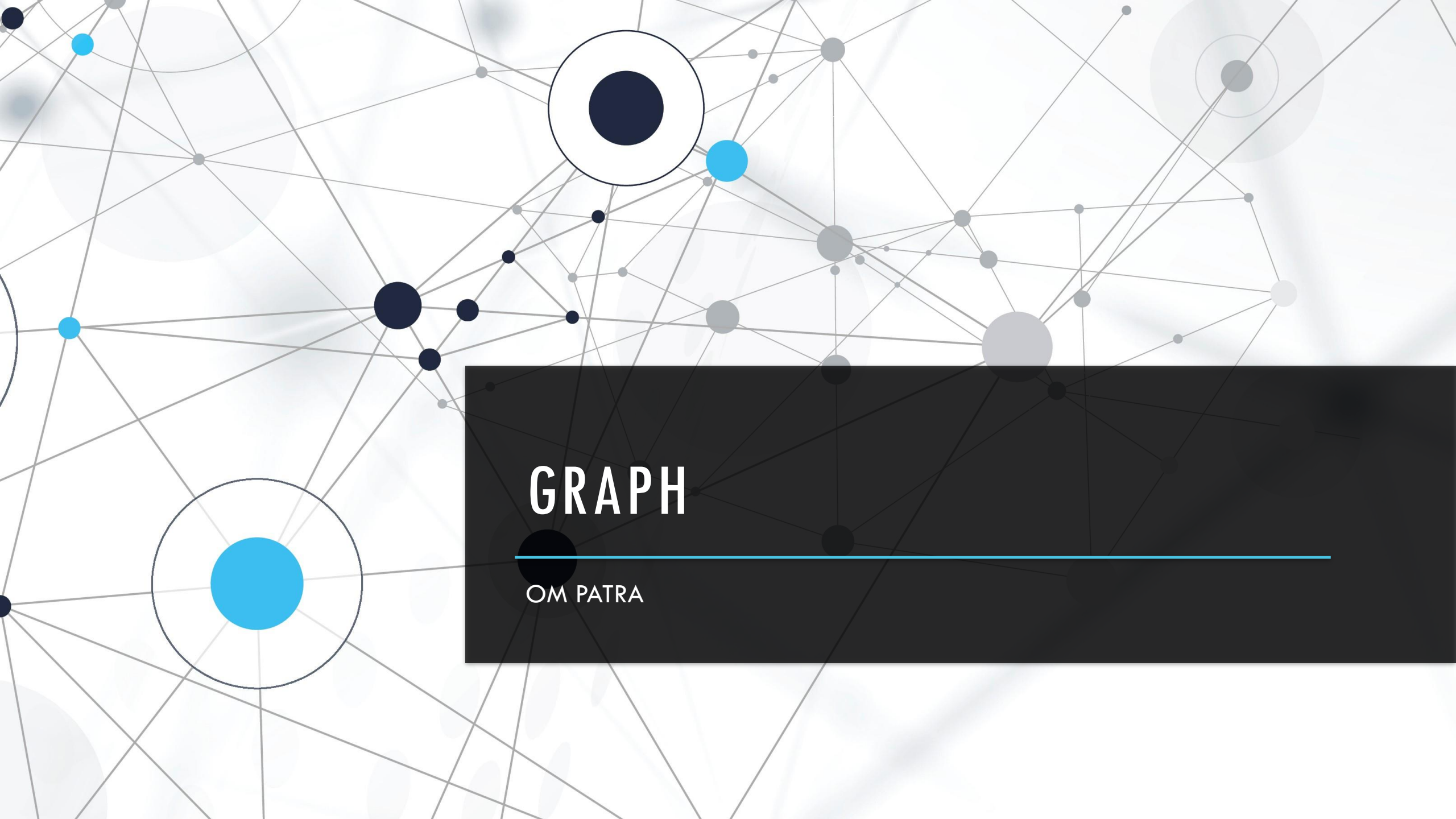


GRAPH

OM PATRA

TOPICS

- Prims Algorithm
- Kruskal's Algorithm

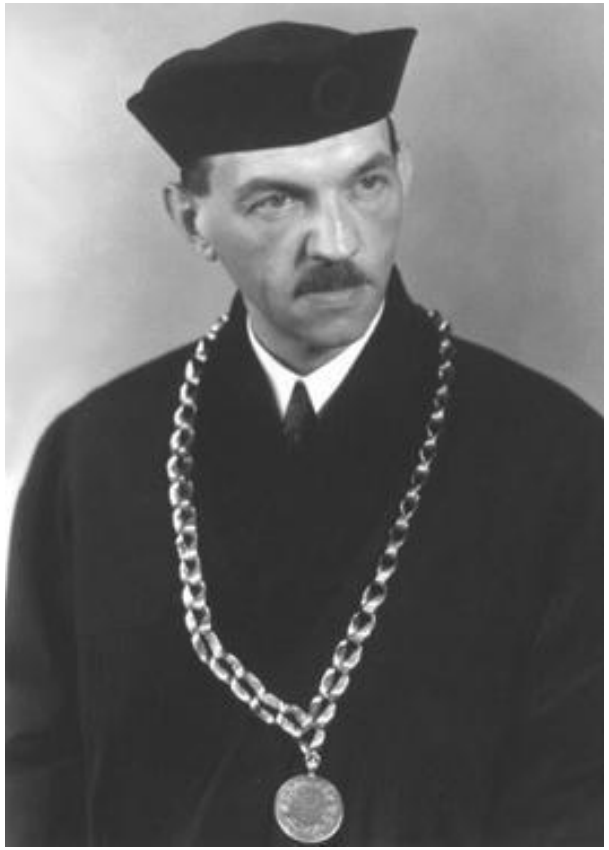
Minimum Spanning Tree

- Dijkstra Algorithm
- Bellman-Ford Algorithm

Single Source Shortest Path Algorithm

JARNÍK'S ALGORITHM

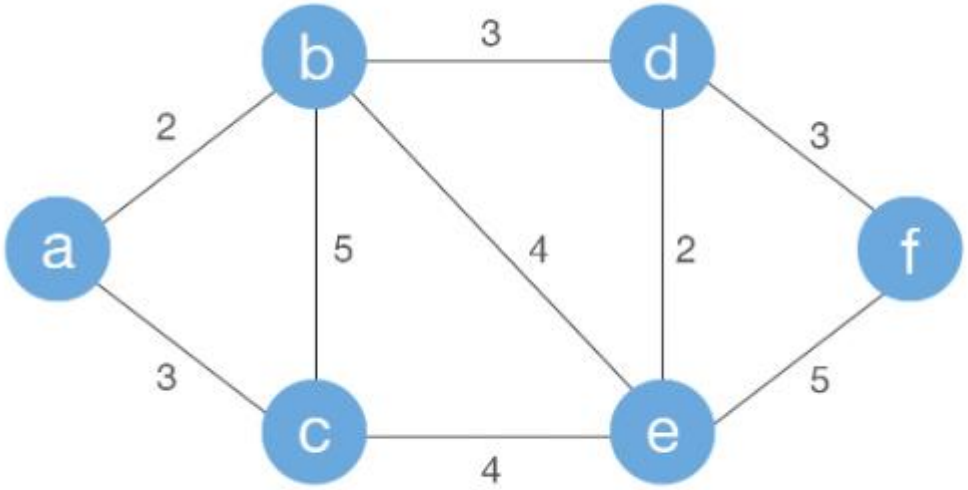
In computer science and combinatorial optimization, Jarník is known for an algorithm for constructing minimum spanning trees that he published in 1930, in response to the publication of Borůvka's algorithm by another Czech