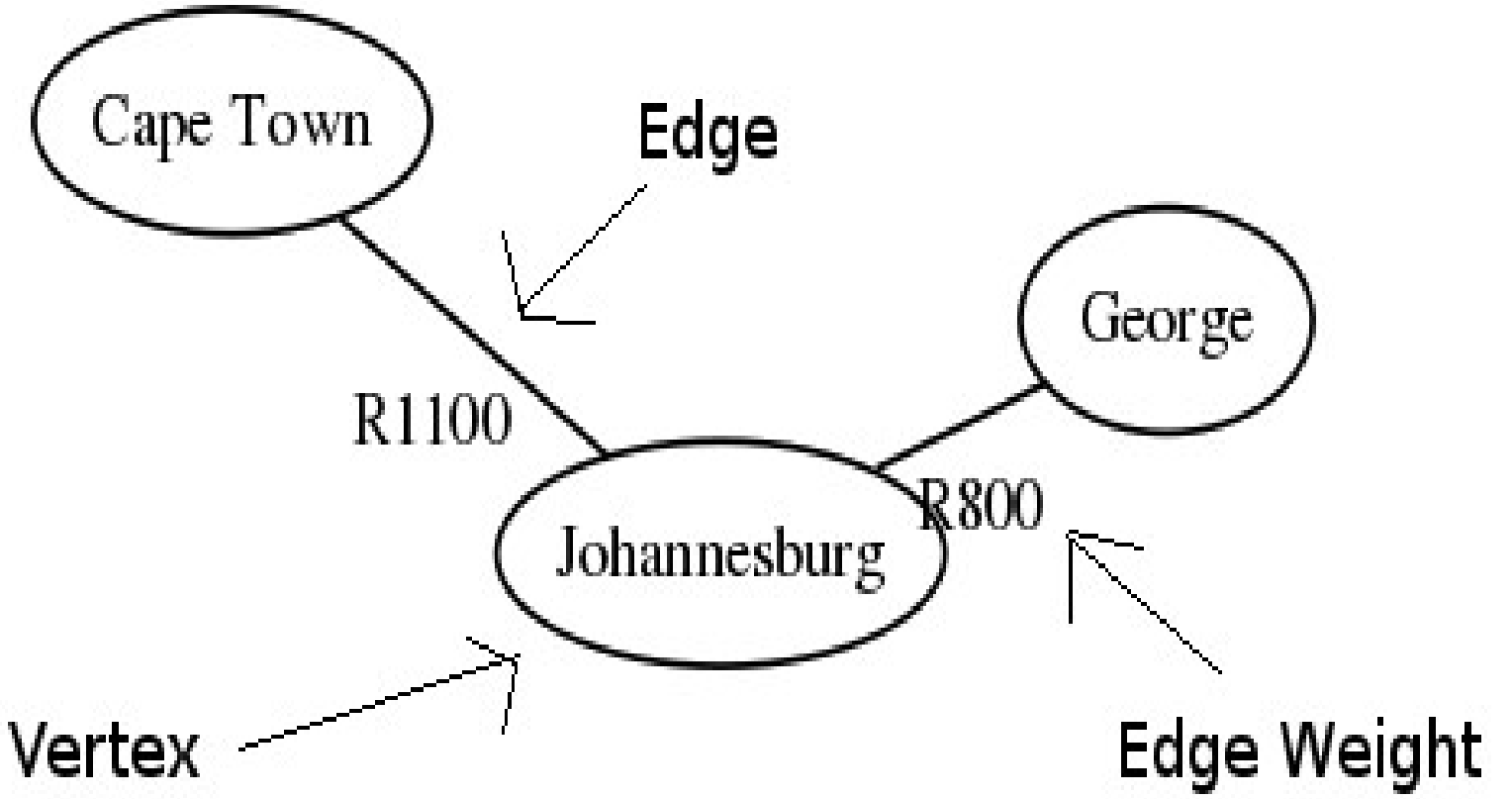
Dijkstra's Algorithm
and
Floyd-Warshall's Algorithm

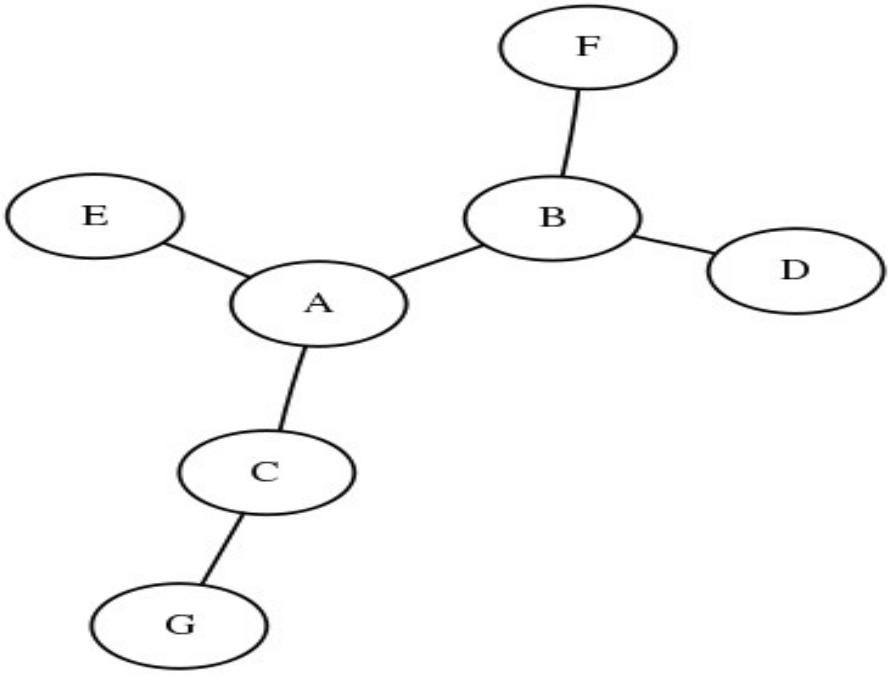
Quick Intro to Graph Theory

Example: Airport Fares



DFS (Depth-First Search)

Visits all the vertices in a graph from a starting vertex. Tries to get as far away from the starting vertex as possible before coming back.

