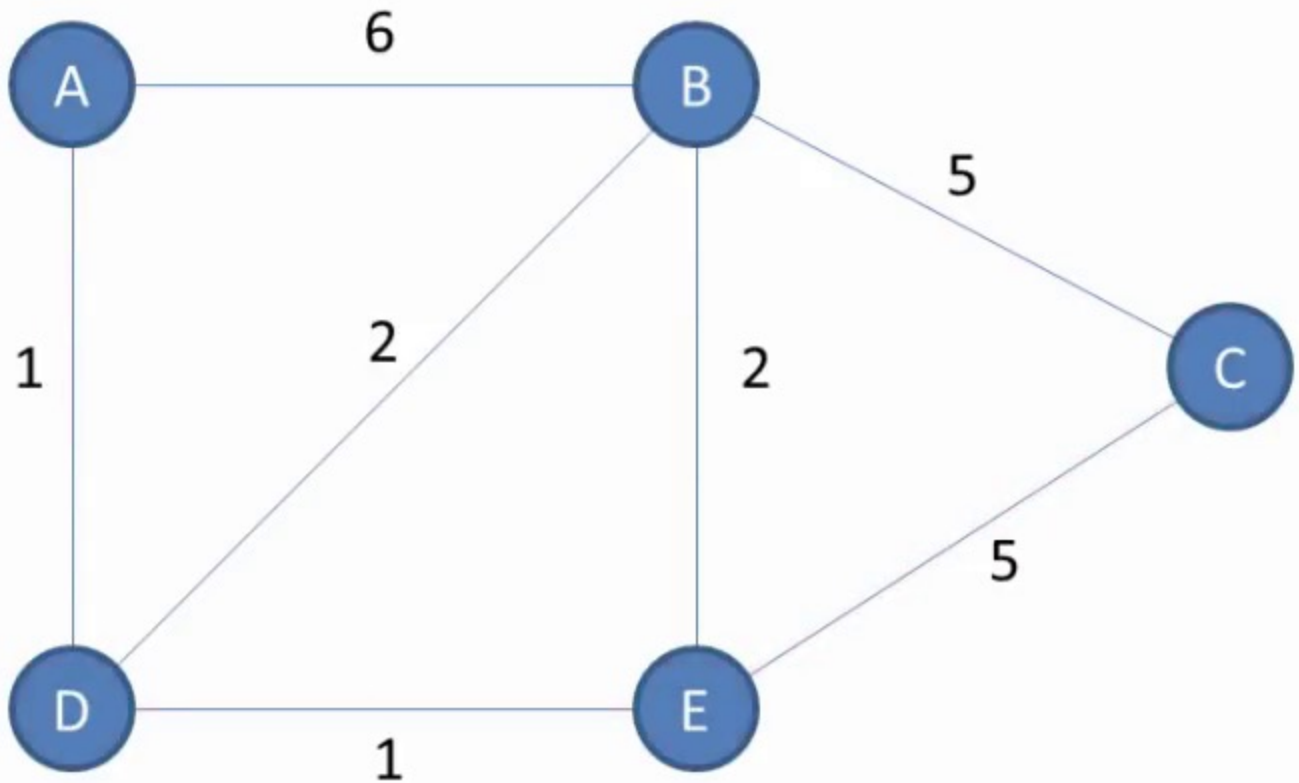


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Path = A → D → E → C

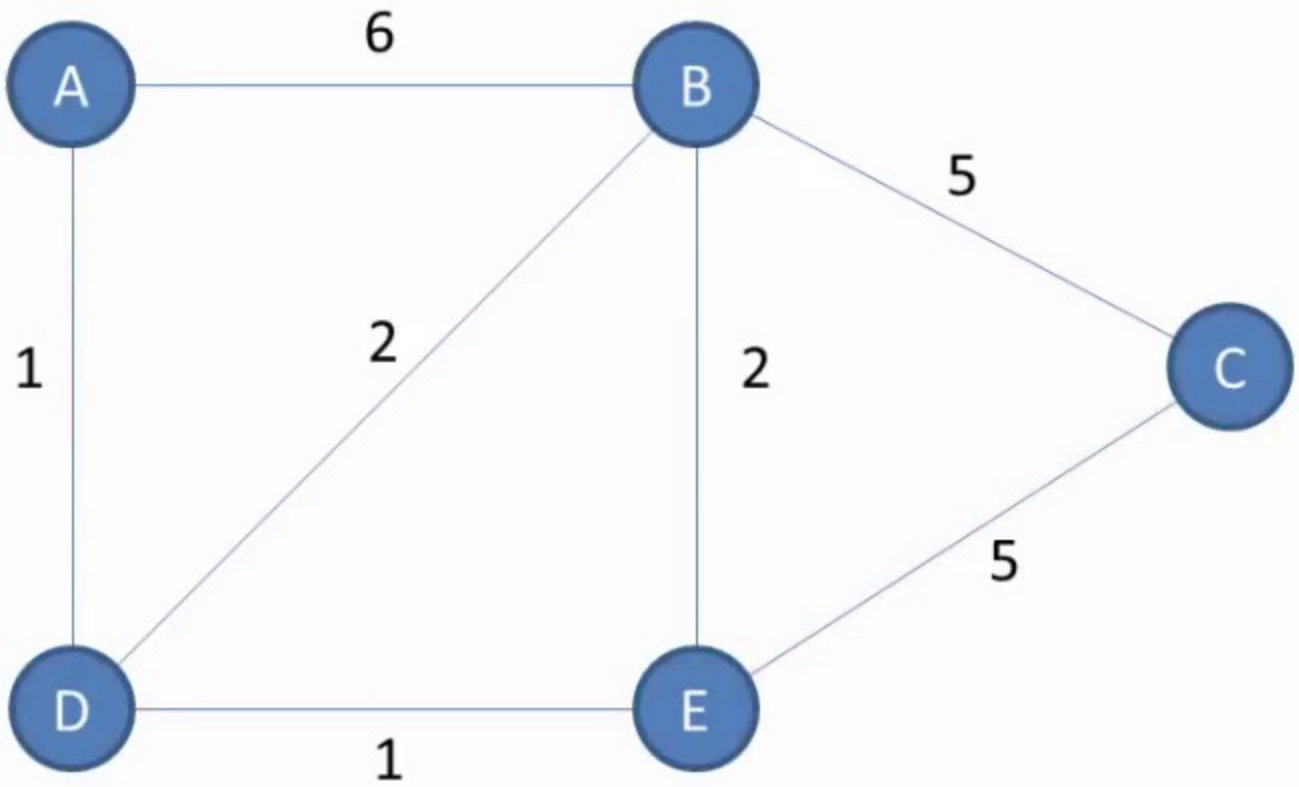


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

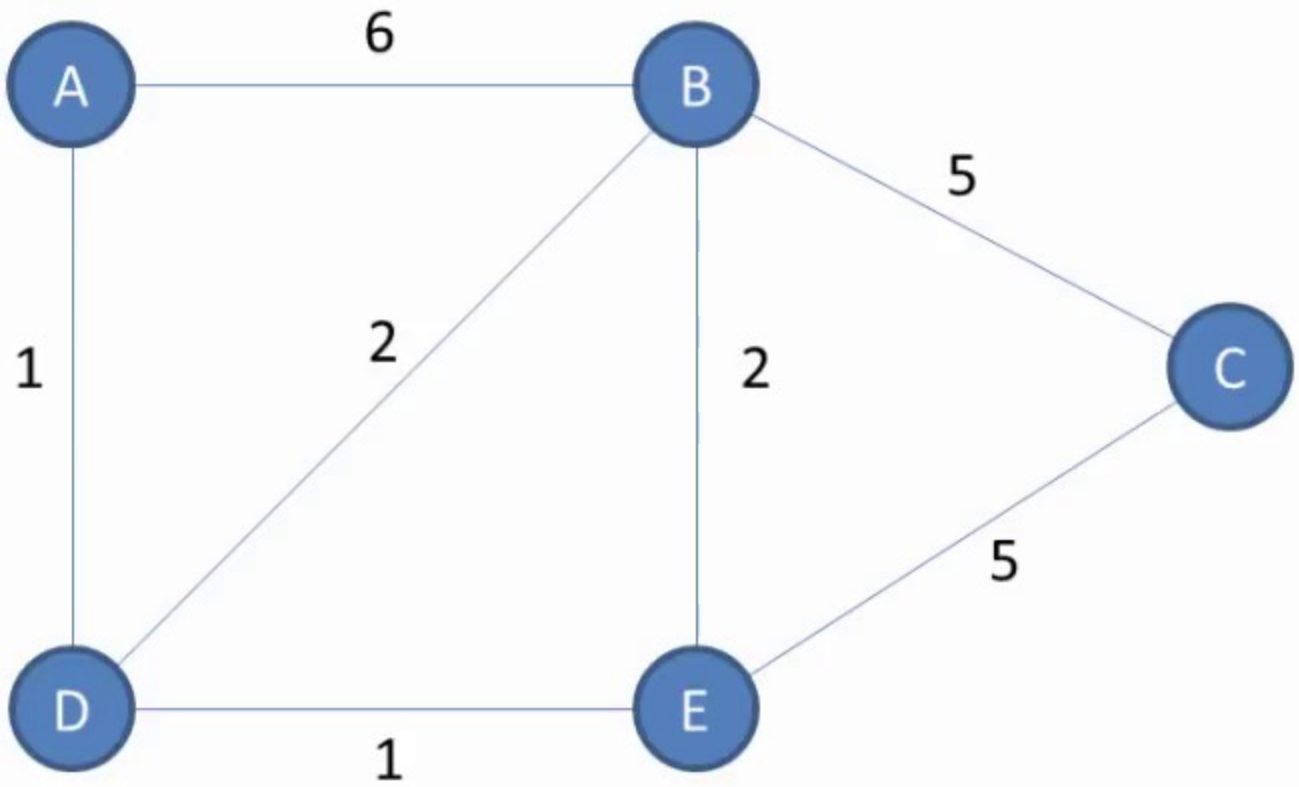


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Shortest Path Algorithm



Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

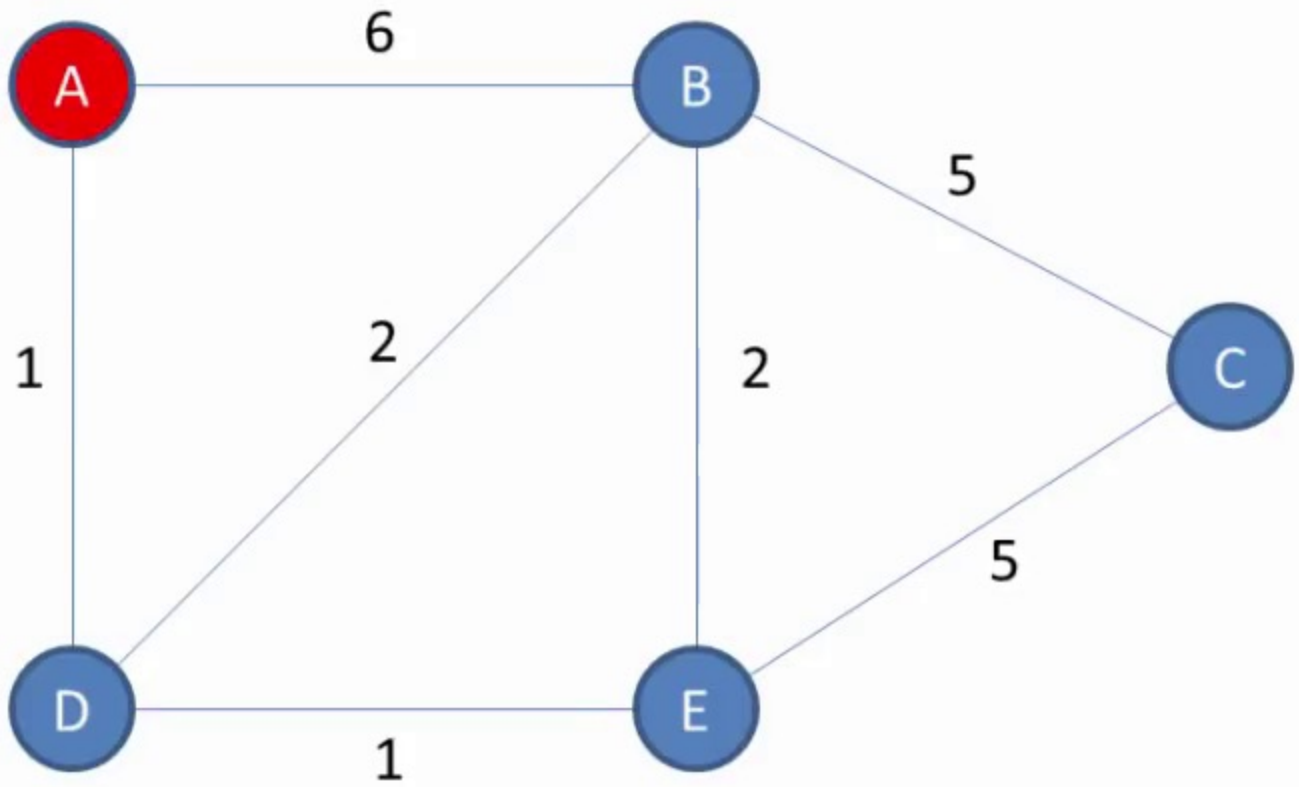


Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ]