mathematician. Jarník's algorithm builds a tree from a single starting vertex of a given weighted graph by repeatedly adding the cheapest connection to any other vertex, until all vertices have been connected.

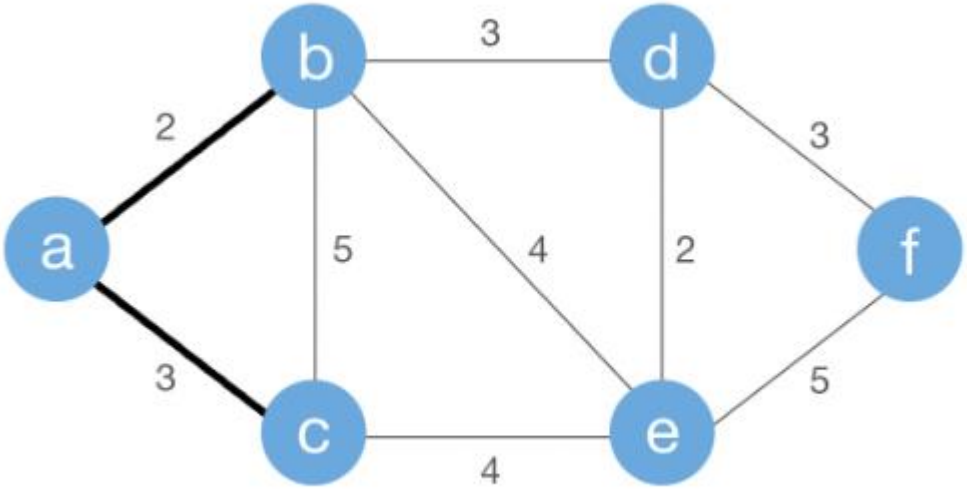


BASIC APPROACH

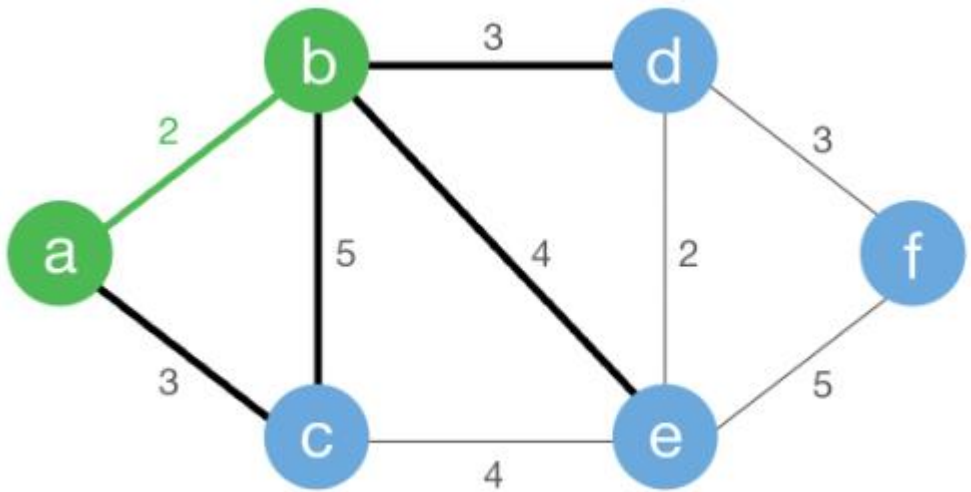
- Start with one(any) vertex
- Branch outwards to grow your connected component.
- Consider only edges that leave the connected component.
- Add smallest considered edge to your connected component.
- Continue until a spanning tree is created.