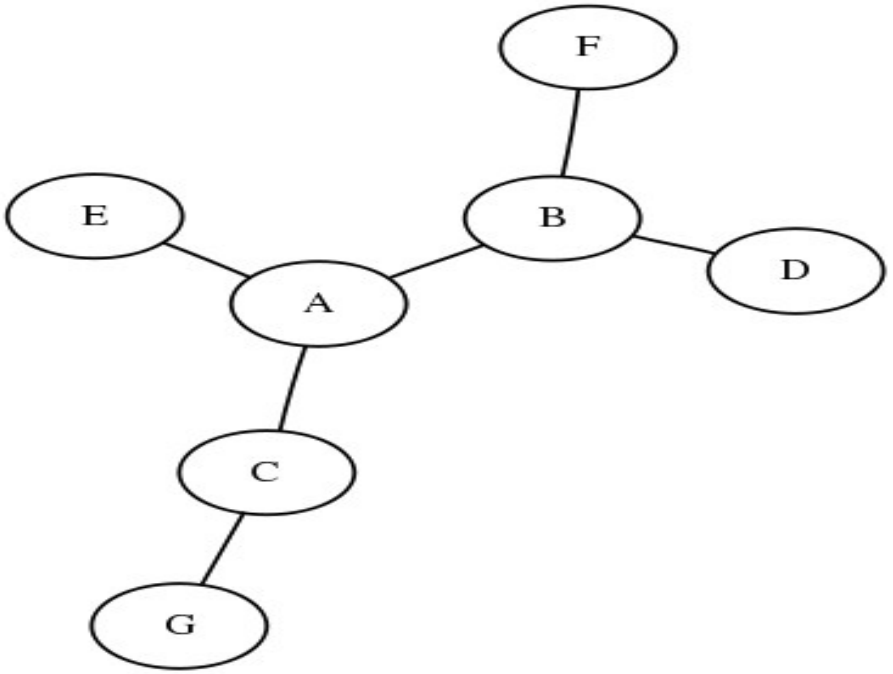


Possible DFS Traversal

A, B, D, F, C, G, E

BFS (Breadth-First Search)

Also Visits all the vertices in a graph from a starting vertex. However, visits them in order of their 'depth' from the starting vertex.

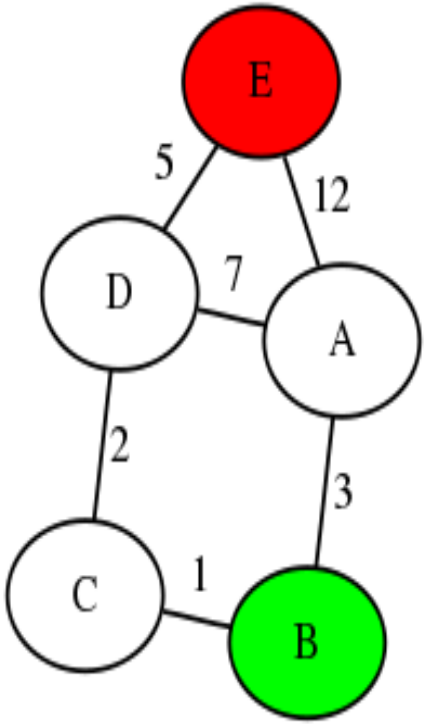


Possible BFS Traversal

A, B, C, E, D, F, G

Problem Statement

How do we find the shortest path between two vertices on a weighted graph?



Example

What is the length of the shortest path from B to E?

Moving on from BFS

We can't use a BFS here because the graph is weighted.

The problem here is that vertices on the graph are no longer visited in the same order of closest to the source first.

Introducing Dijkstra's Algorithm

A small modification to BFS can be made to ensure the correct order is maintained.

We replace the BFS's queue with a Priority Queue. Vertices are added to the Priority Queue by their distance away from the source.

Using a Priority Queue(C++)

```
#include <queue>

class Vertex
{
    public:
        bool operator<(const Vertex& v) const;
};

priority_queue<Vertex> pq; // Vertex must overload < operator.
```

Using a Priority Queue(Java)

```
class Vertex implements Comparable
{
    public int  compareTo(Vertex v)
    {...}
}

PriorityQueue<Vertex> pq; // Vertex must implement Comparable
```

Run-time

The run-time of Djikstras' depends on how the Priority Queue is implemented.

Linked List/Array:	O(|V|^2)
Binomial Heap:	O(|E|log|V|)

Djikstra's Algorithm

We need to keep track of several things with the vertices.