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

Consider the start vertex, A

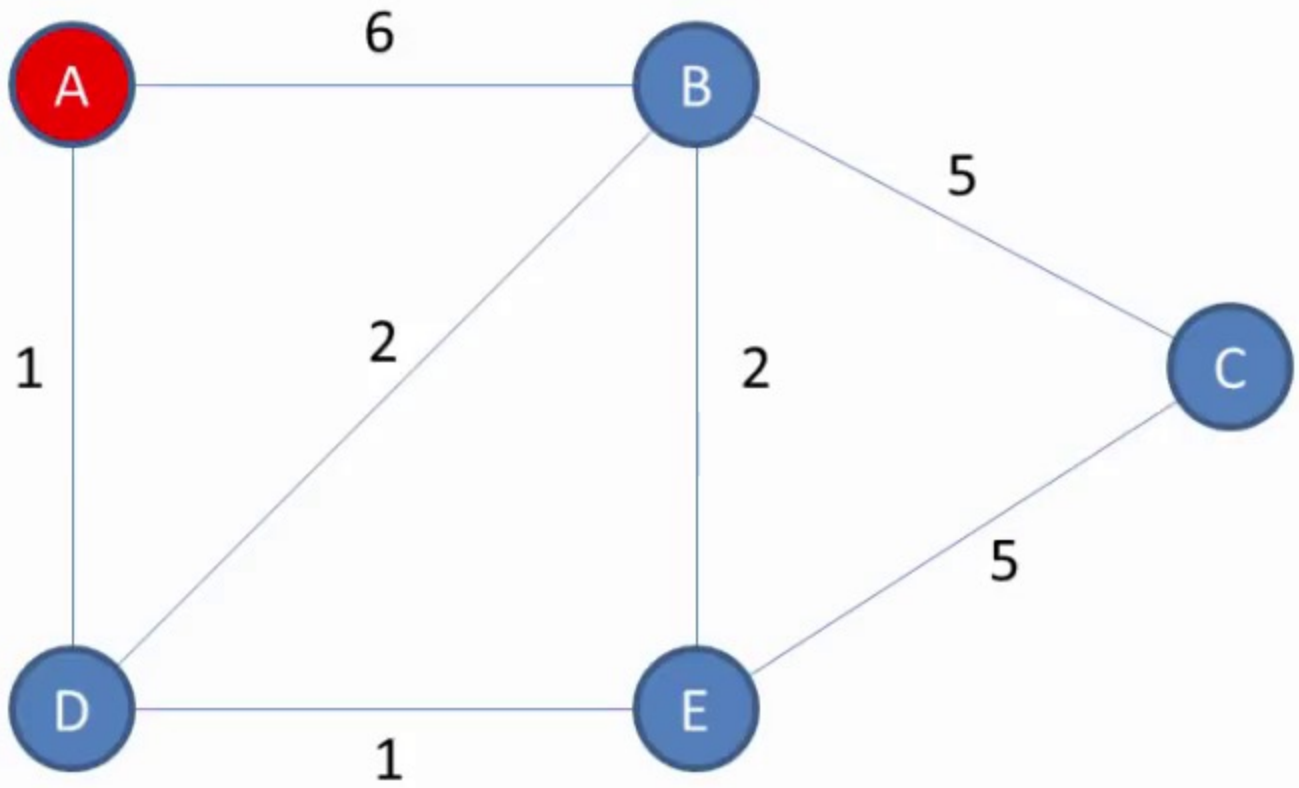


Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ]

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

Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)

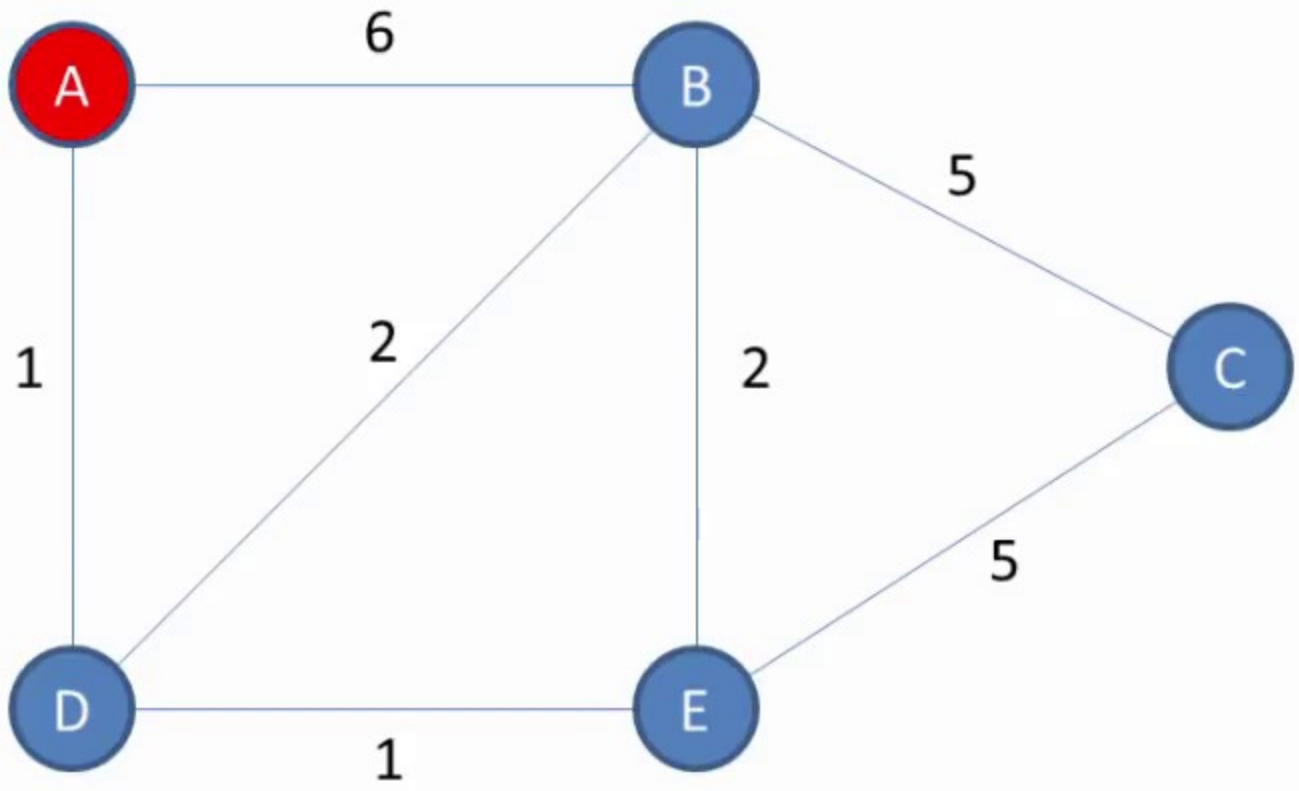


Vertex	Shortest distance from A	Previous vertex
A		
B		
C		
D		
E		

Visited = [ ]

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

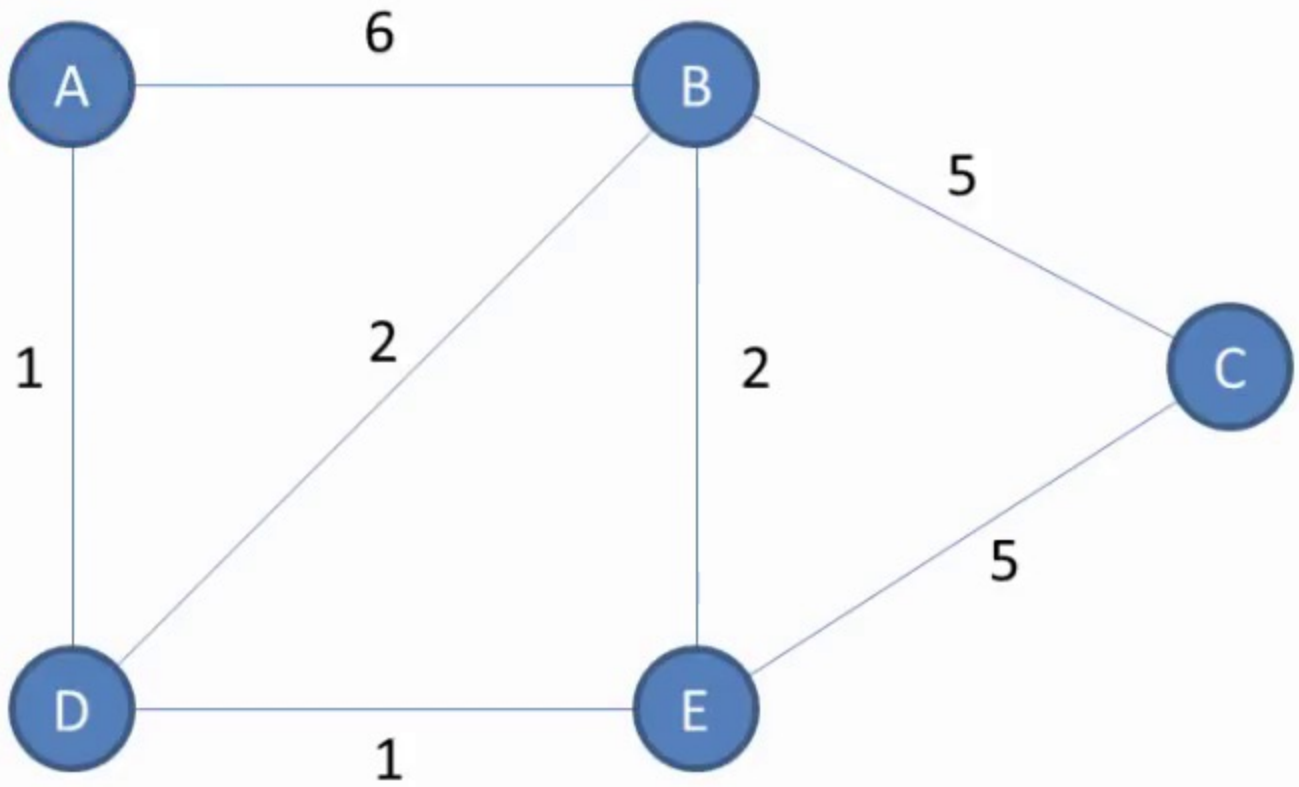
Consider the start vertex, A
Distance to A from A = 0
Distances to all other vertices from A are unknown, therefore ∞ (infinity)



Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

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



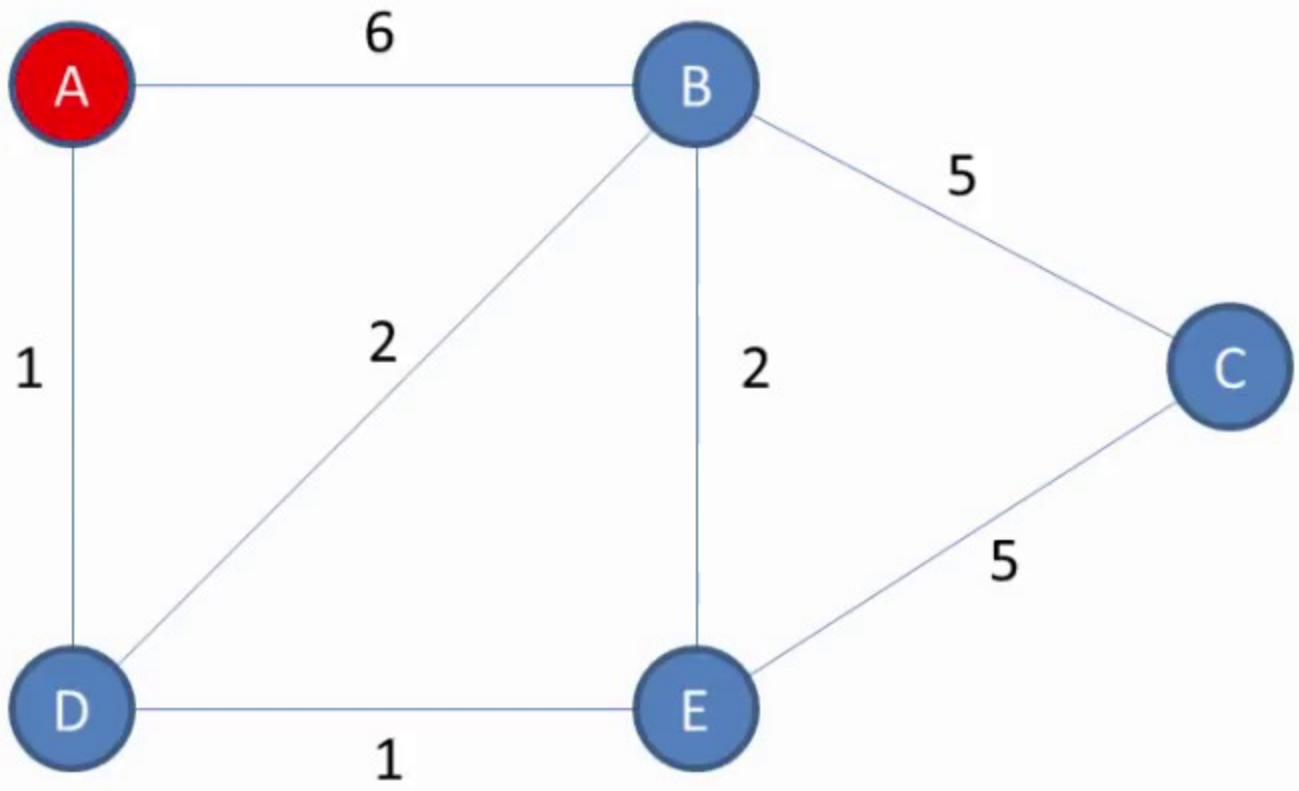
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

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

Visit the unvisited vertex with the smallest known distance from the start vertex

First time around, this is the start vertex itself, A



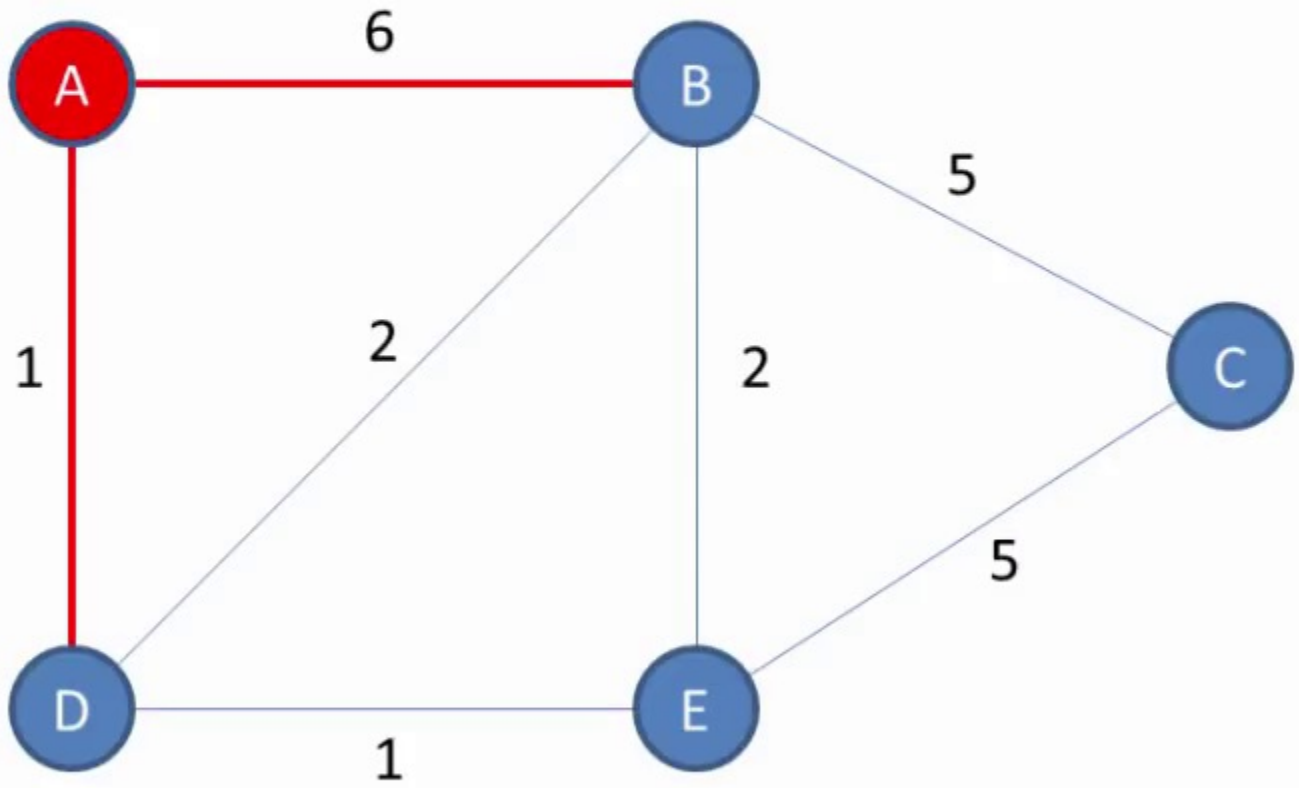
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