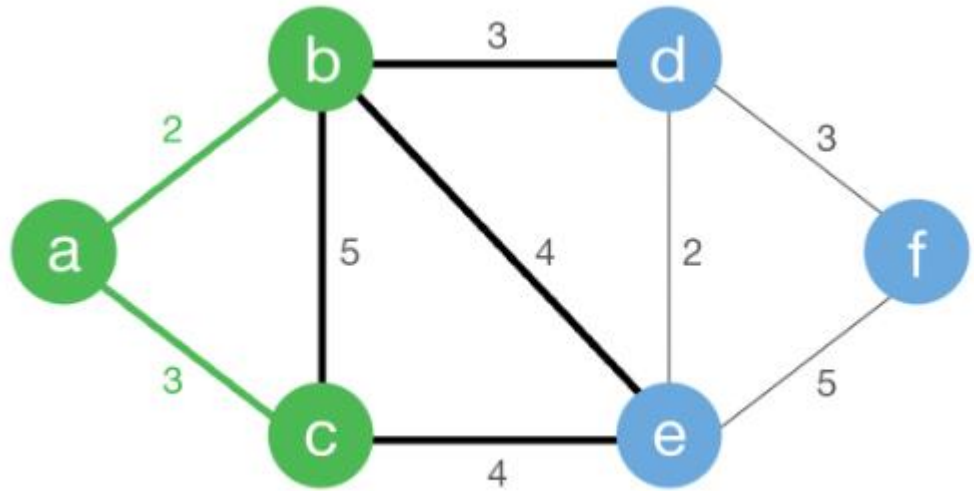
EXAMPLE

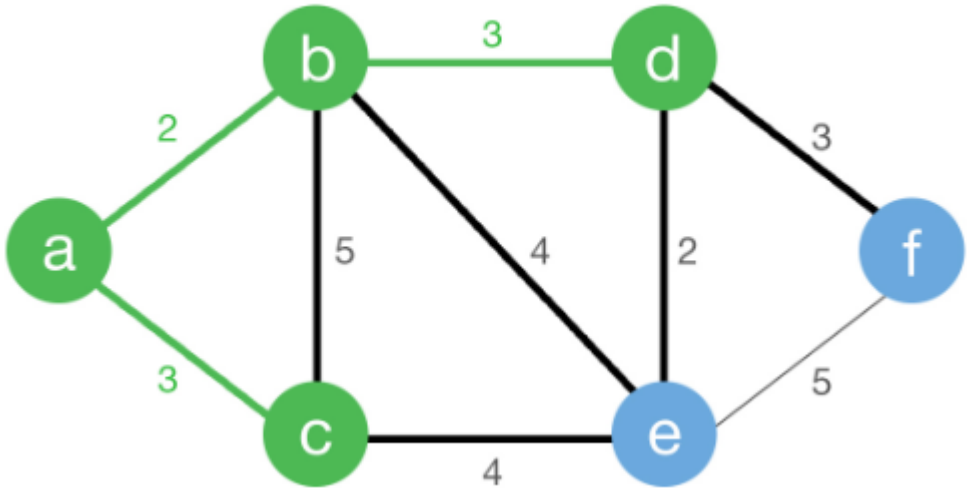


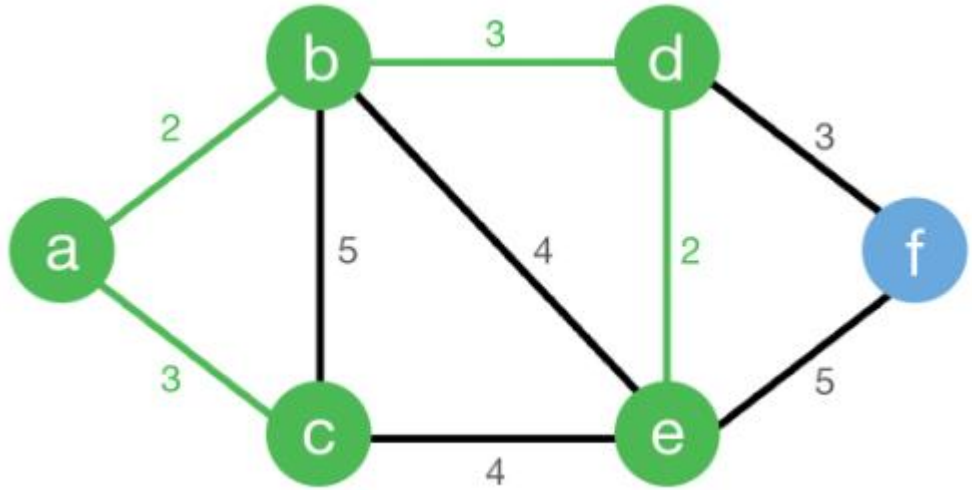


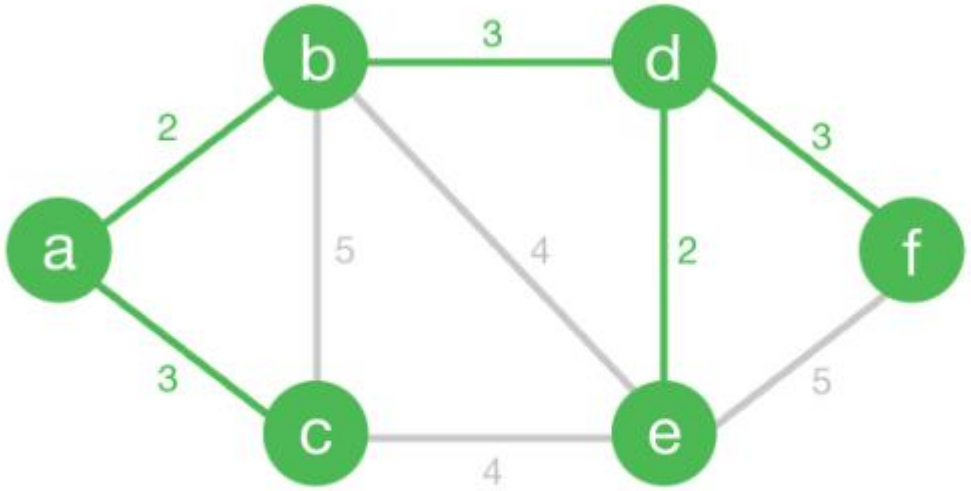
Start with one(any) vertex and branch outwards to grow your connected component