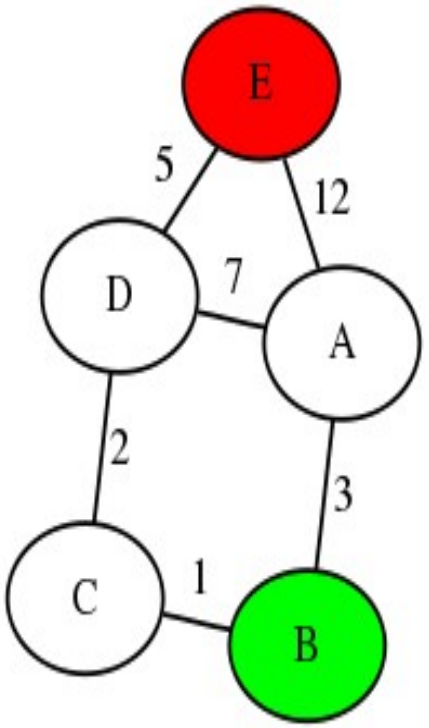
- If the vertex has been visited (initially false).
- Distance from the source (initially infinite).

Pseudocode

```
set source distance to 0
add source to priority queue
while priority queue is not empty
    vertex = top of priority queue
    remove top element
    if vertex is not visited
        mark vertex as visited
        for each neighbour of vertex
            if neighbour is not visited
                if neighbour distance > vertex distance + edge weight from
                vertex to neighbour
                    set neighbour distance to vertex distance + edge
weight from vertex to neighbour
                    add neighbour to priority queue
```

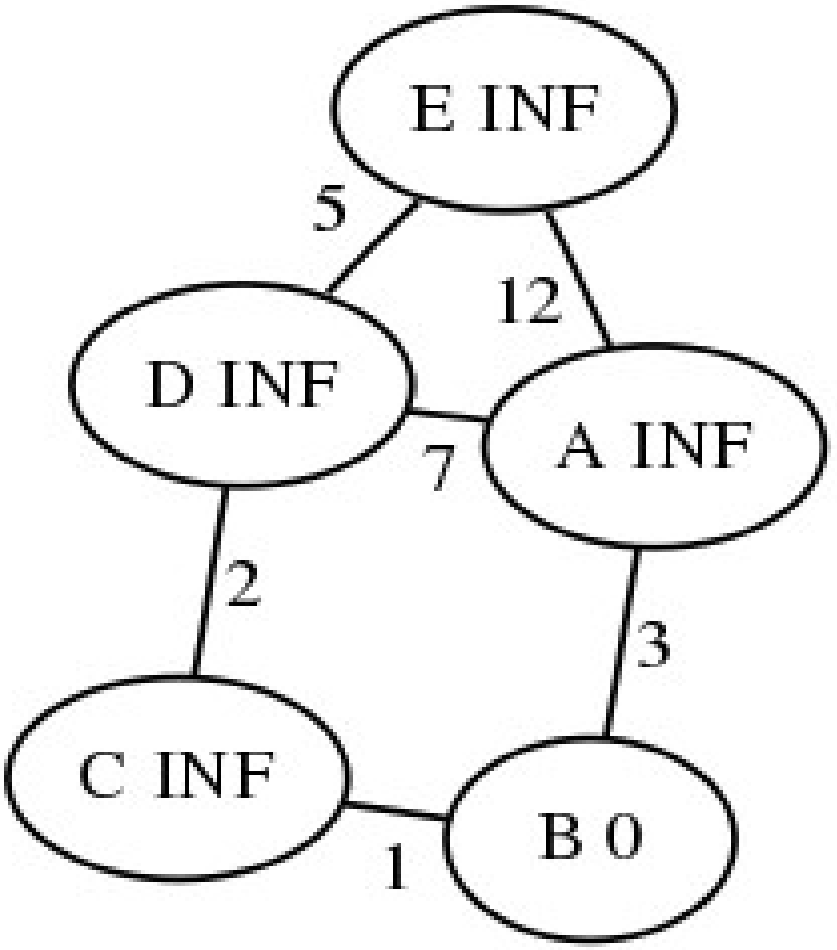
Example

Lets head back to our original example.



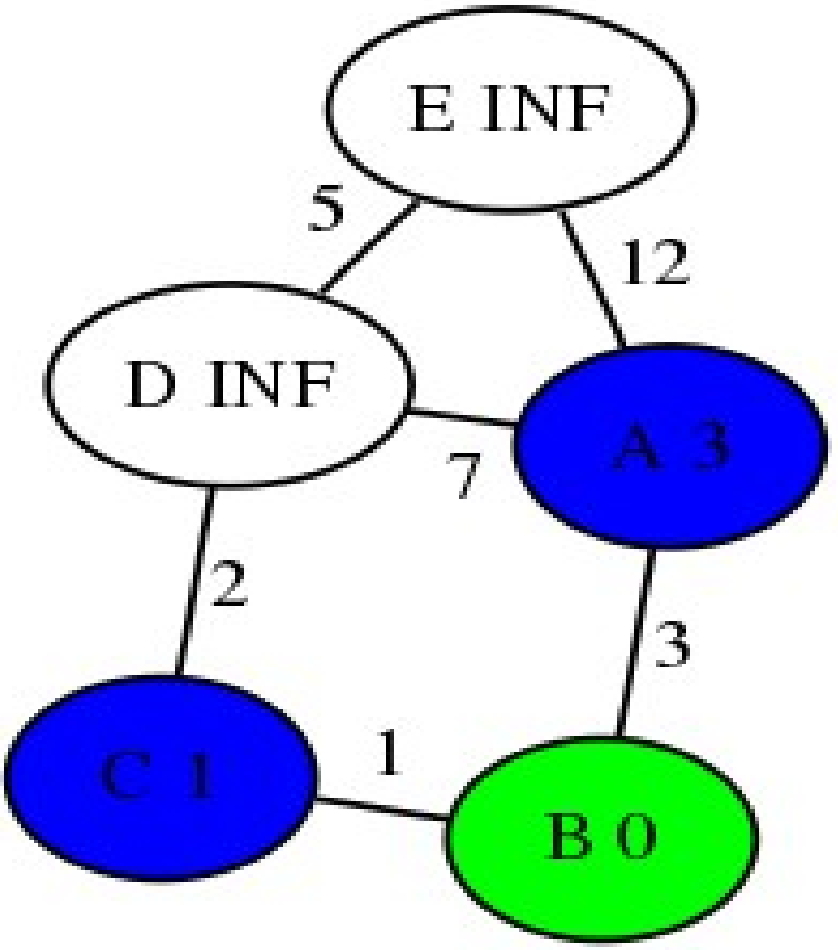
Find the length of the shortest path from B to E.