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

For the current vertex, examine its unvisited neighbours

We are currently visiting A and its unvisited neighbours are B and D

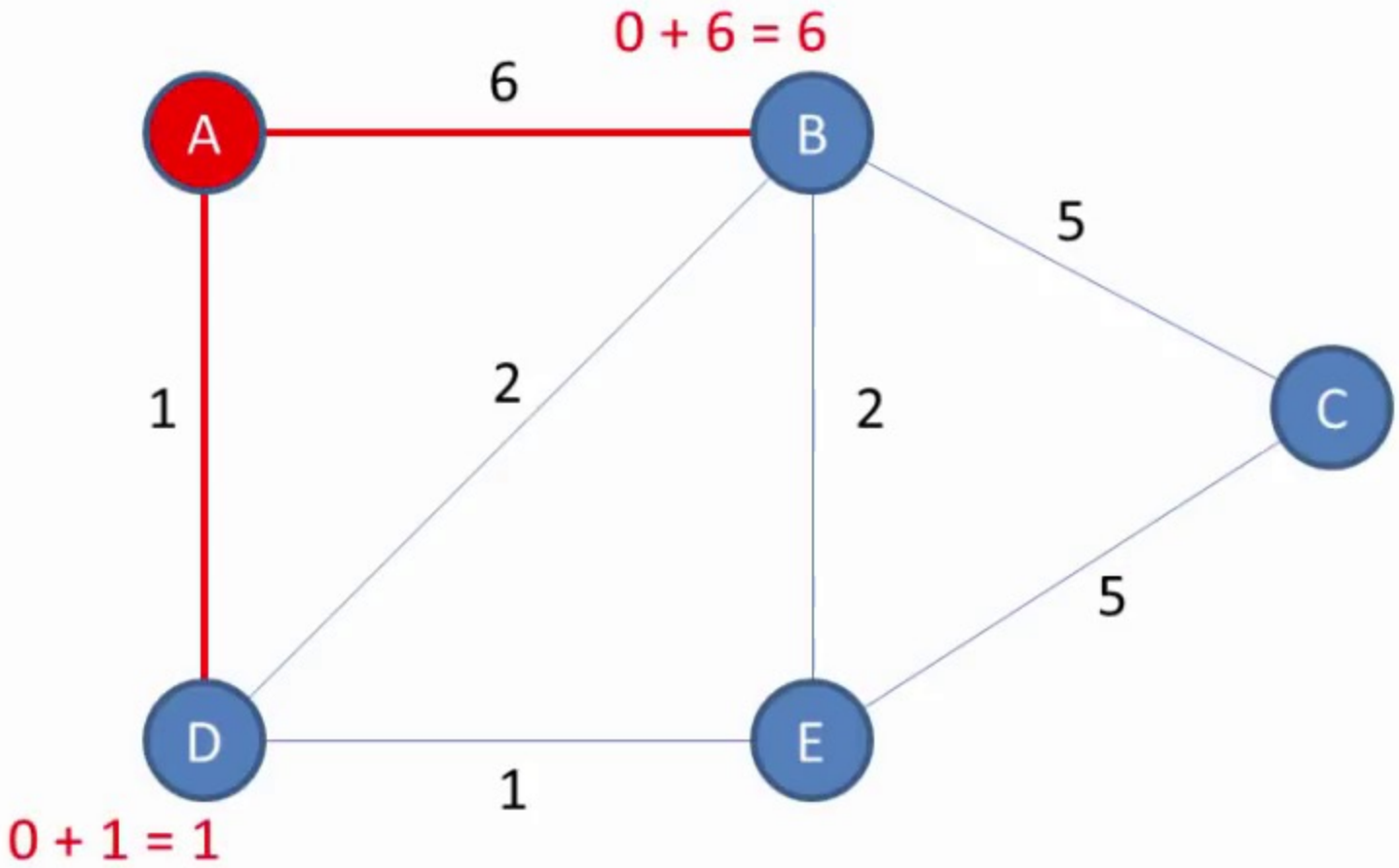


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

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

For the current vertex, calculate the distance of each neighbour from the start vertex

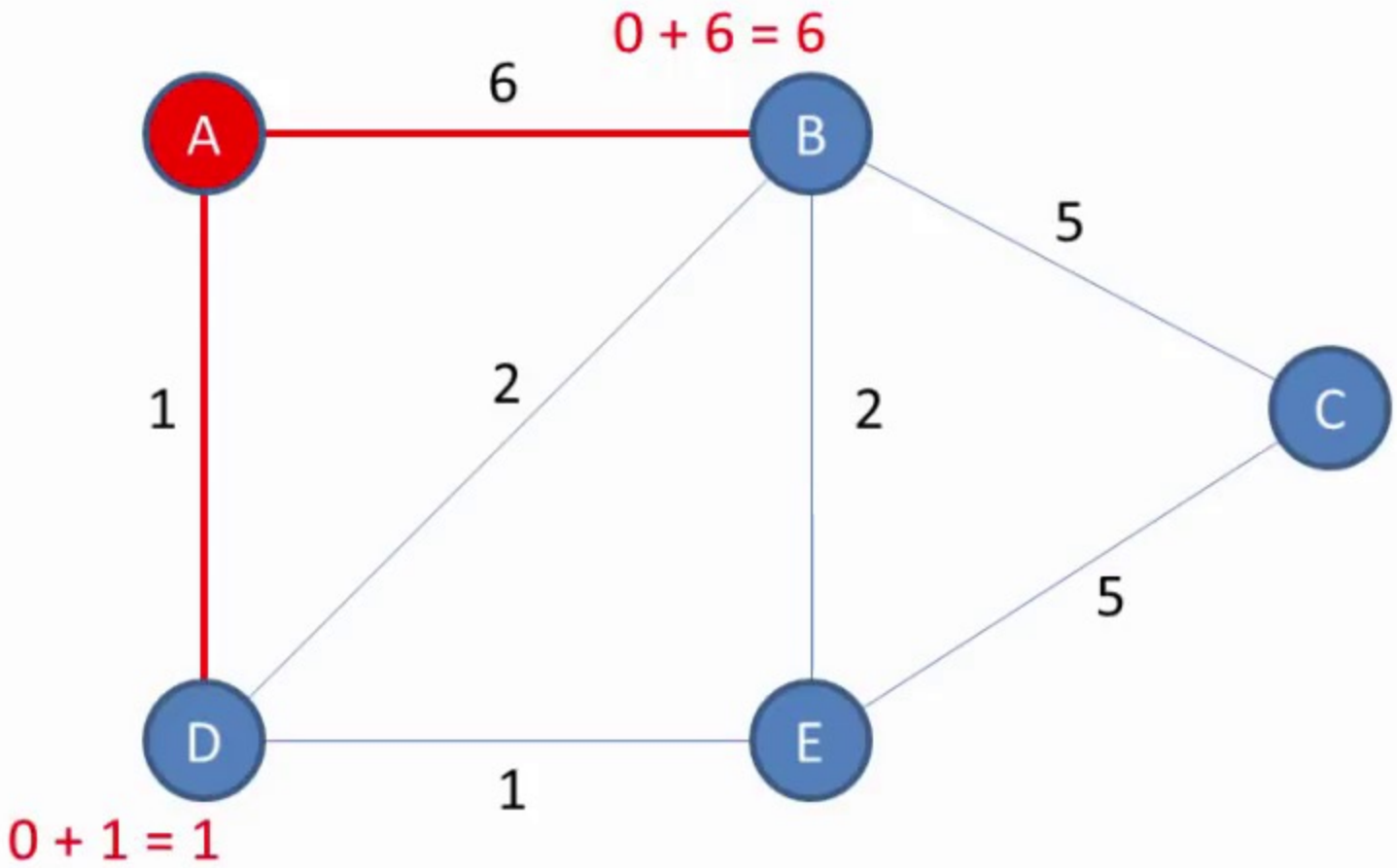


Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

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

If the calculated distance of a vertex is less than the known distance, update the shortest distance



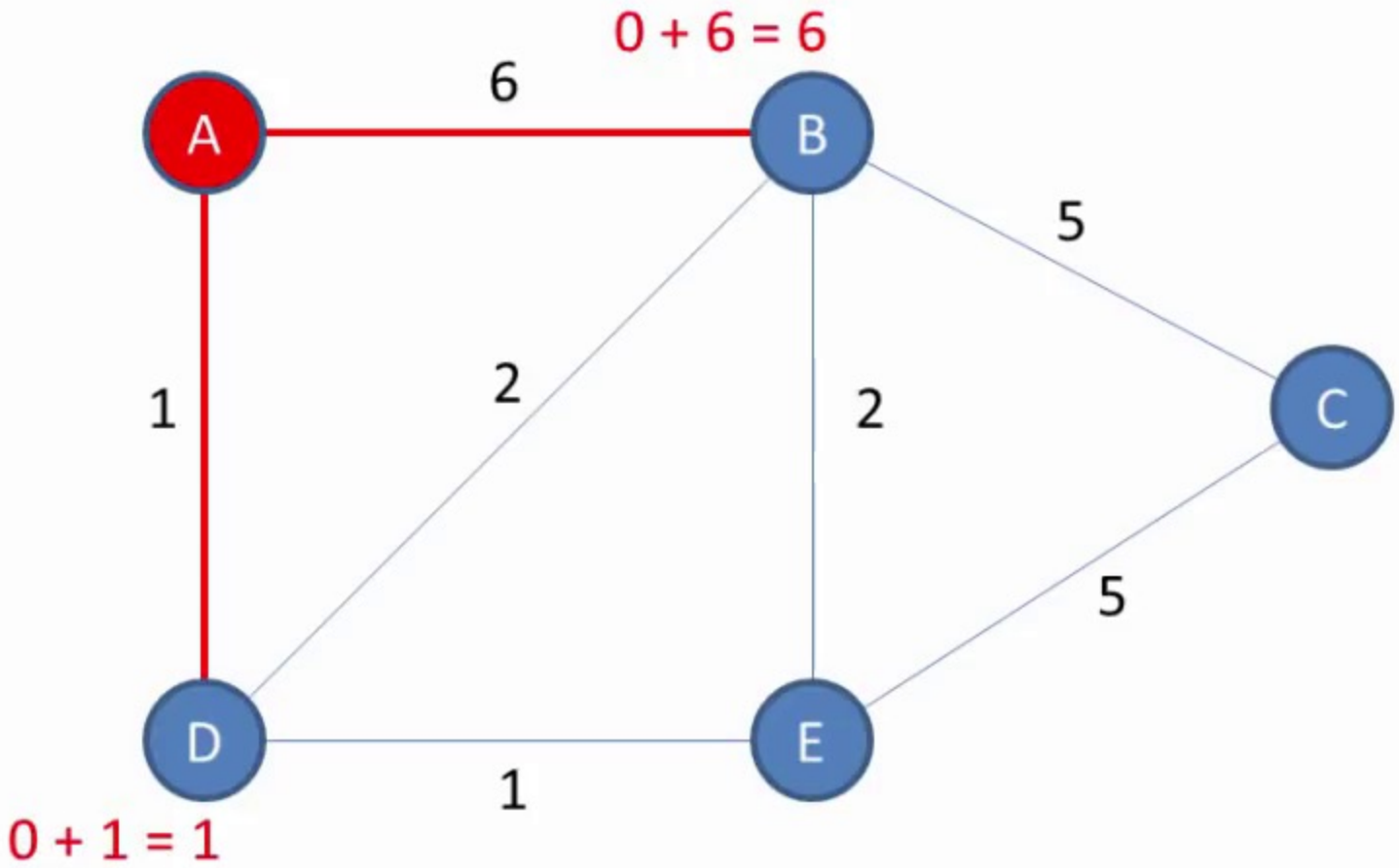
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Visited = [ ]

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

Update the previous vertex for each of the updated distances

In this case we visited B and D via A

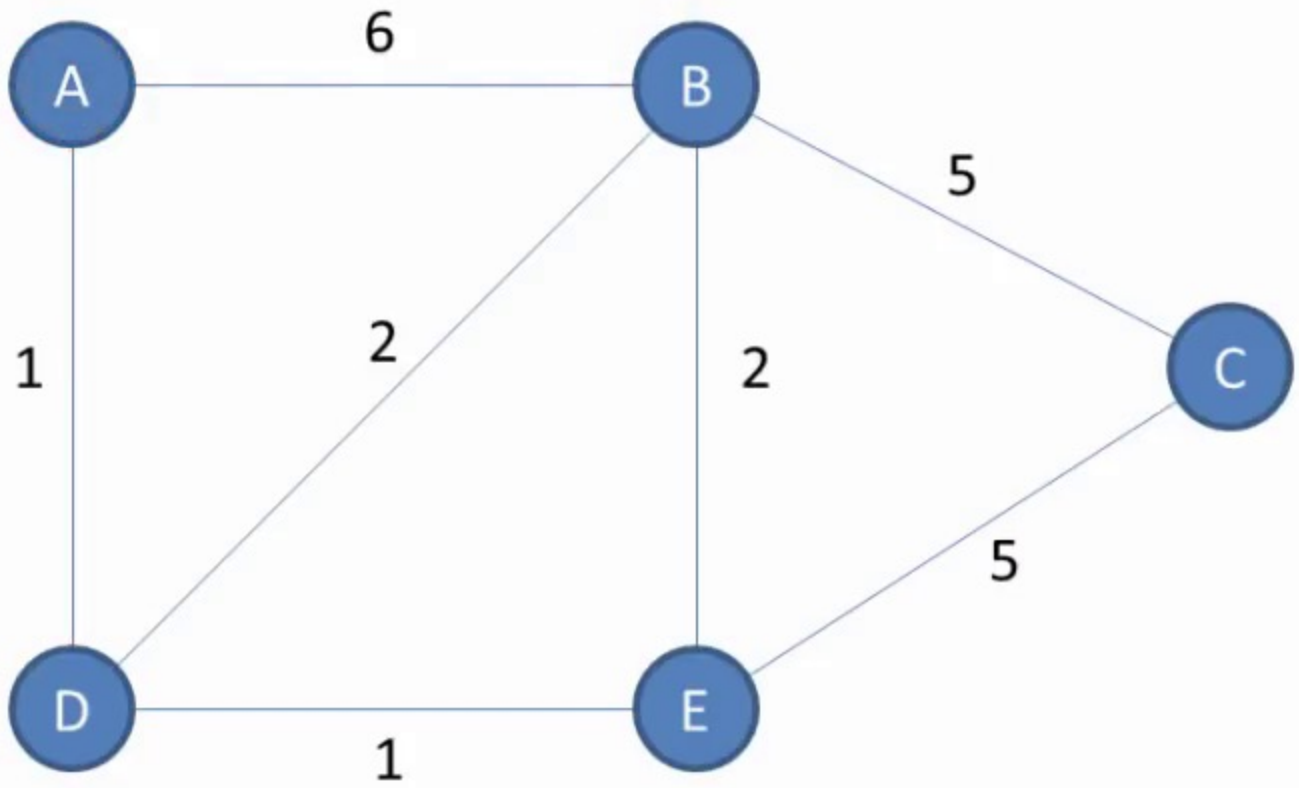


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	
C	∞	
D	1	
E	∞	

Visited = [ ]