b
3
d
2
3
a
5
4
2
f
3
5
c
e
4


b
3
d
2
3
a
5
4
2
f
3
5
c
e
4


a
b
c
d
e
f
2
3
5
3
4
4
2
3
5


a
b
c
d
e
f
2
3
5
3
4
4
2
3
5



Edge weight total = 13

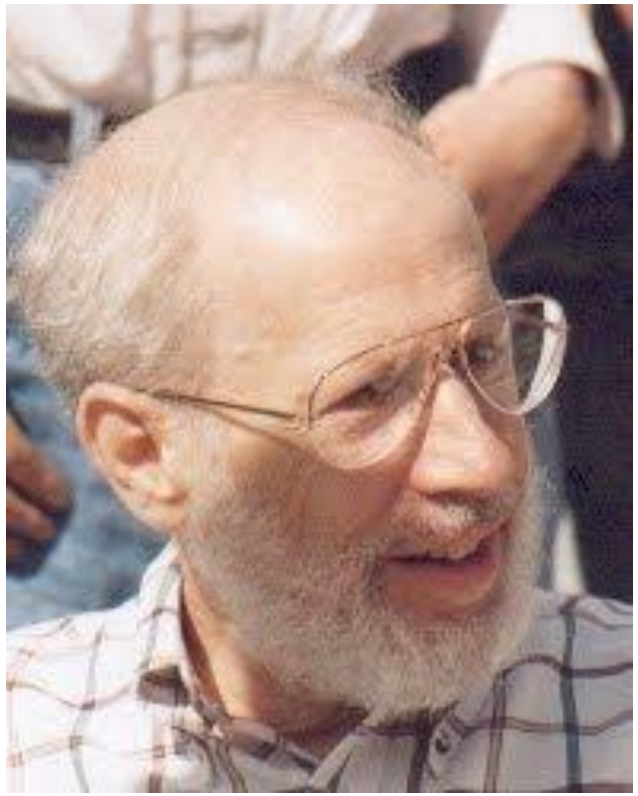
This algorithm runs until the number of edges in MST is equal to the number of vertices in the graph minus 1. So in the example above, the number of vertices in the graph is 6, so Prim's algorithm will run until the MST contains 5 edges. Once the algorithm is complete, the MST will have successfully connected all vertices in the graph with the minimum weighted edges.

KRUSKAL'S ALGORITHM

This algorithm first appeared in *Proceedings of the American Mathematical Society*, pp. 48–50 in 1956, and was written by Joseph Kruskal.

It is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.

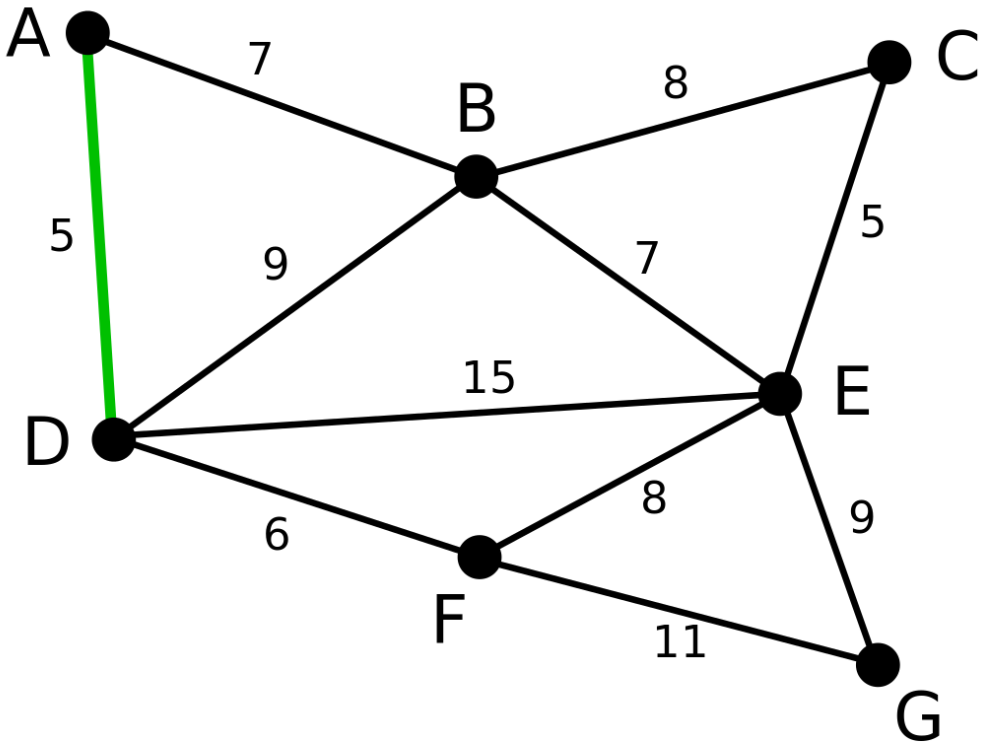
It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step