Example(cont)



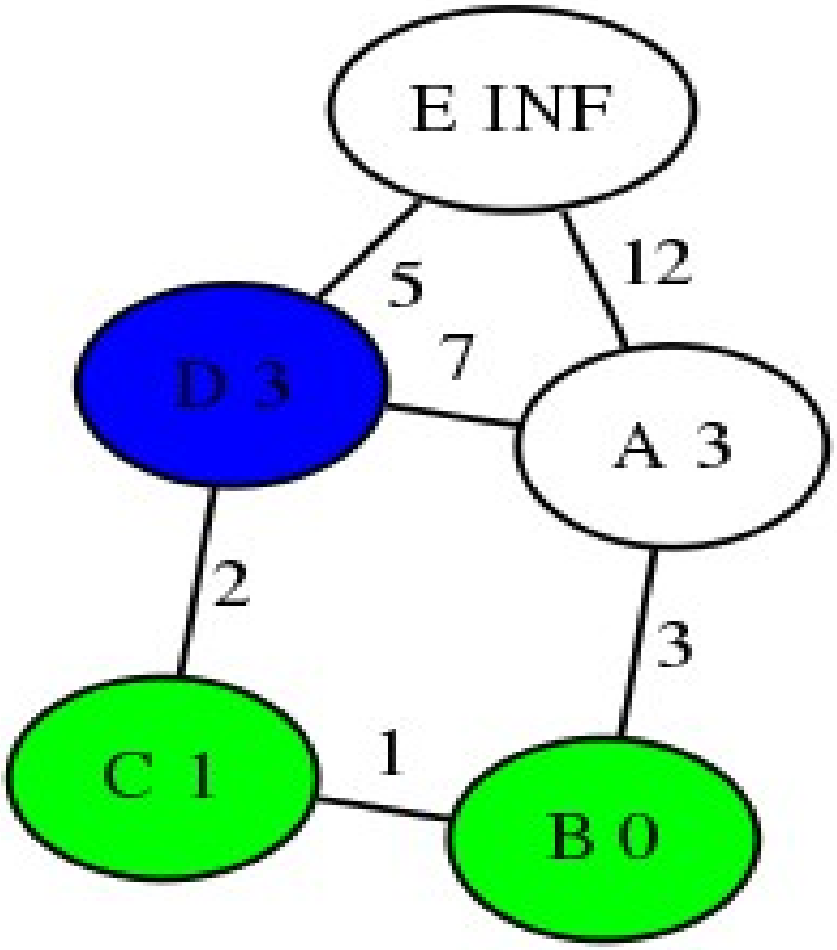
Priority Queue: B(0)

Example(cont)



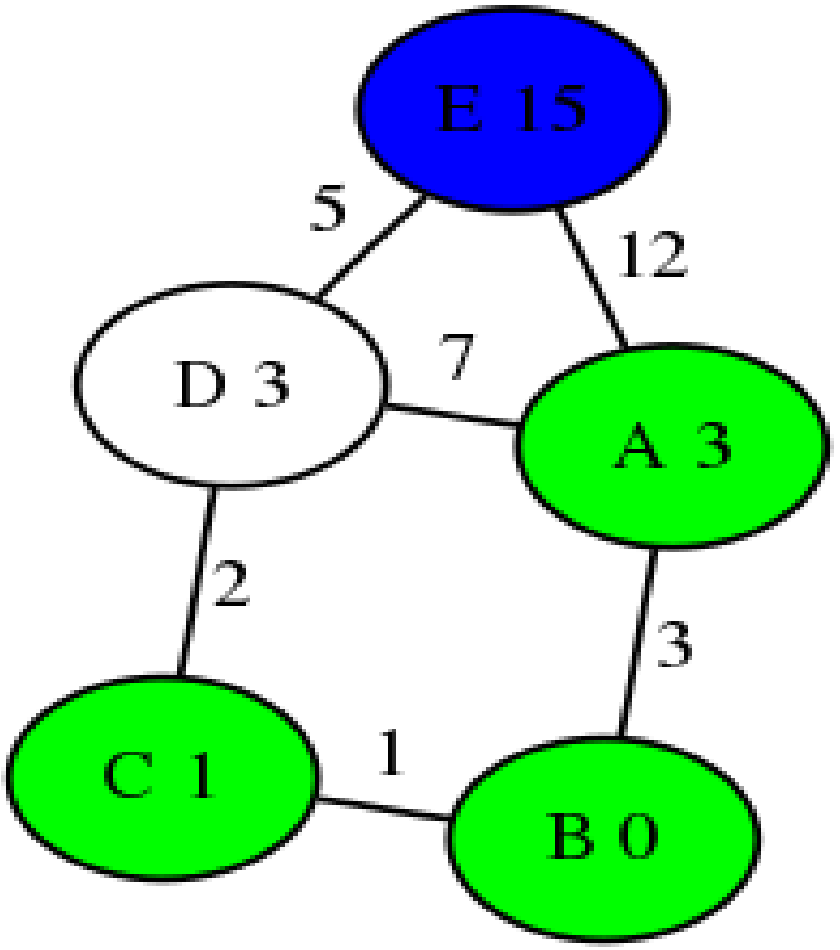
Priority Queue: C(1), A(3)