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

Add the current vertex to the list of visited vertices



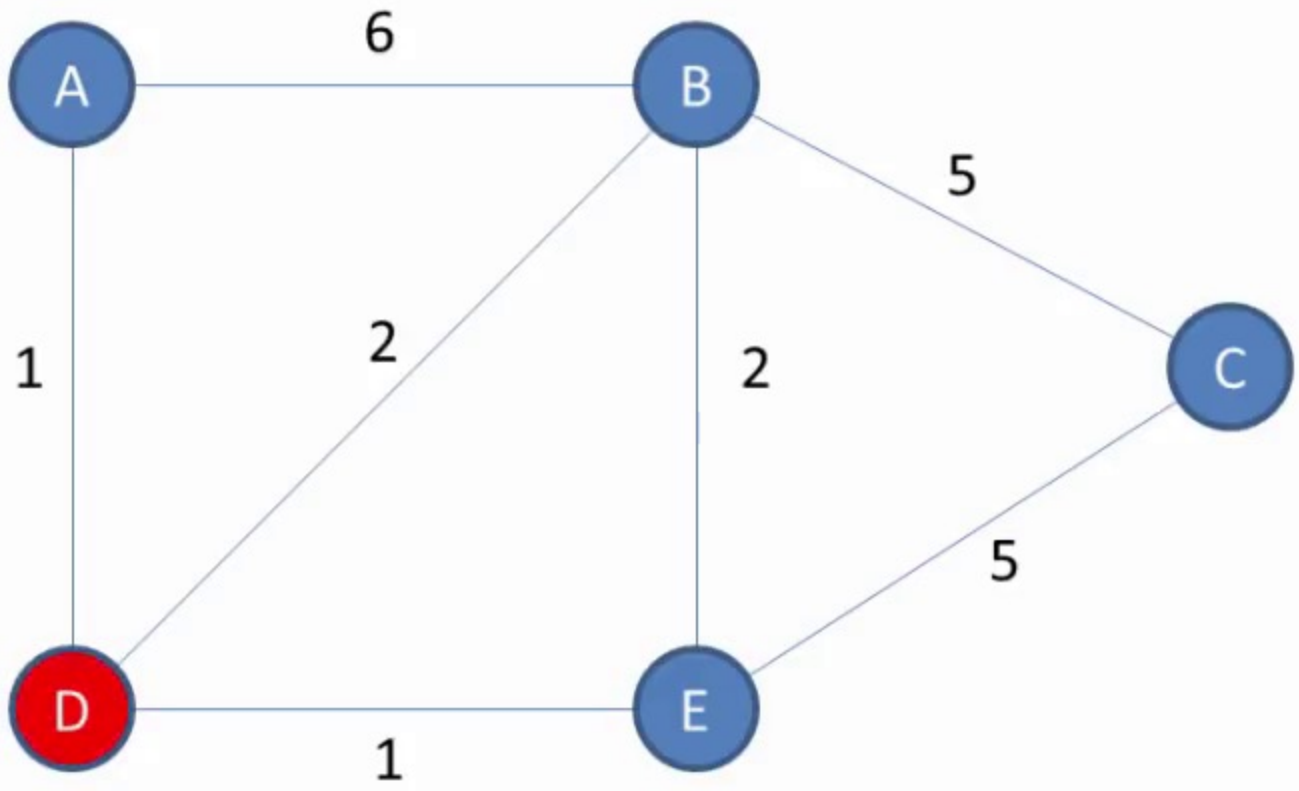
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex D



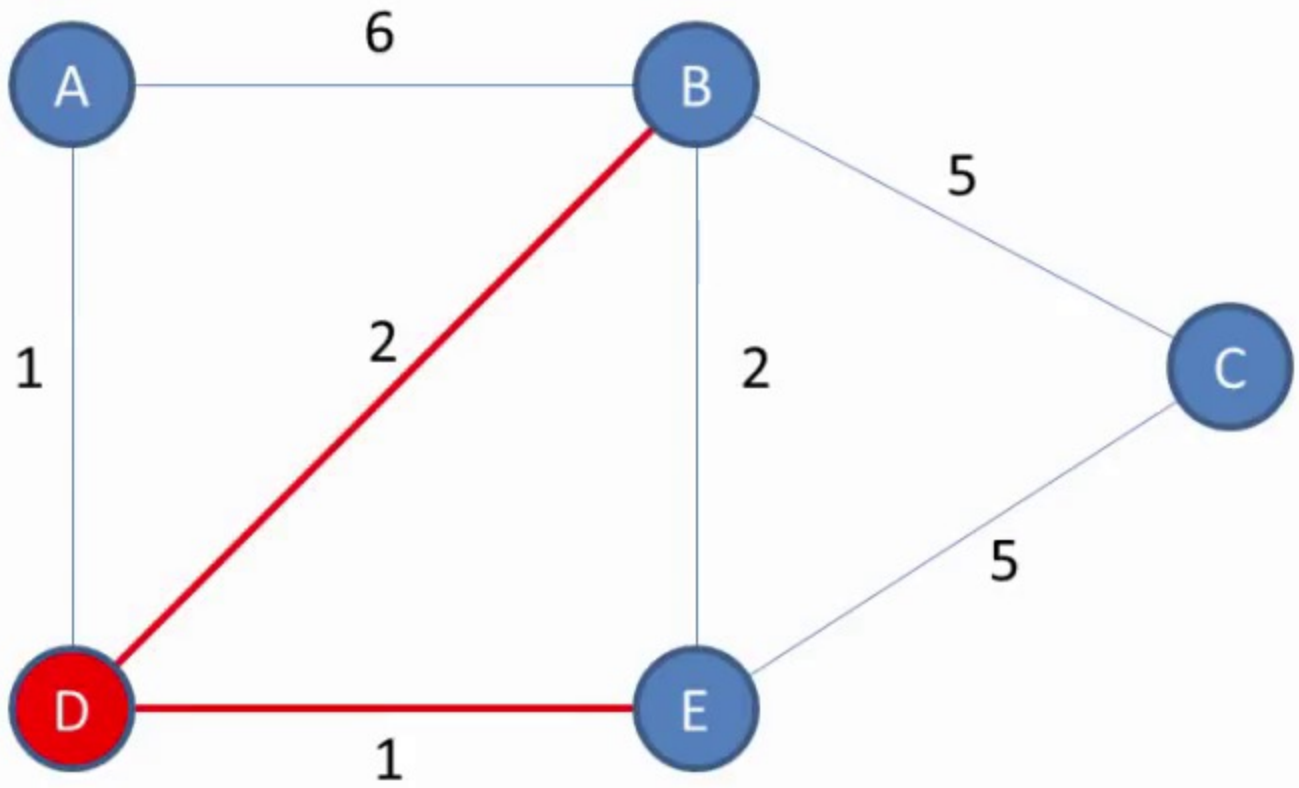
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting D and its unvisited neighbours are B and E

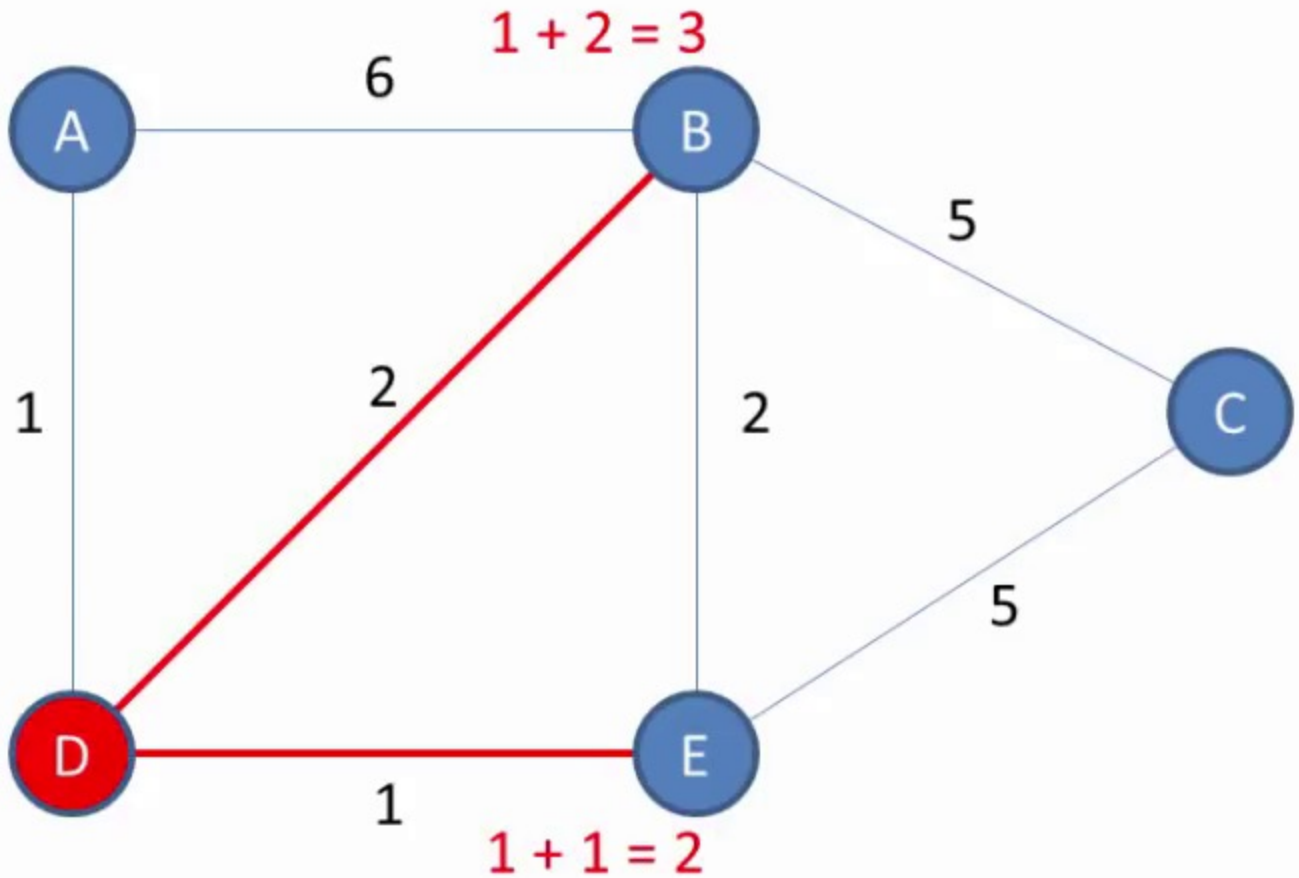


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

For the current vertex, calculate the distance of each neighbour from the start vertex