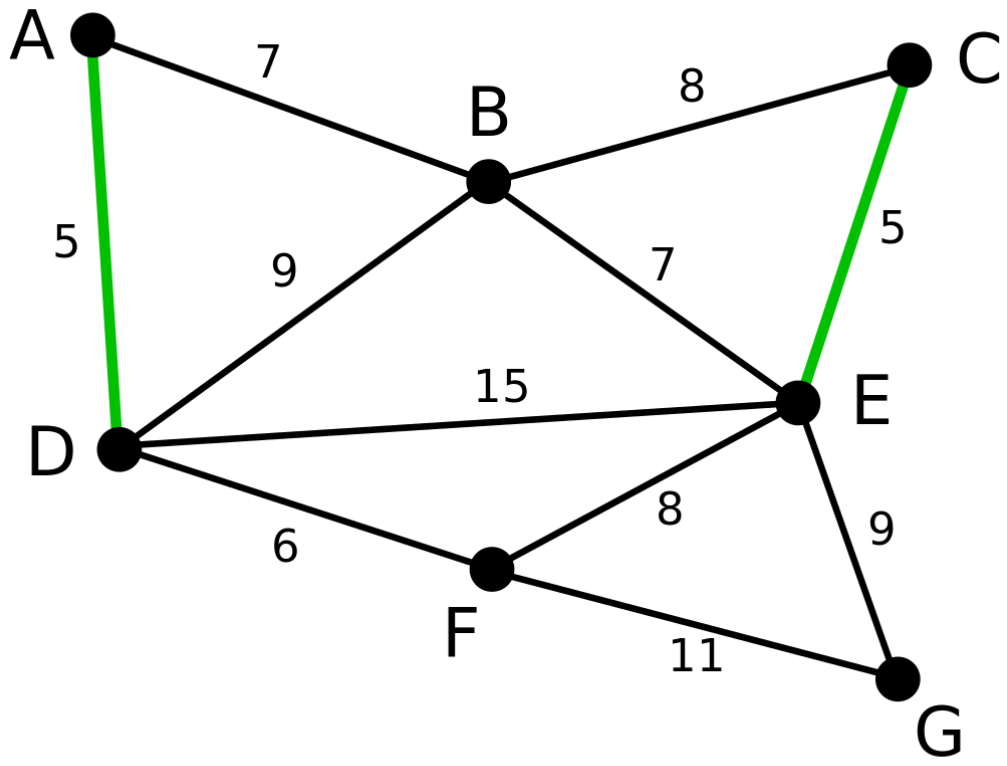
BASIC APPROACH

- Start with |V| disjoint components
- Consider lesser weight edges to incrementally connect components
- Make certain to avoid cycles
- Continue until spanning tree is created