Example(cont)



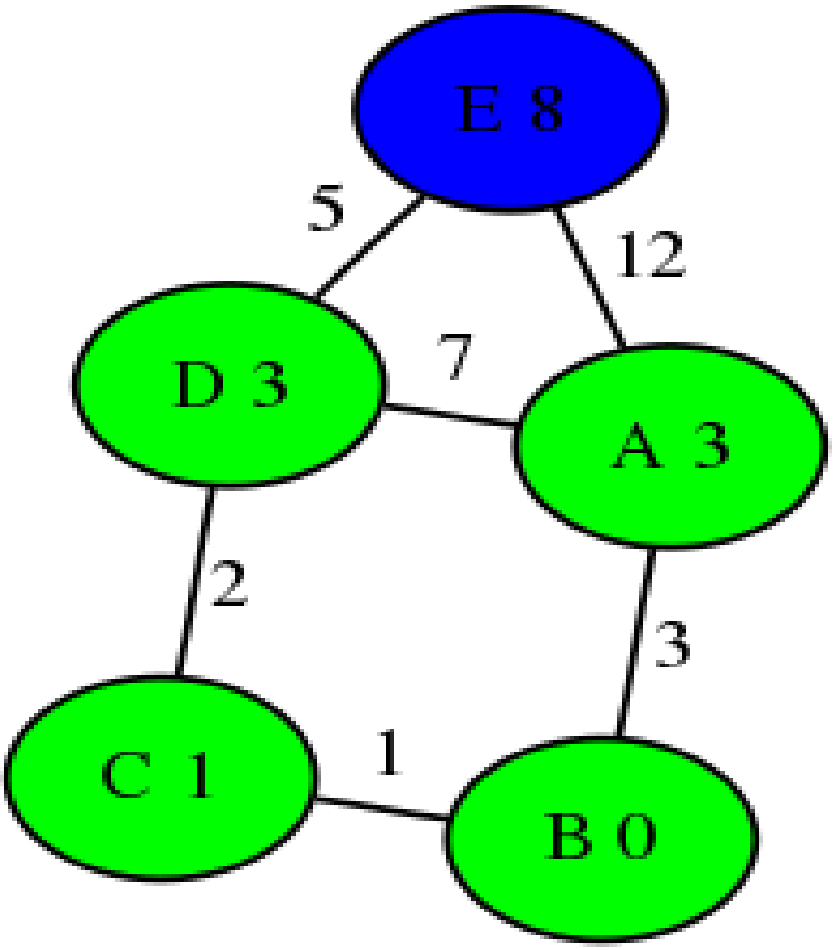
Priority Queue: A(3), D(3)

Example(cont)



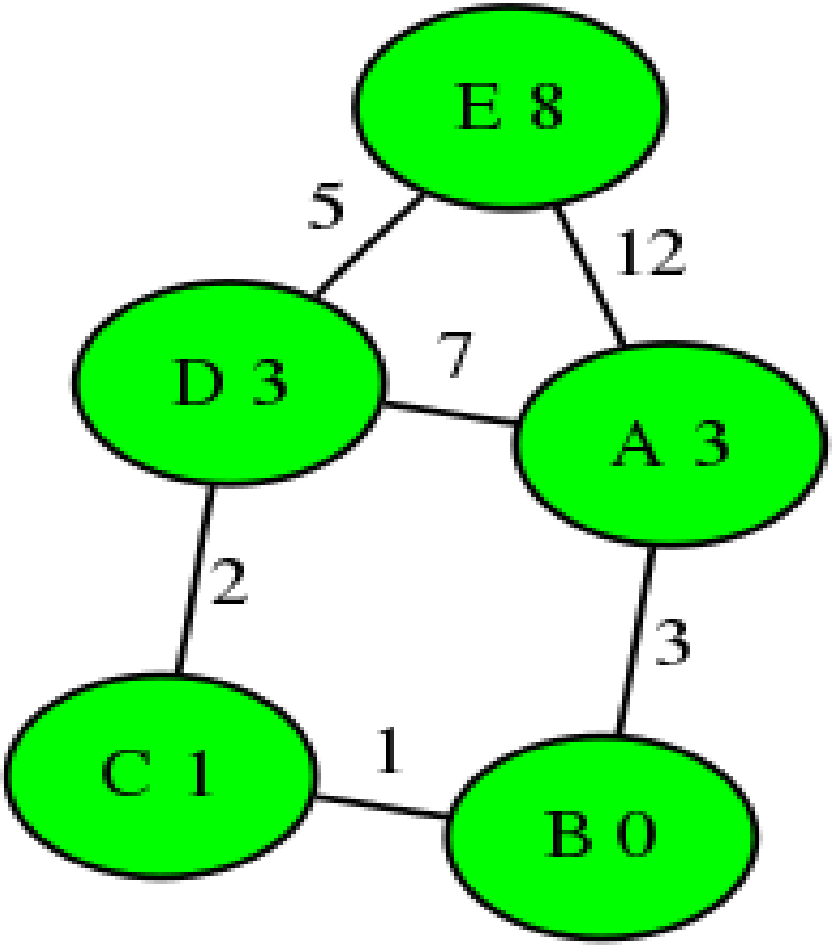
Priority Queue: D(3), E(15)