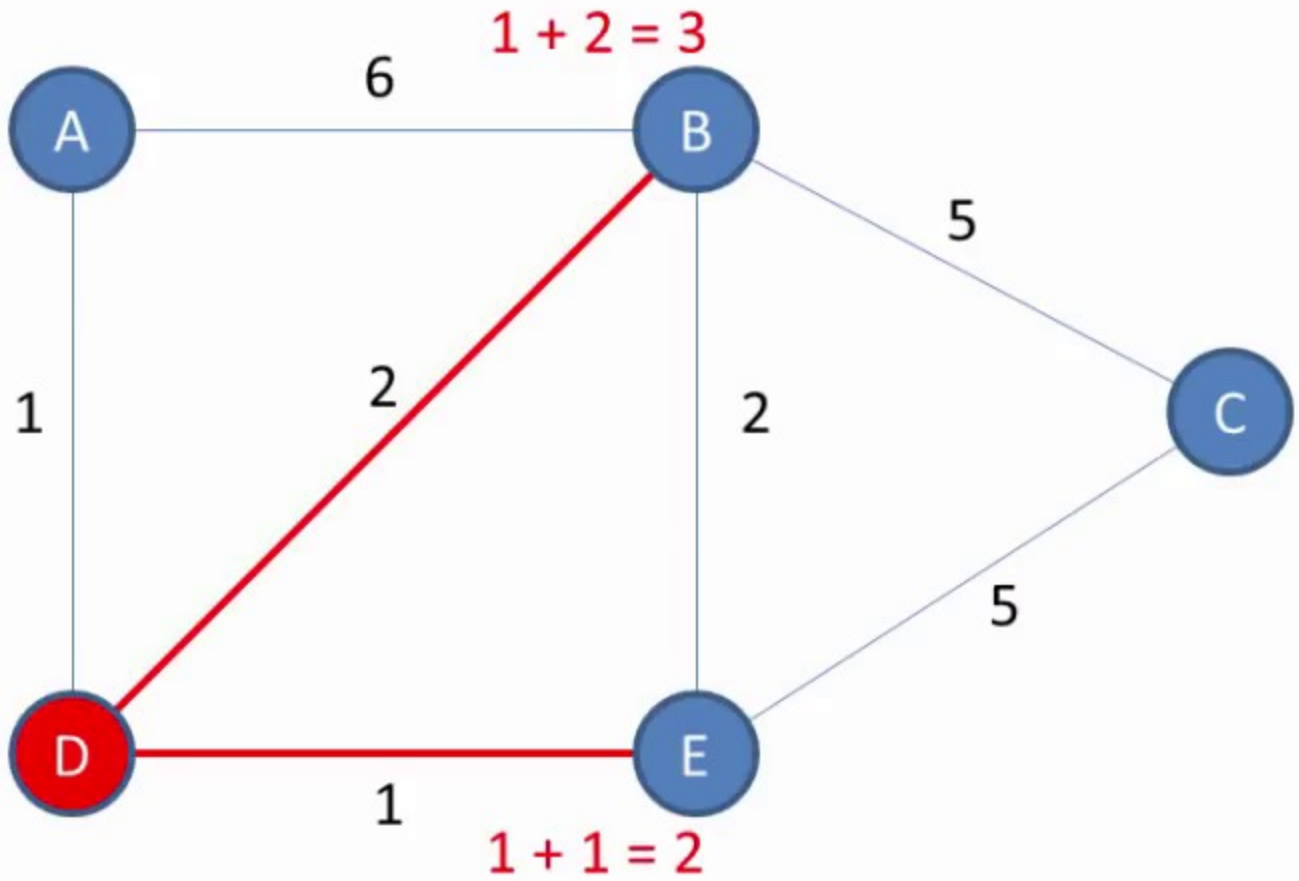


Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance



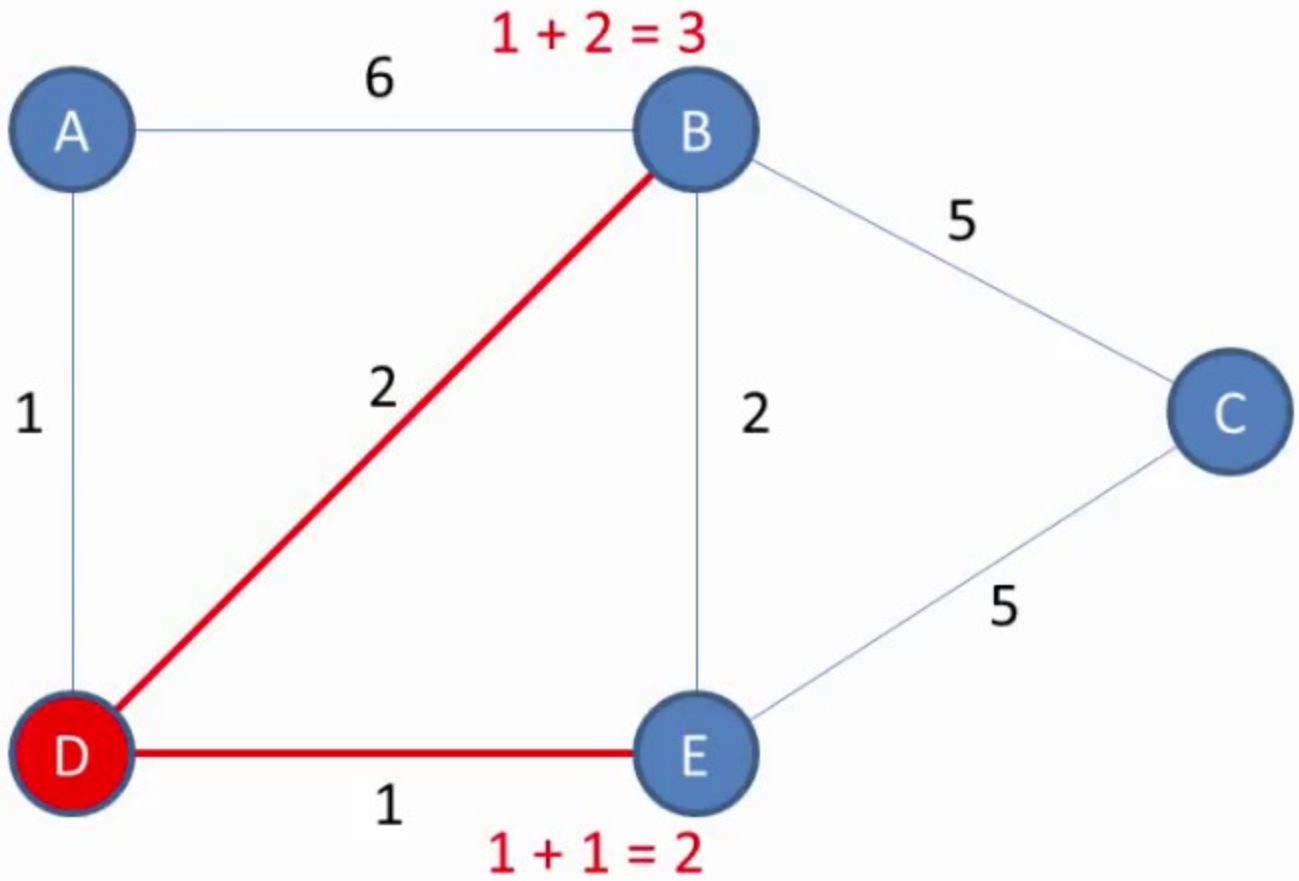
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Visited = [A]

Unvisited = [B, C, D, E]

Update the previous vertex for each of the updated distances

In this case we visited B and E via D

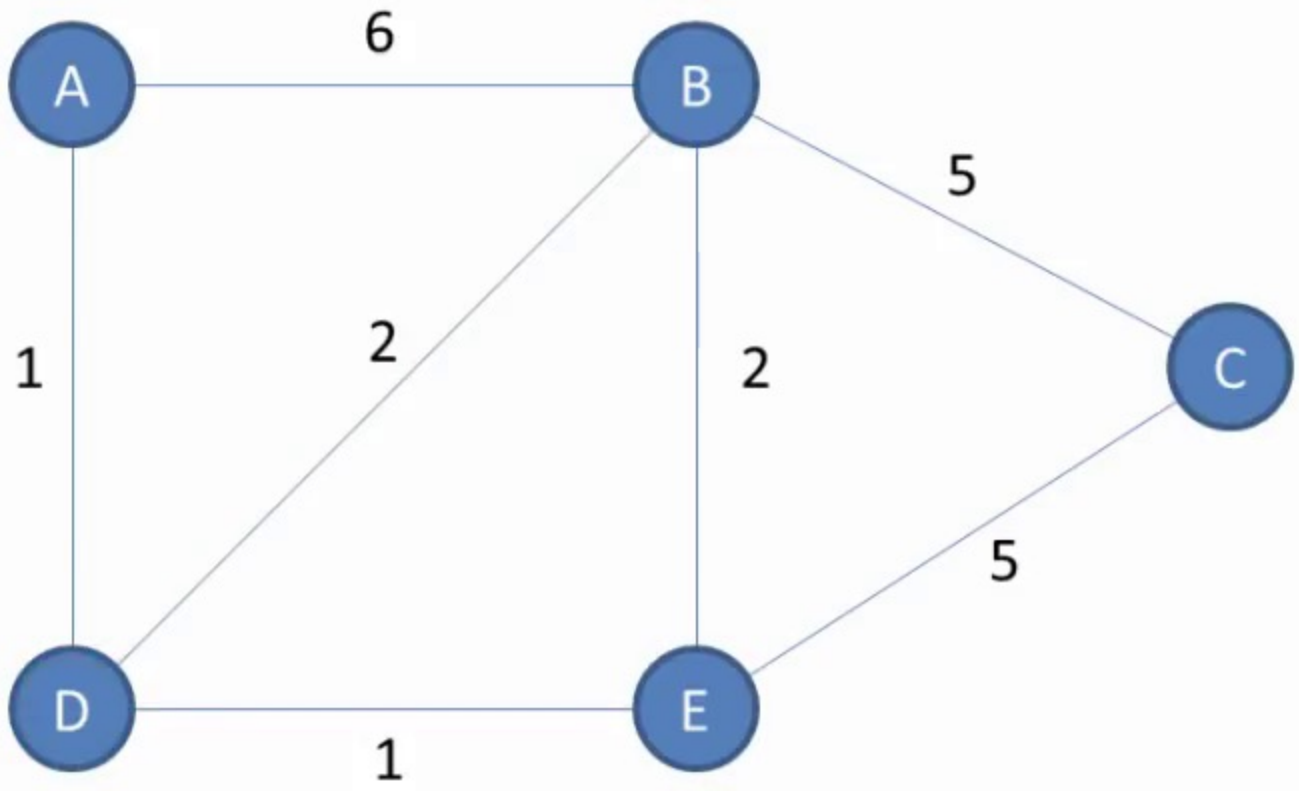


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	A
C	∞	
D	1	A
E	2	

Visited = [A]

Unvisited = [B, C, D, E]

Add the current vertex to the list of visited vertices



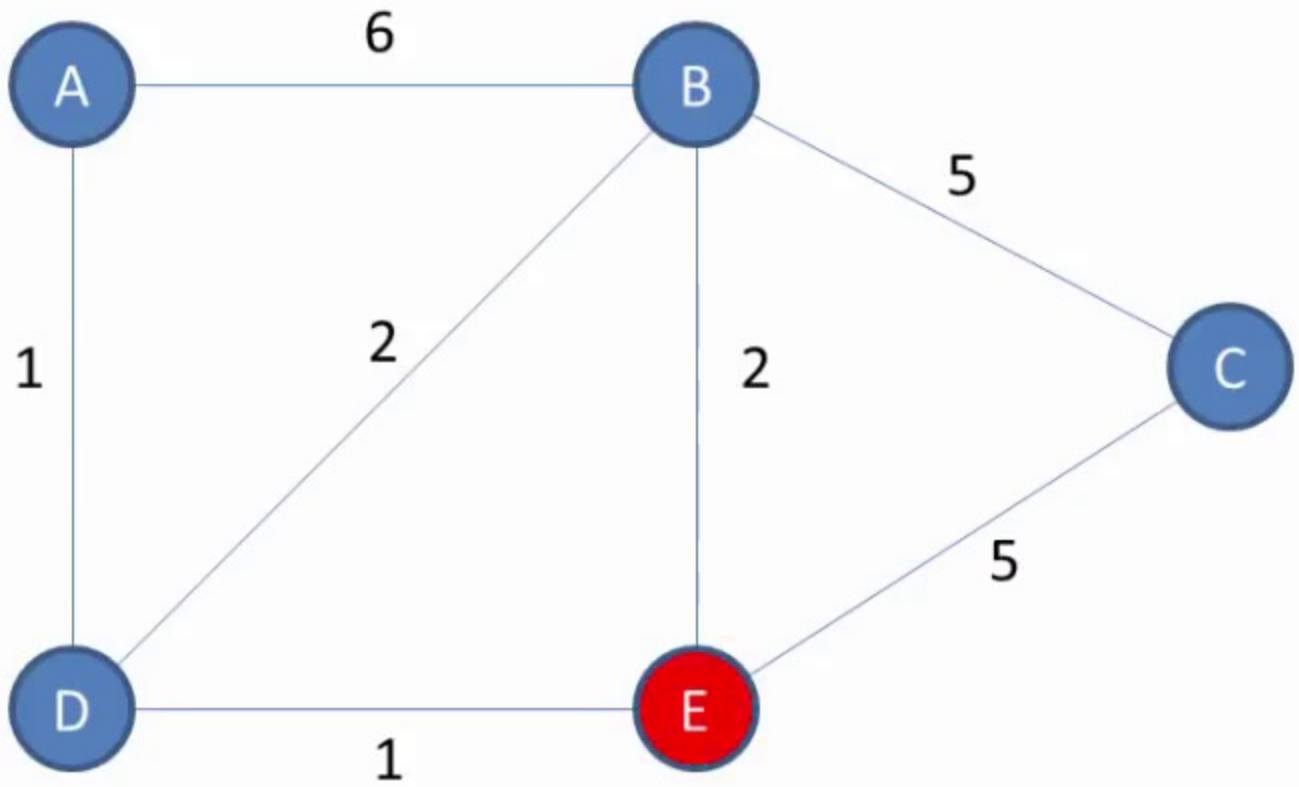
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex E



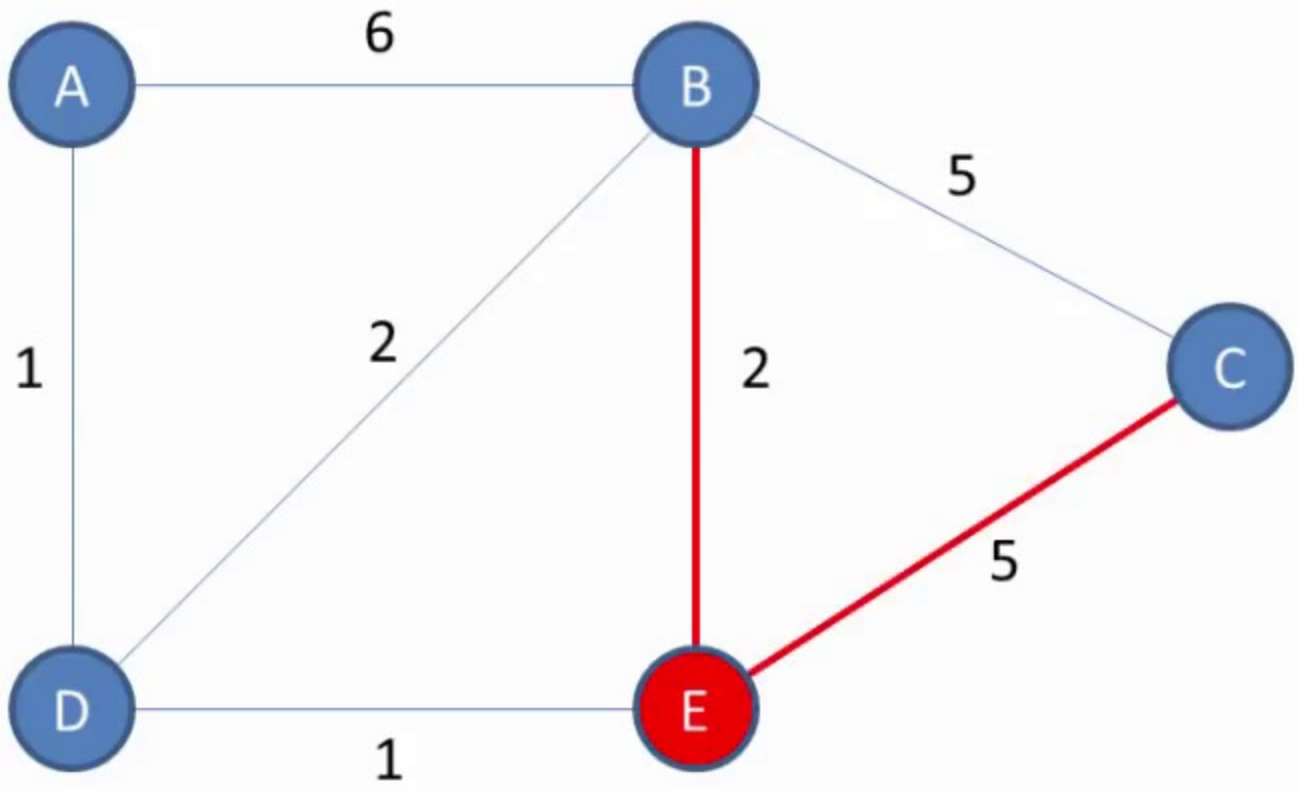
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