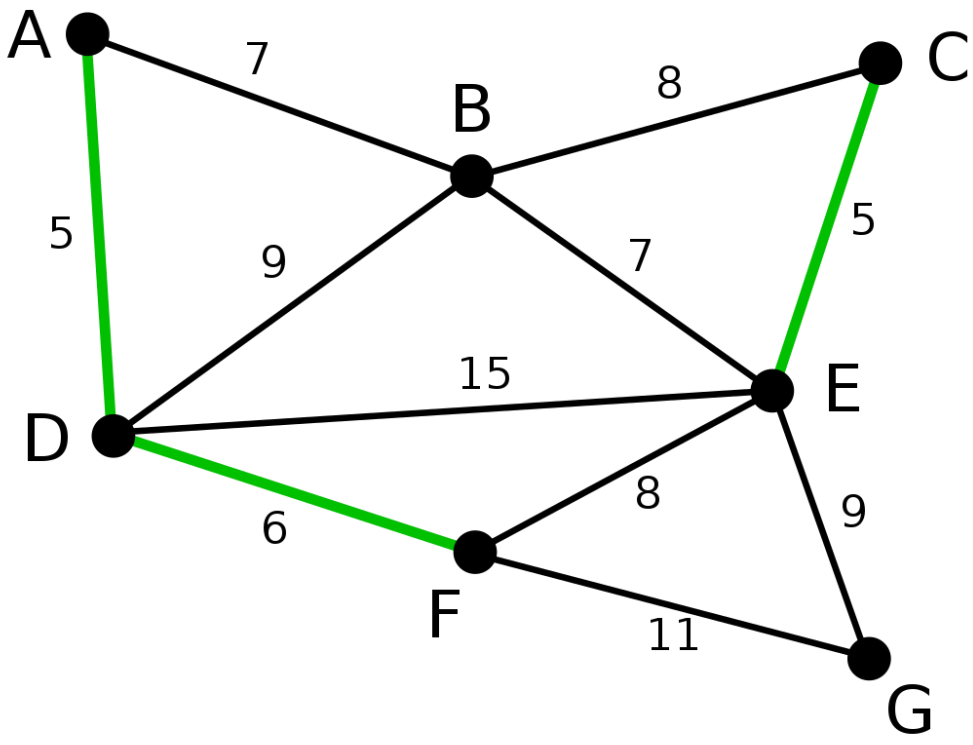
EXAMPLE



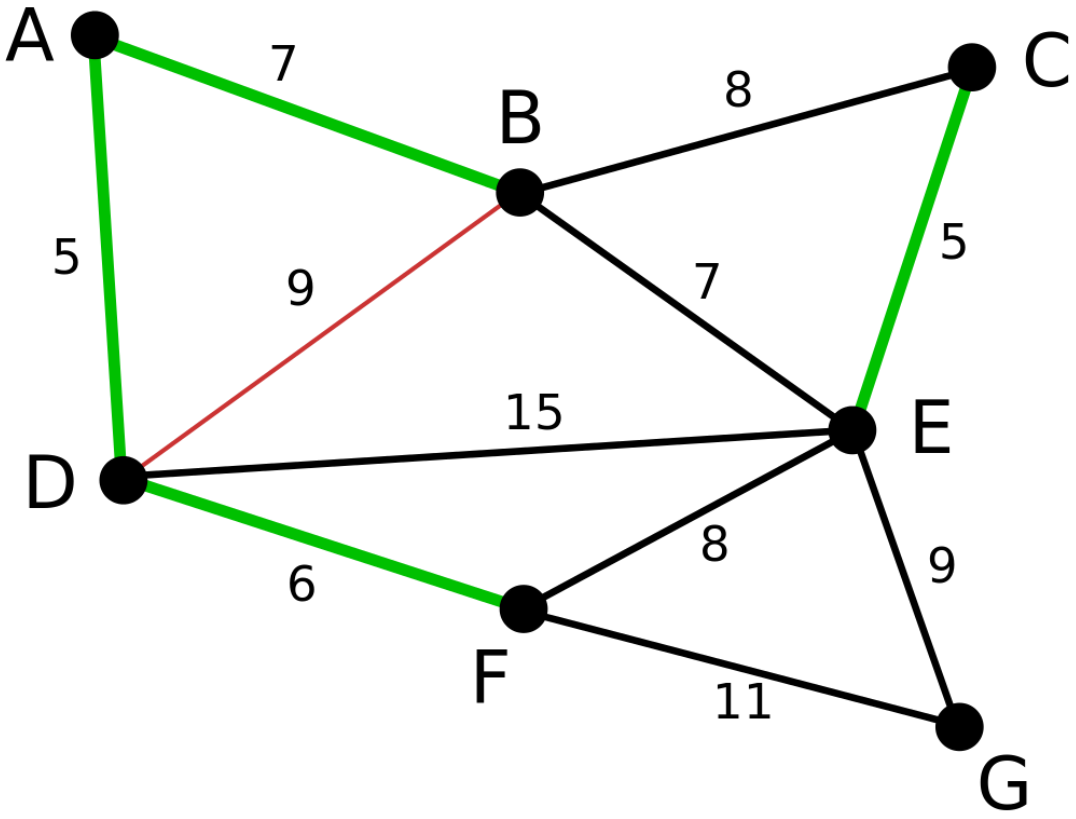
AD and **CE** are the shortest edges, with length 5, and **AD** has been arbitrarily chosen, so it is highlighted.



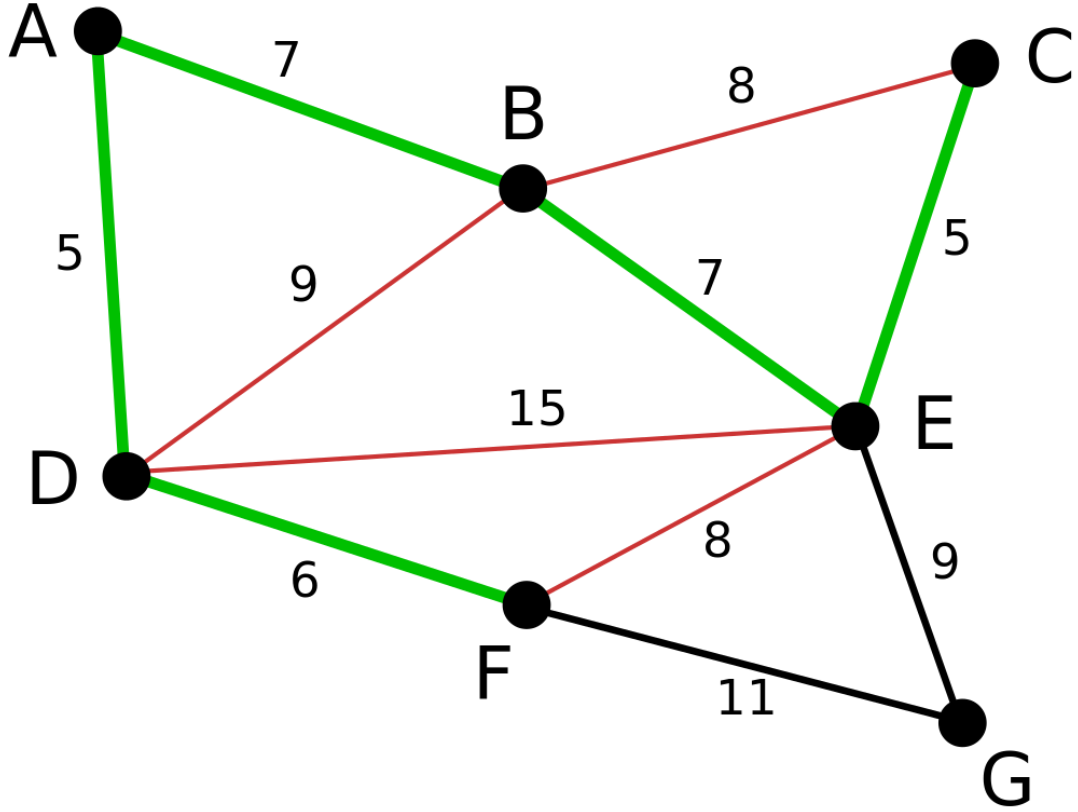
CE is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.