Example(cont)



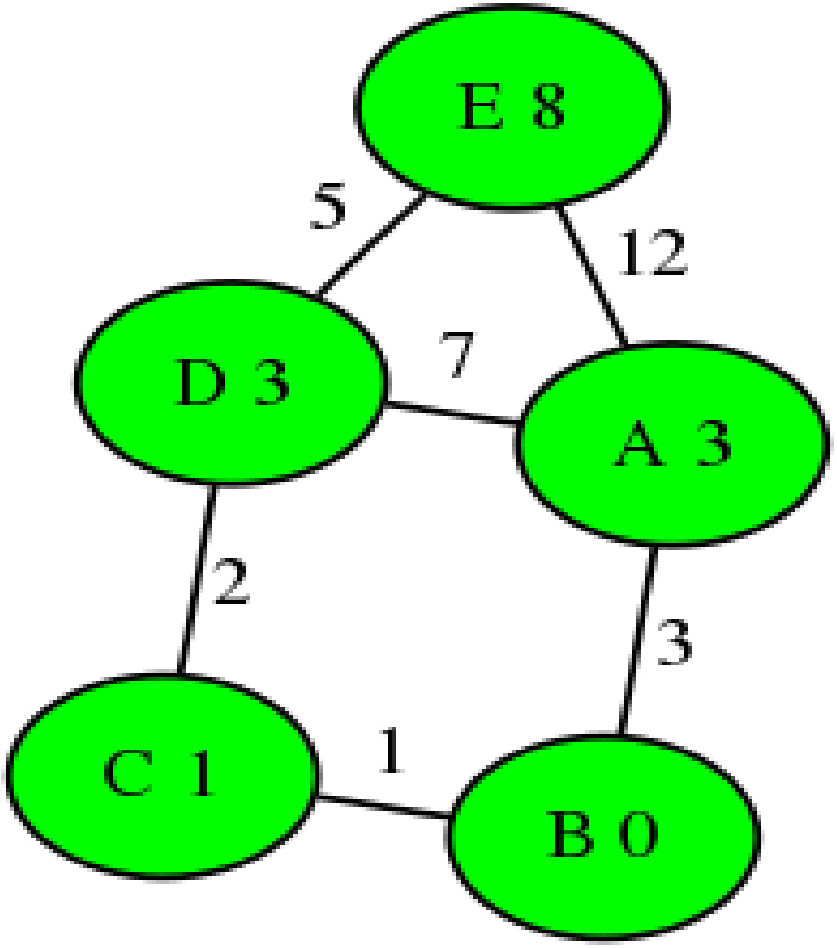
Priority Queue: E(8) E(15)

Example(cont)



Priority Queue: E(15)

Example(cont)



No more vertices are on the queue and we have our answer: 8

Floyd-Warshall's Algorithm

Another way of solving the same problem is Floyd-Warshall's Algorithm.

Advantages:

- Much easier to code.
- You get more. All pairs of shortest paths are solved.

Disadvantages:

- Slower. O(V^3).
- Harder to understand.