For the current vertex, examine its unvisited neighbours

We are currently visiting E and its unvisited neighbours are B and C

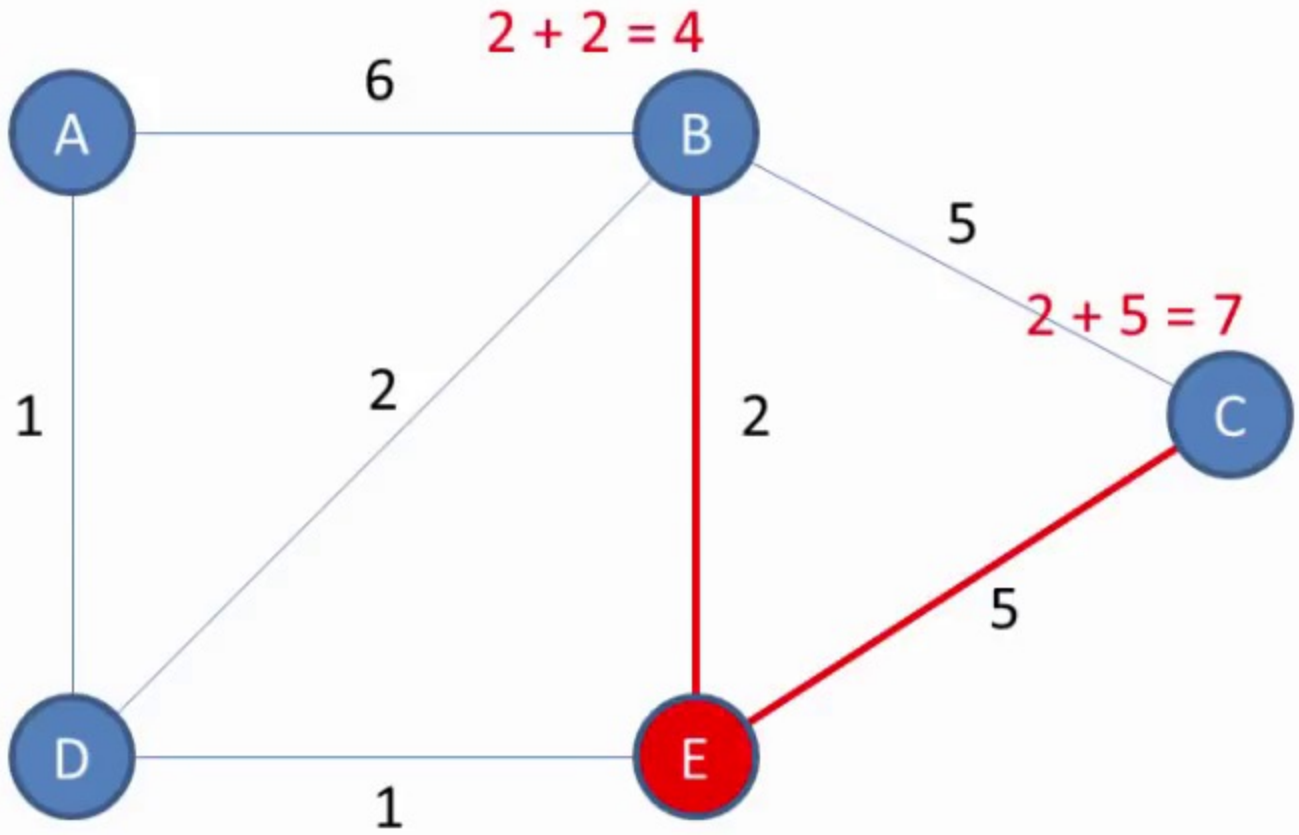


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

For the current vertex, calculate the distance of each neighbour from the start vertex



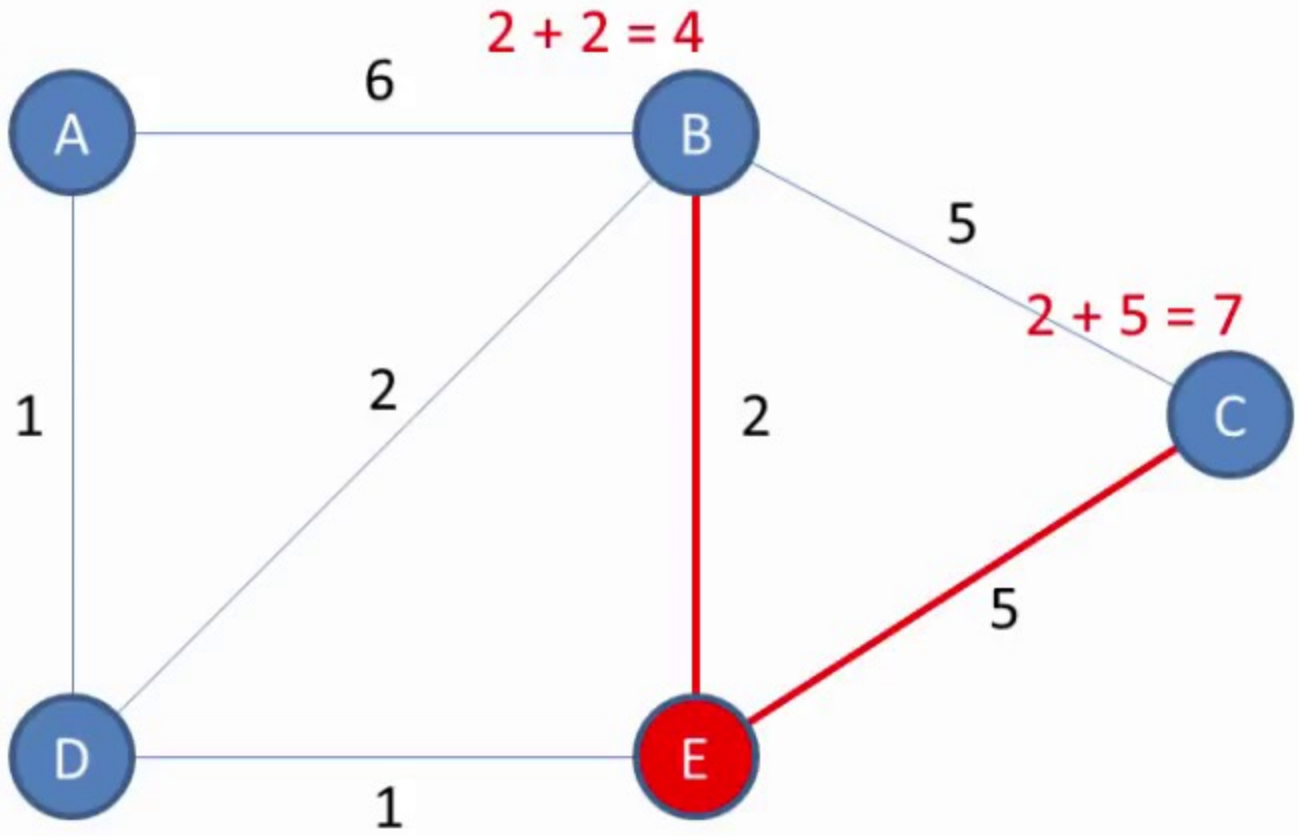
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to B



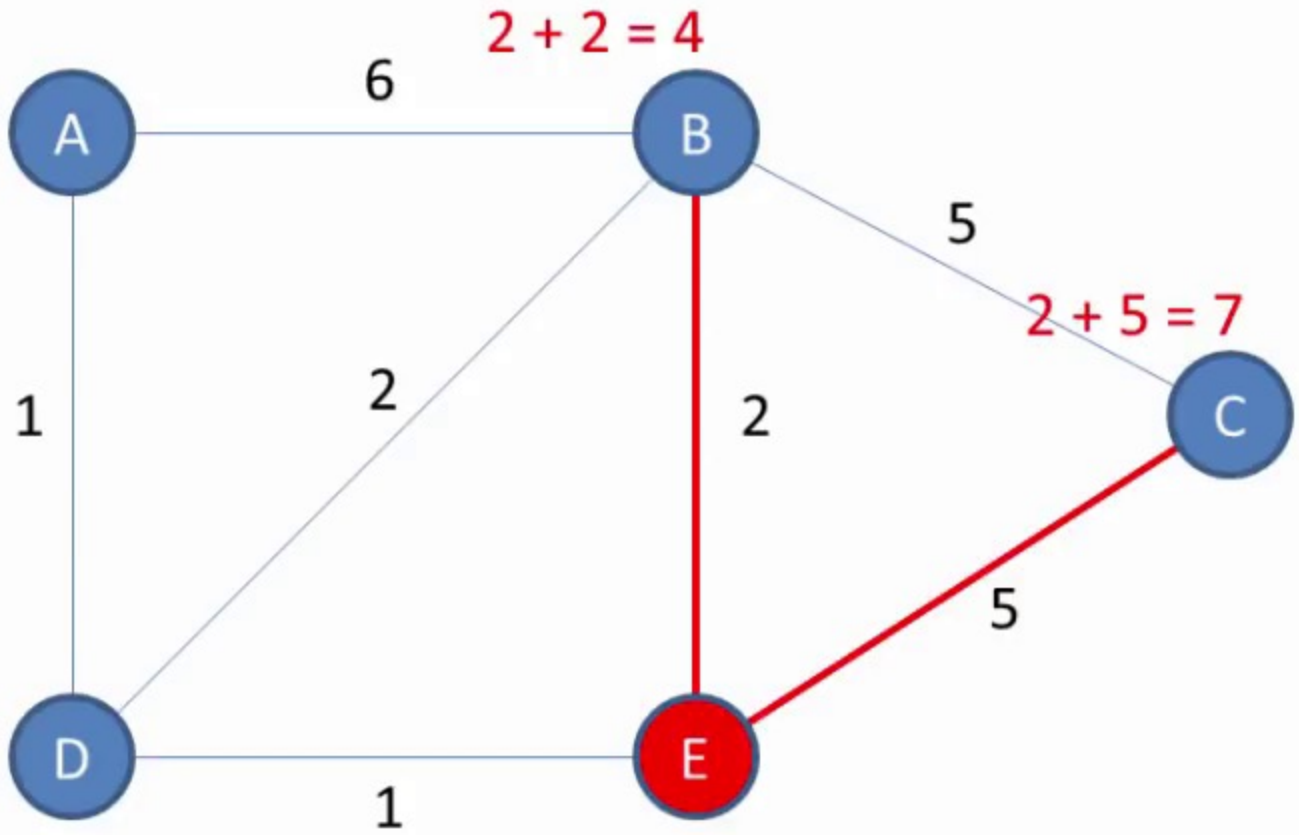
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Update the previous vertex for each of the updated distances

In this case we visited C via E

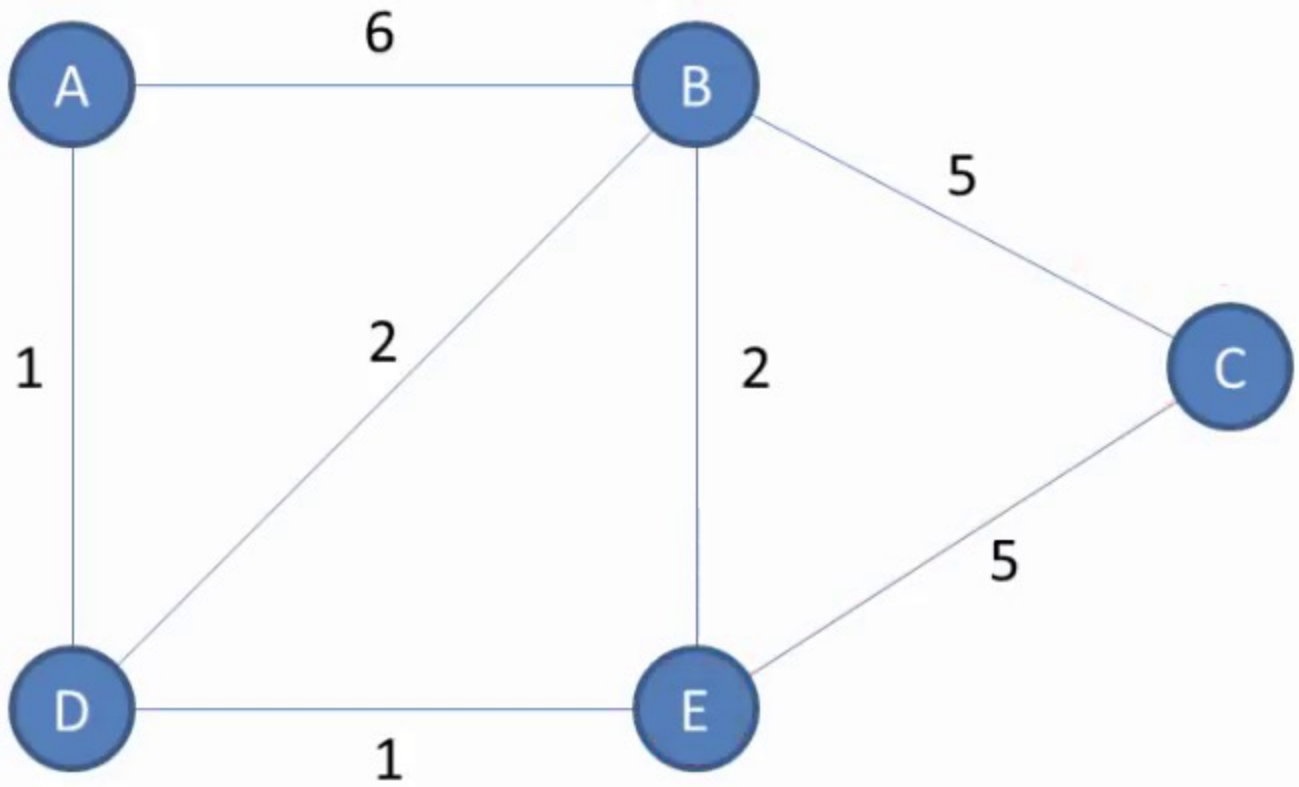


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	
D	1	A
E	2	D

Visited = [A, D]

Unvisited = [B, C, E]