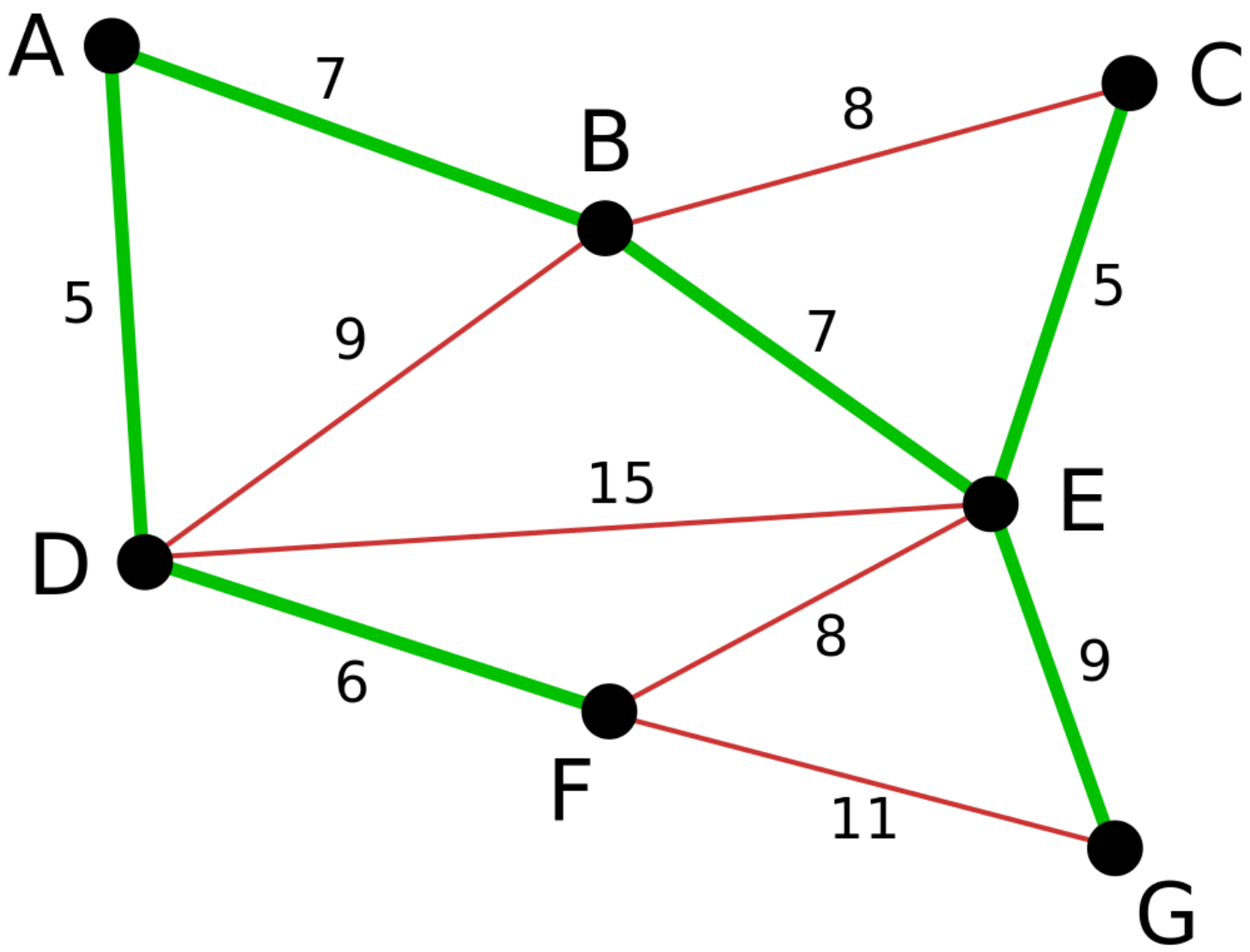
The next edge, **DF** with length 6, is highlighted using much the same method.



The next-shortest edges are **AB** and **BE**, both with length 7. **AB** is chosen arbitrarily, and is highlighted. The edge **BD** has been highlighted in red, because there already exists a path (in green) between **B** and **D**, so it would form a cycle (**ABD**) if it were chosen.



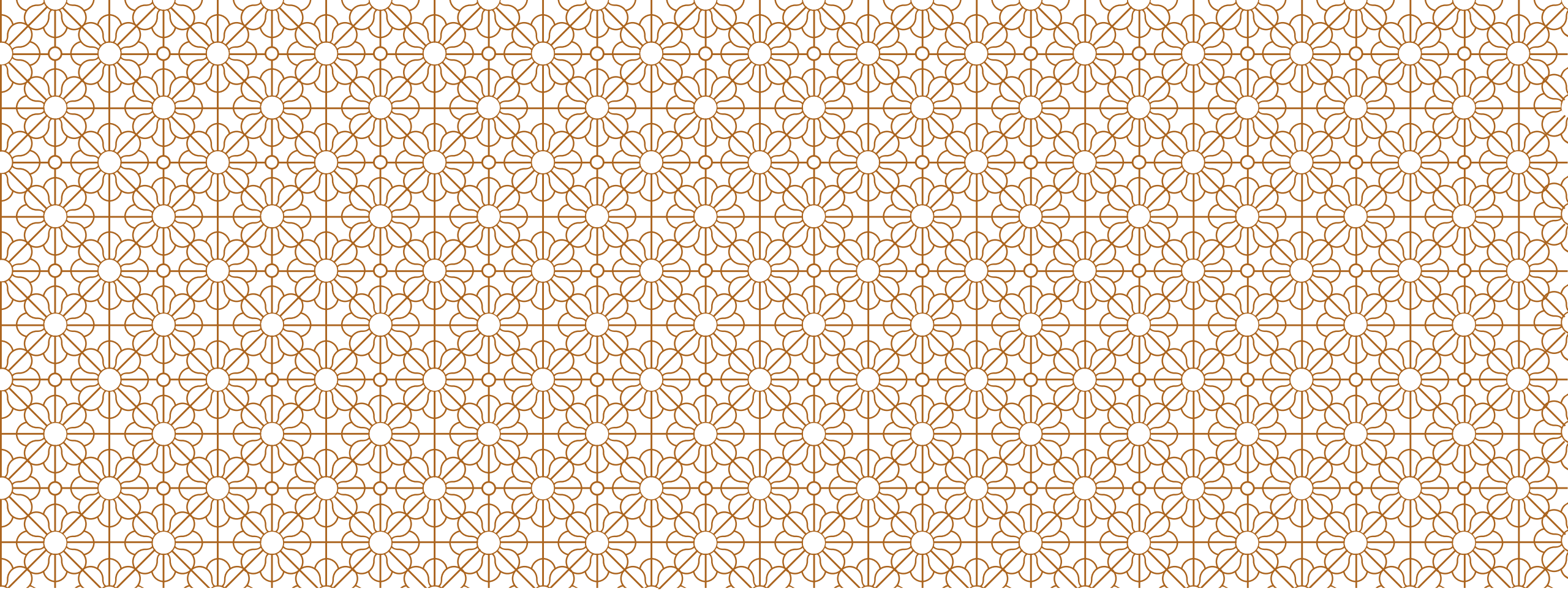
The process continues to highlight the next-smallest edge, **BE** with length 7. Many more edges are highlighted in red at this stage: **BC** because it would form the loop **BCE**, **DE** because it would form the loop **DEBA**, and **FE** because it would form **FEBAD**.