Pseudocode

Initialise dists to the adjacency matrix of the graph

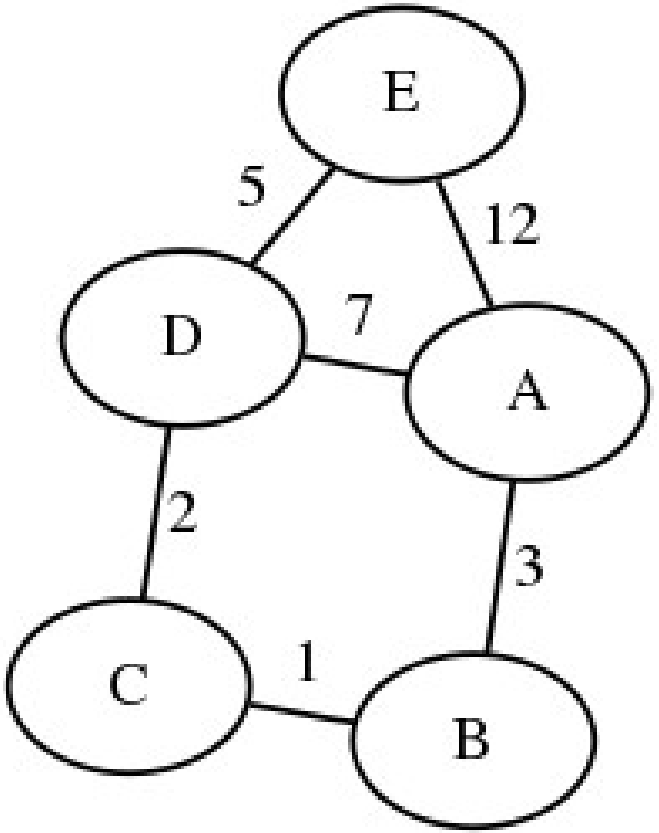
```
for(int k = 0; k < N; ++k)
{
    for(int i = 0; i < N; ++i)
    {
        for(int j = 0; j < N; ++j)
        {
            dists[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

Floyd-Warshall's Algorithm

Why does it work?

It works by iteratively choosing a vertex on the graph as a 'waypoint.' If the path through the waypoint is shorter than the current shortest path the shortest path cost is updated.

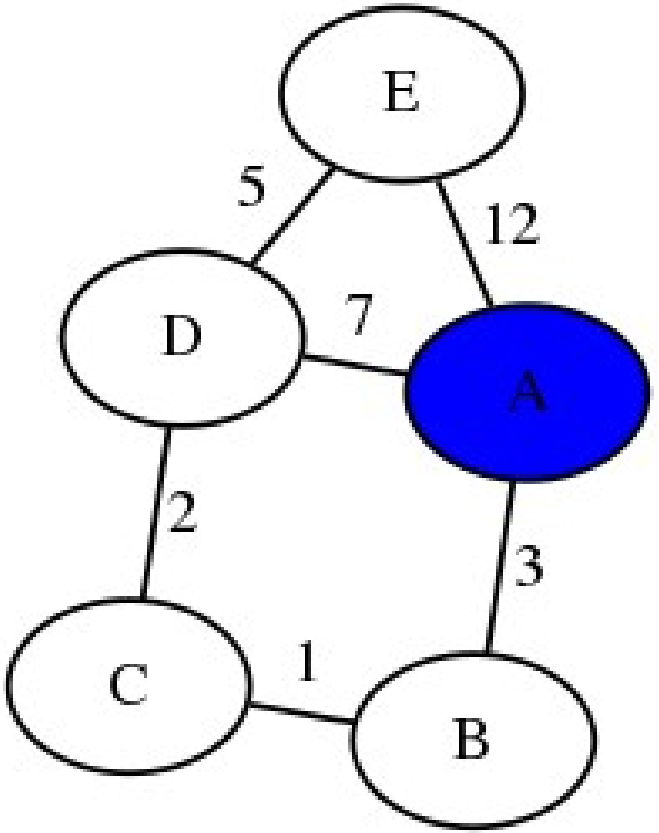
Example



Adjacency Matrix:

0	3	∞	7	12
3	0	1	∞	∞
∞	1	0	2	∞
7	∞	2	0	5
12	∞	∞	5	0

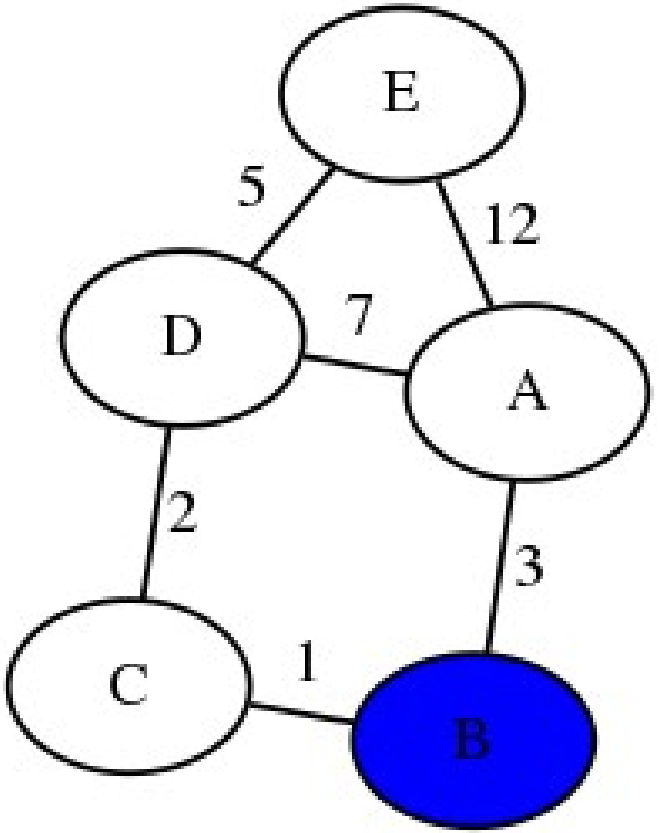