Add the current vertex to the list of visited vertices



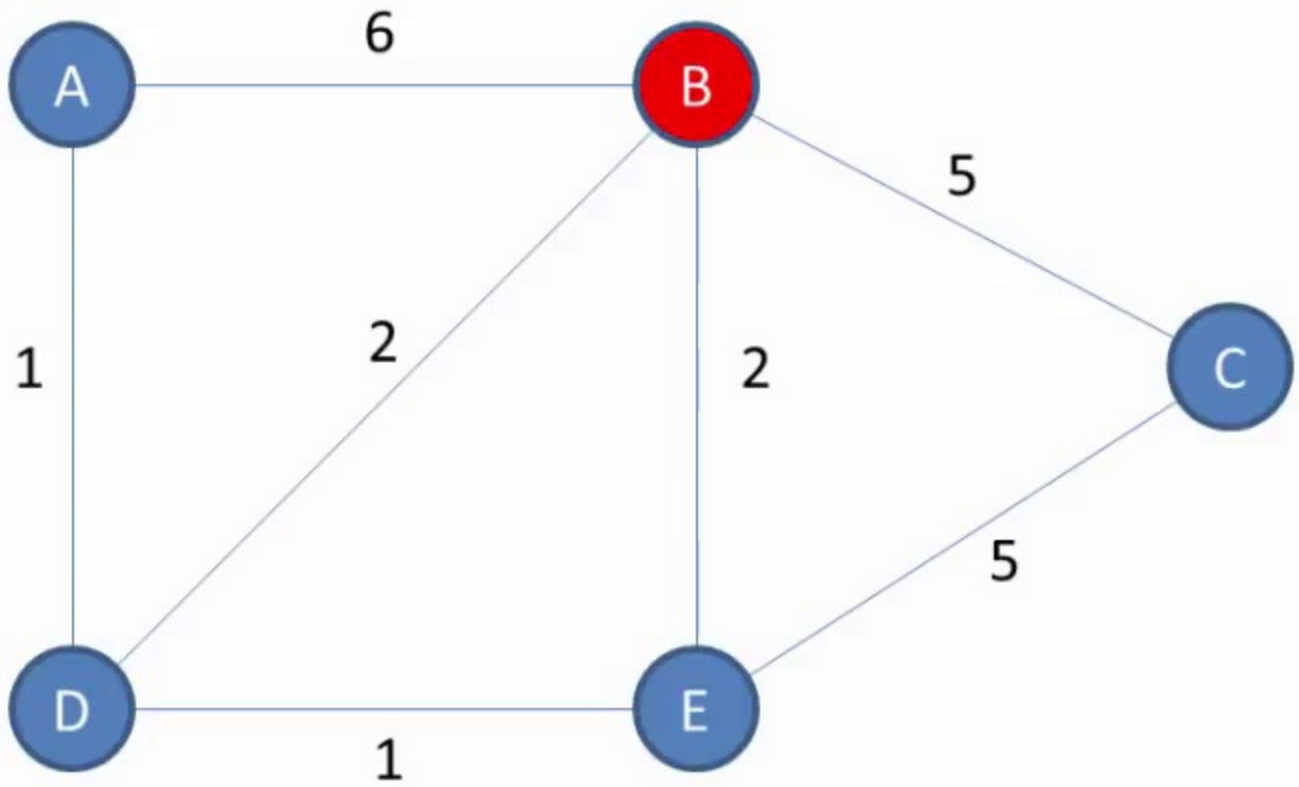
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex B



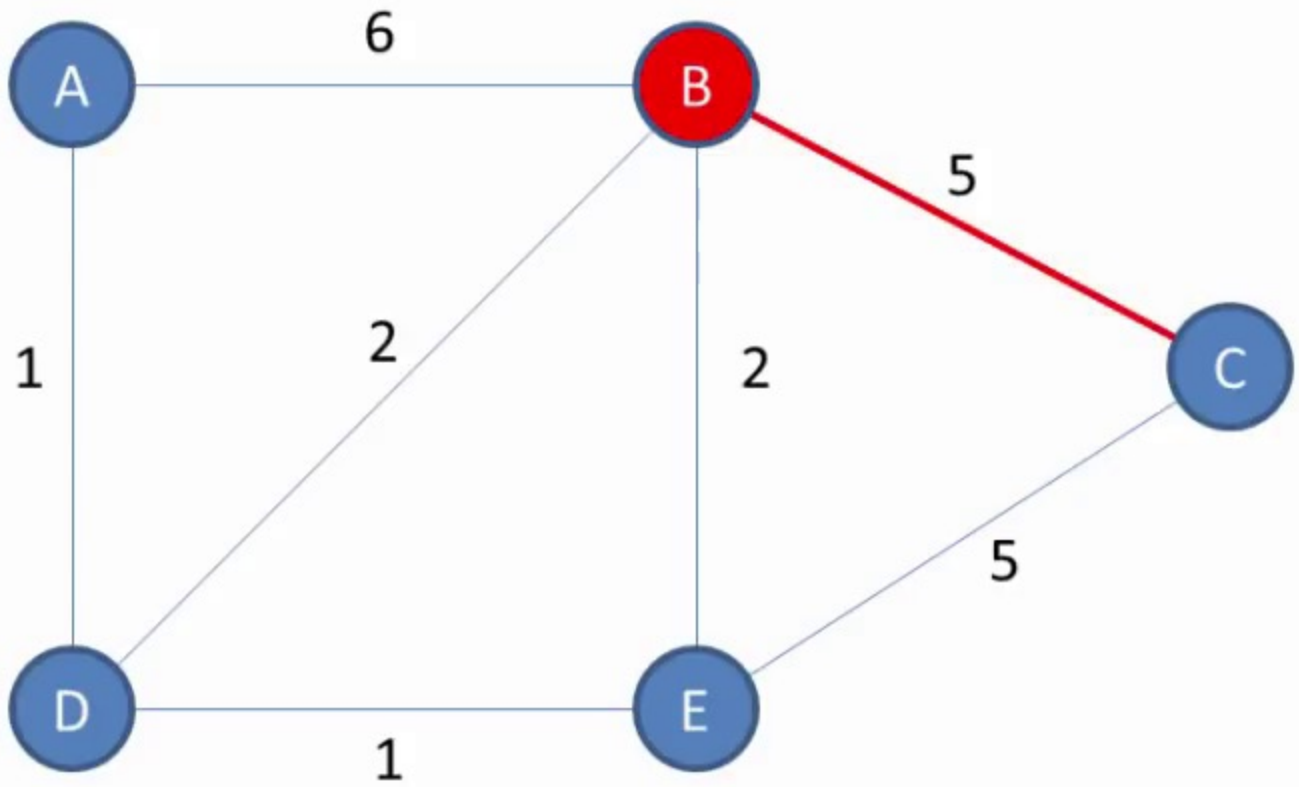
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

For the current vertex, examine its unvisited neighbours

We are currently visiting B and its only unvisited neighbour is C

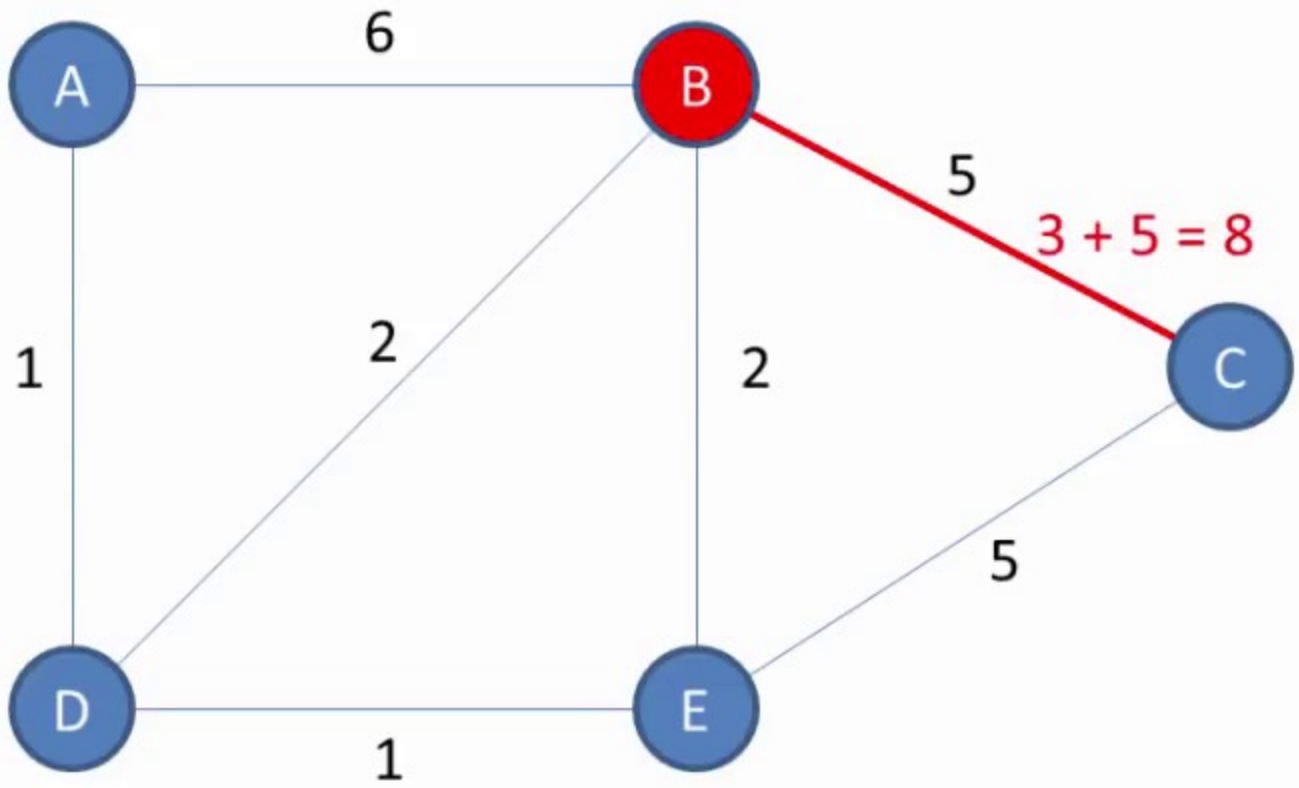


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

For the current vertex, calculate the distance of each neighbour from the start vertex



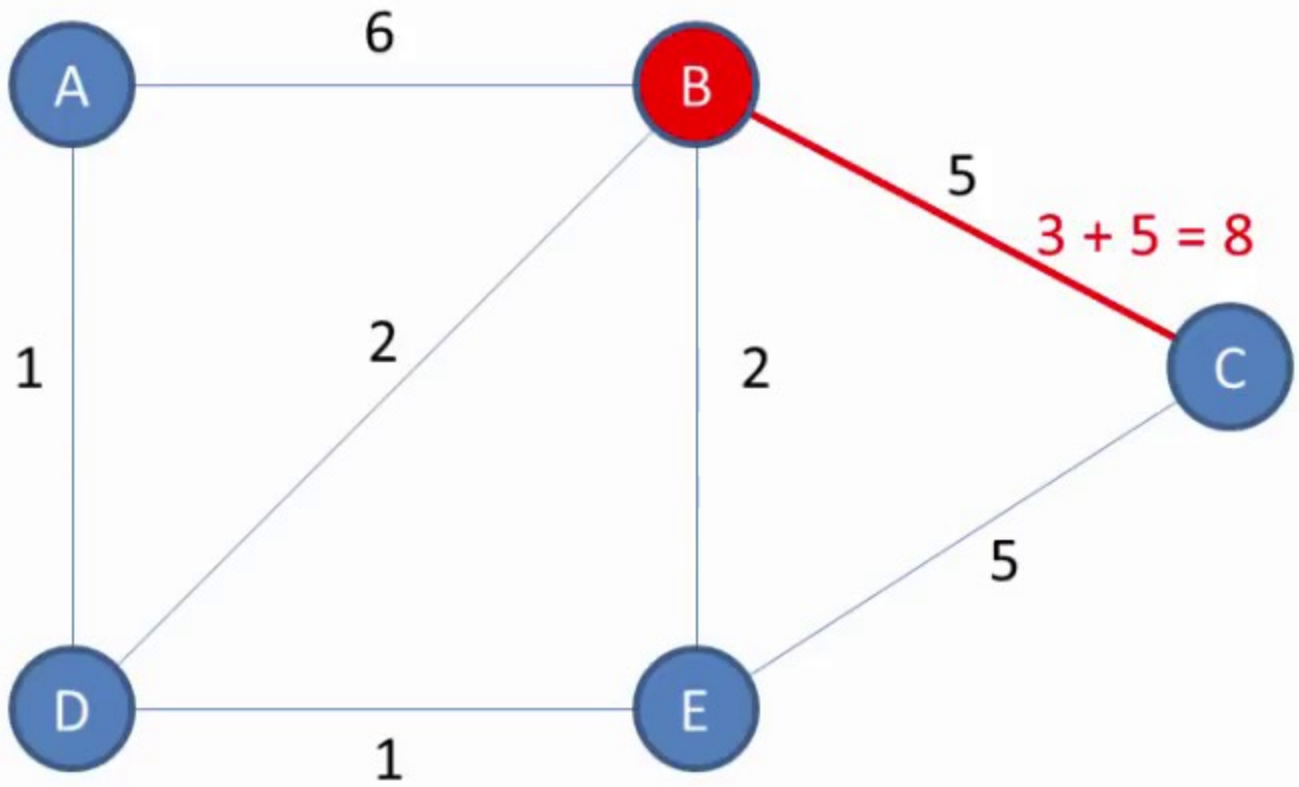
Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

If the calculated distance of a vertex is less than the known distance, update the shortest distance

We do not need to update the distance to C

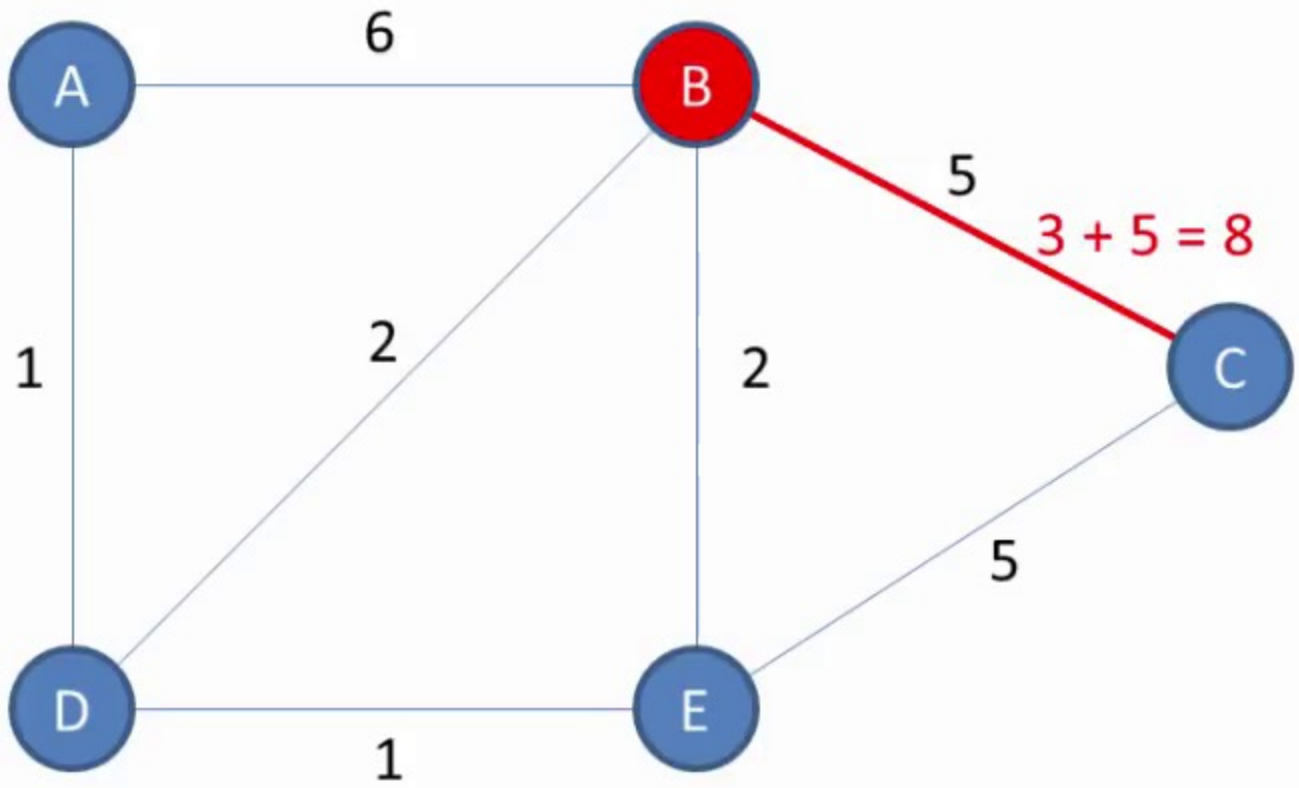


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E] Unvisited = [B, C]