Example(cont)



Adjacency Matrix:

0	3	∞	7	12
3	0	1	**10**	**15**
∞	1	0	2	∞
7	**6**	2	0	5
12	**15**	∞	5	0

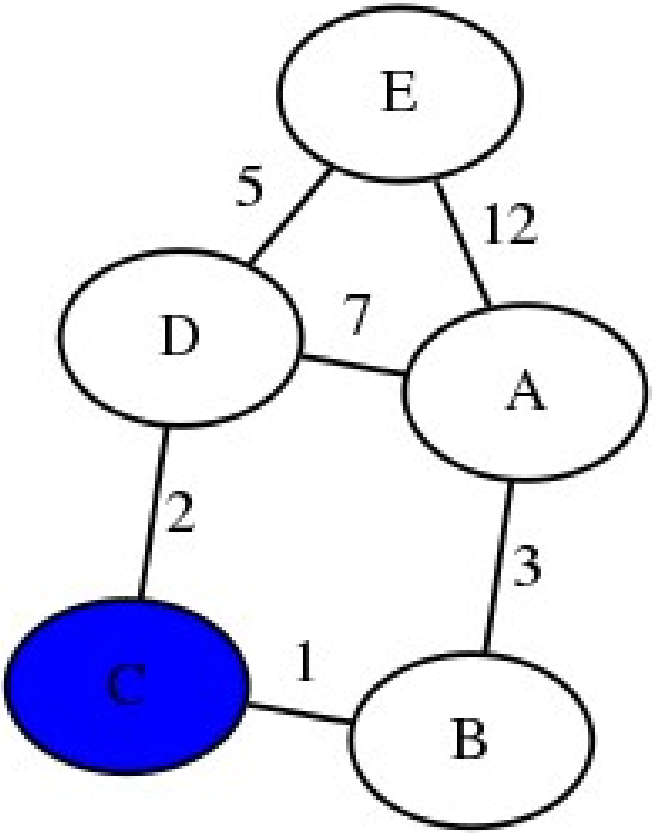
Example(cont)



Adjacency Matrix:

0	3	**4**	7	12
3	0	1	10	15
4	1	0	2	**16**
7	6	2	0	5
12	15	**16**	5	0

Example(cont)



Adjacency Matrix:

0 3 4 **6** 12

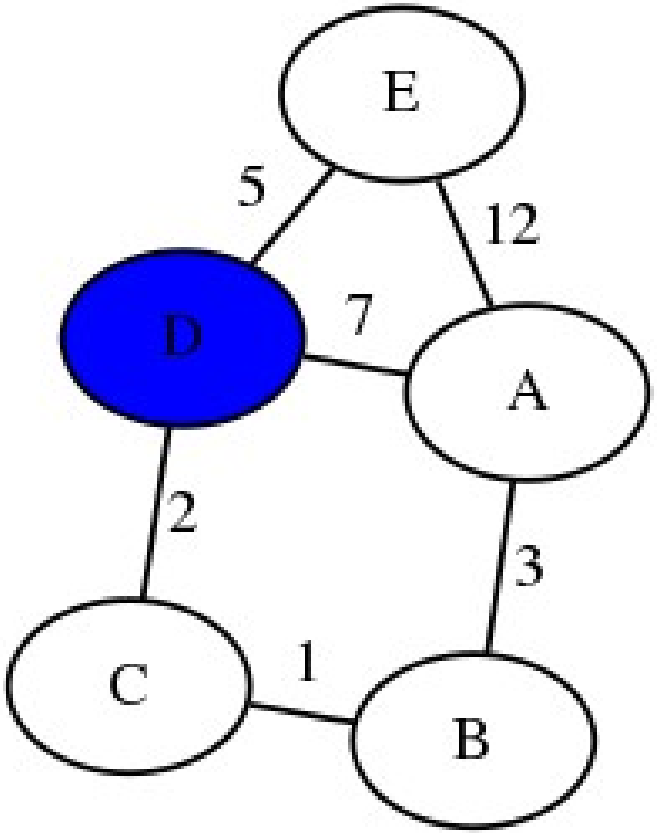
3 0 1 **3** 15

4 1 0 2 16

6 **3** 2 0 5

12 15 16 5 0

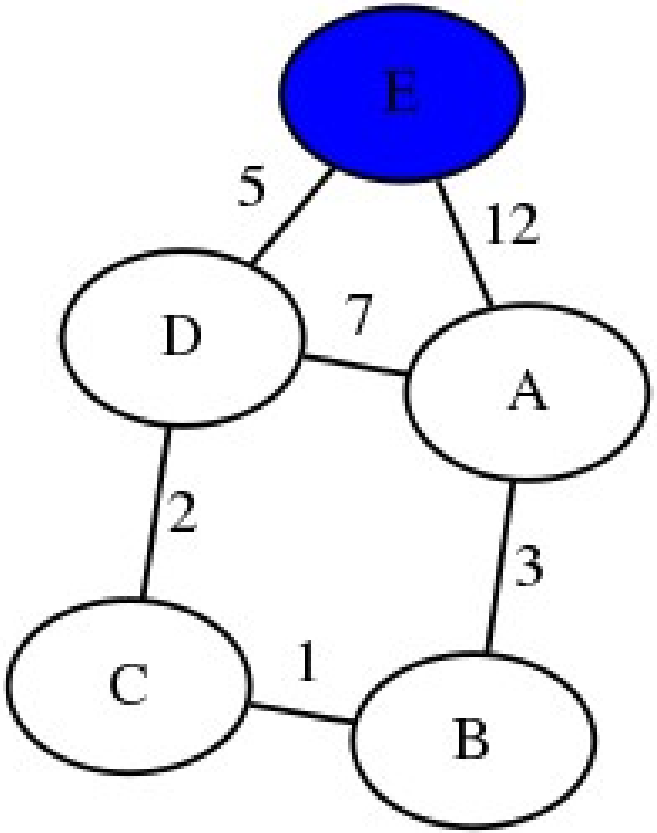
Example(cont)



Adjacency Matrix:

0	3	4	6	**11**
3	0	1	3	**8**
4	1	0	2	**7**
6	3	2	0	5
11	**8**	**7**	5	0

Example(cont)



Adjacency Matrix:

0	3	4	6	11
3	0	1	3	8
4	1	0	2	7
6	3	2	0	5
11	8	7	5	0

The algorithm has finished with 8 as the answer.