Update the previous vertex for each of the updated distances

No distances were updated, so we don't need to do this either

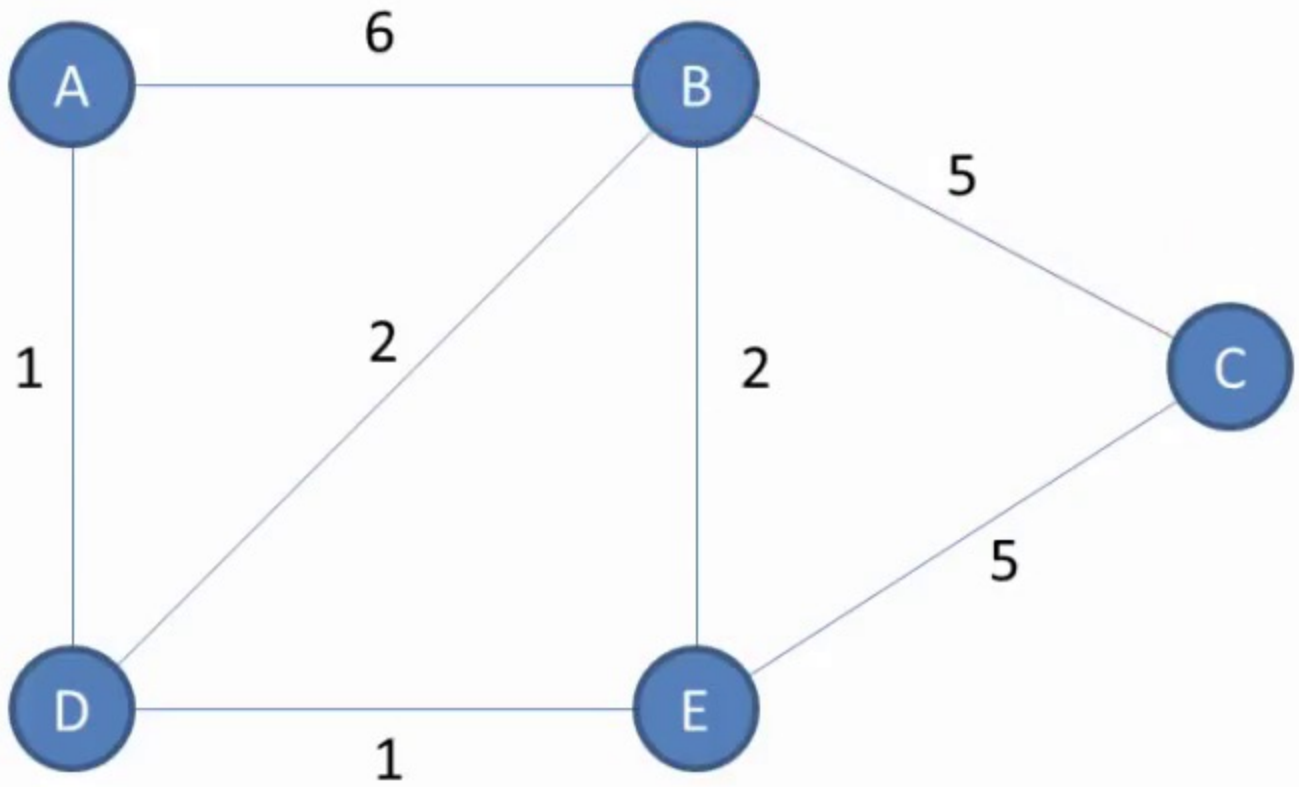


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E]

Unvisited = [B, C]

Add the current vertex to the list of visited vertices

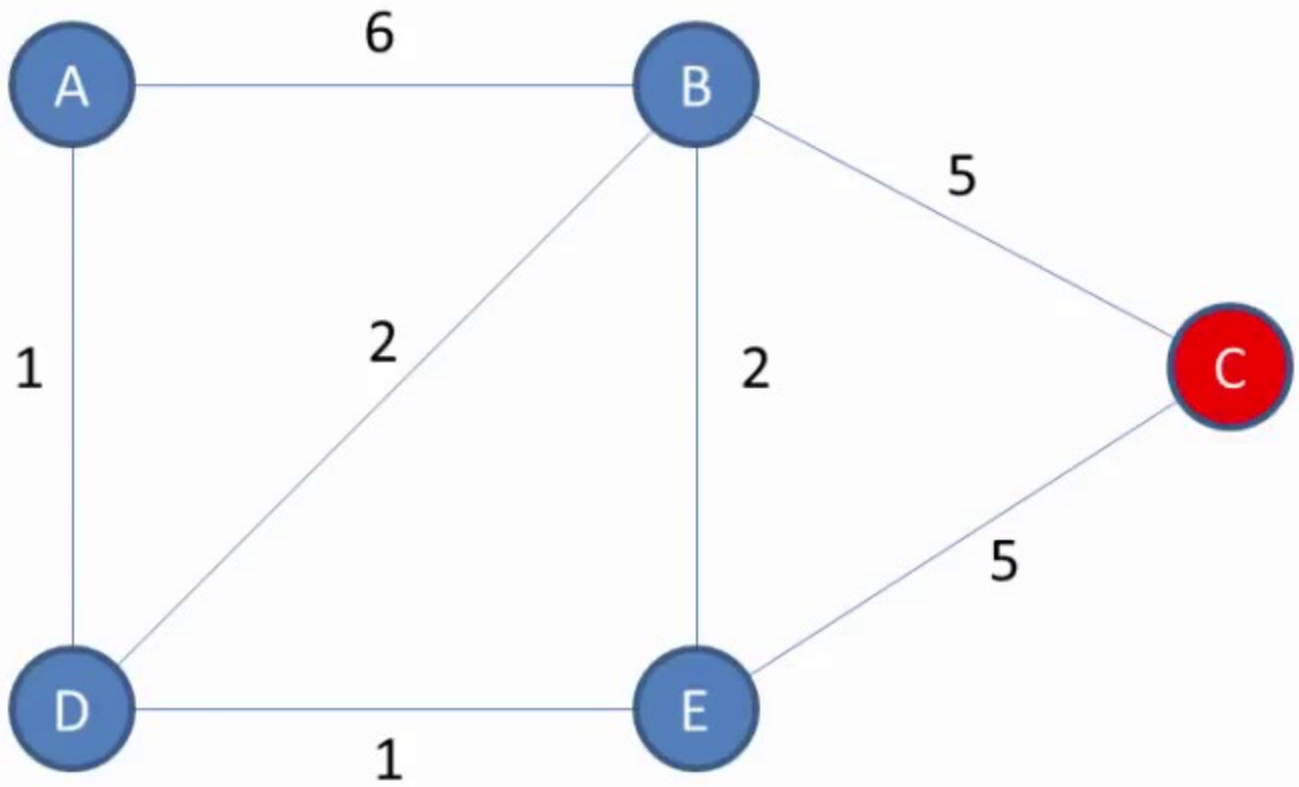


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

Visit the unvisited vertex with the smallest known distance from the start vertex

This time around, it is vertex C

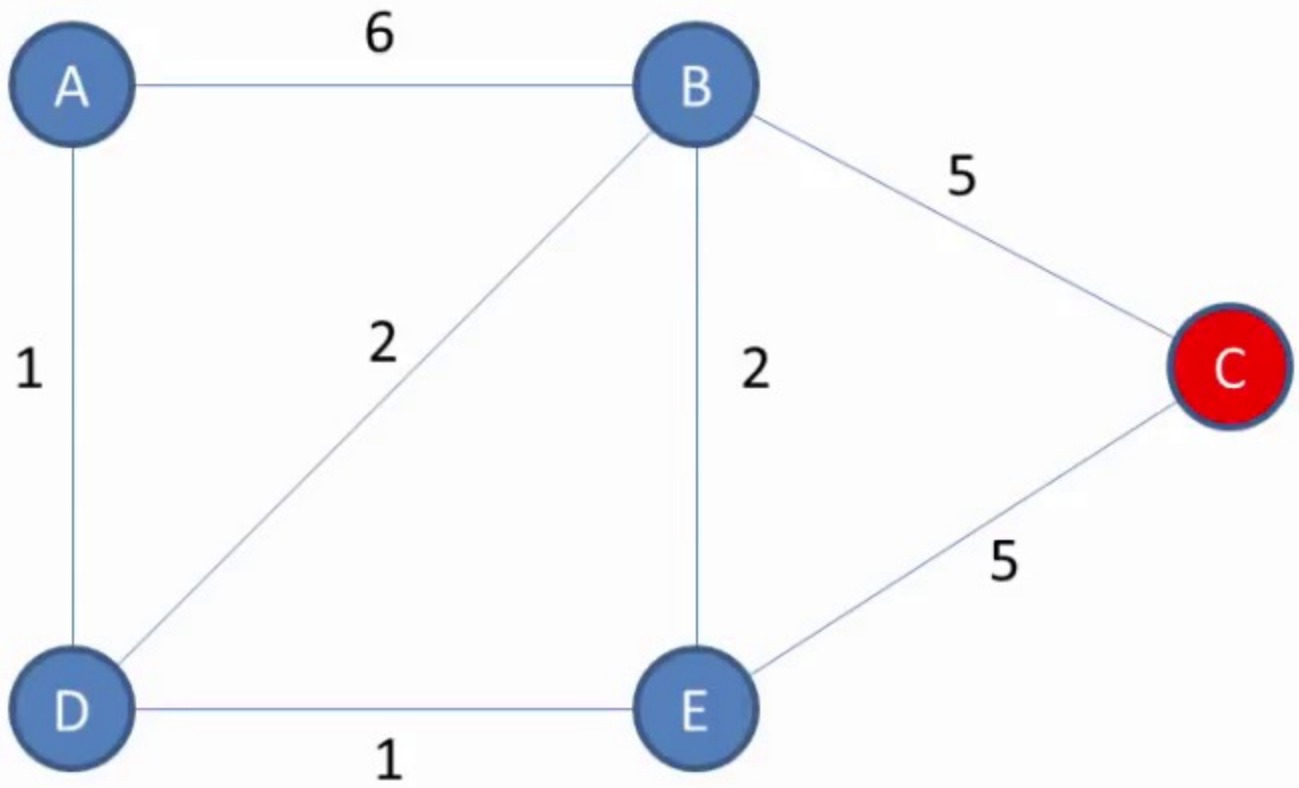


Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

For the current vertex, examine its unvisited neighbours

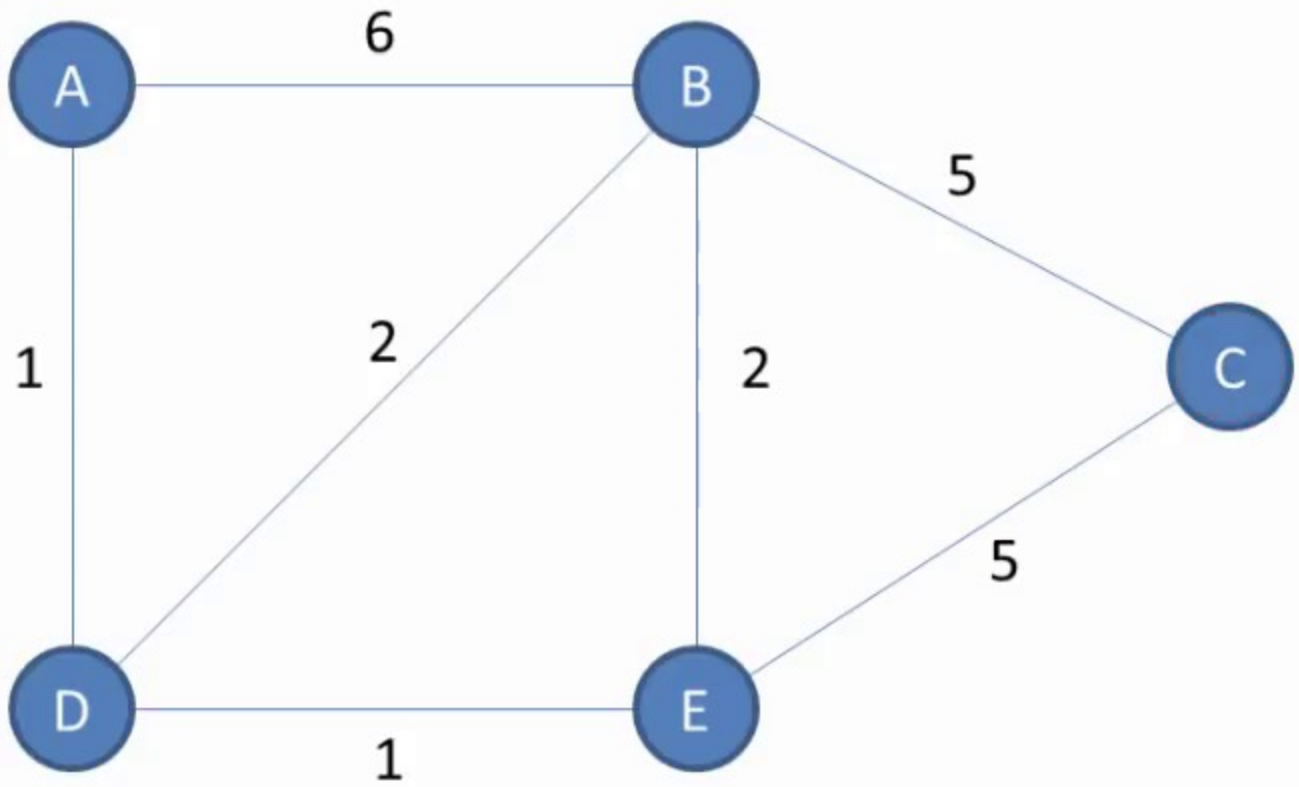
We are currently visiting C and it has no unvisited neighbours



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B] Unvisited = [C]

Add the current vertex to the list of visited vertices



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Visited = [A, D, E, B, C] Unvisited = []

Algorithm

Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

Repeat
- Visit the unvisited vertex with the smallest known distance from the start vertex
- For the current vertex, examine its unvisited neighbours
- For the current vertex, calculate distance of each neighbour from start vertex
- If the calculated distance of a vertex is less than the known distance, update the shortest distance
- Update the previous vertex for each of the updated distances
- Add the current vertex to the list of visited vertices

Until all vertices visited

Section 4: Q&A

Don't forget online quiz 2!