Finally, the process finishes with the edge **EG** of length 9, and the minimum spanning tree is found.

APPLICATION

- They find numerous applications in image processing. For example, if you have an image of cells on a slide, then you could use the minimum spanning tree of the graph formed by the nuclei to describe the arrangement of these cells.
- Taxonomy Construction in construction of road or telephone networks. We would like to connect places/houses with the minimum length of road/wire possible. This is exactly the same as computing the minimum spanning tree.
- They are an important part of many approximation algorithms for NP-hard and NP-complete problems. For example, the first step in most approximation algorithms for the Steiner tree problem requires computing the MST. It is also the first step in the Christofede's algorithm for the traveling salesman problem.Travelling Salesman Problem Approximation



SINGLE SHORTEST PATH ALGORITHM

Djistra's Algorithm
Bellman-Ford Algorithm

DIJKSTRA'S ALGORITHIM

It is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

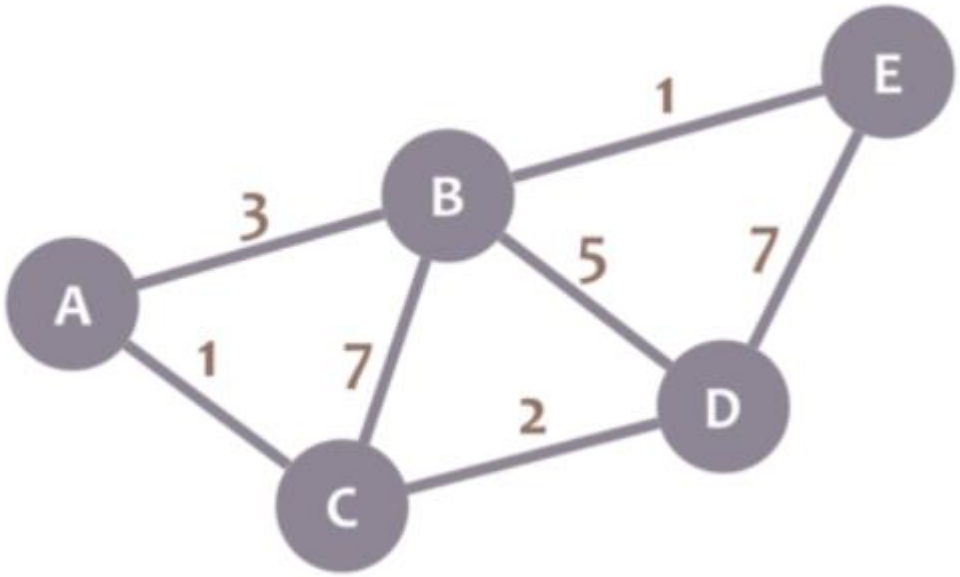
The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

BASIC APPROACH

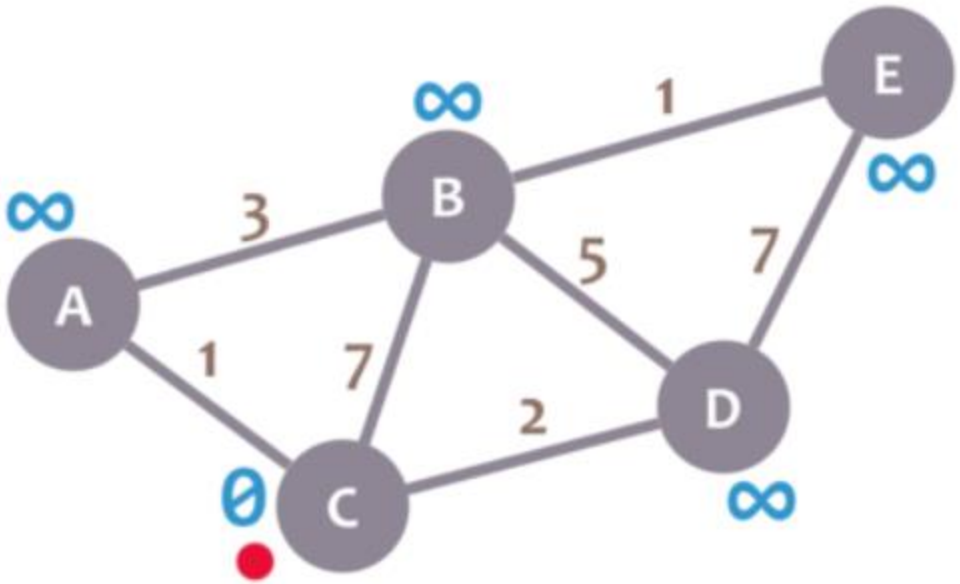
```
dist[s] ←o                                  (distance to source vertex is zero)
for all v ∈ V–{s}
      do dist[v] ←∞                         (set all other distances to infinity)
S←∅                                         (S, the set of visited vertices is initially empty)
Q←V                                         (Q, the queue initially contains all vertices)
while Q ≠∅                                  (while the queue is not empty)
do  u ← mindistance(Q,dist)                 (select the element of Q with the min. distance)
    S←S∪{u}                                 (add u to list of visited vertices)
     for all v ∈ neighbors[u]
          do if  dist[v] > dist[u] + w(u, v)        (if new shortest path found)
                then     d[v] ←d[u] + w(u, v)      (set new value of shortest path)
                                                   (if desired, add traceback code)

return dist
```

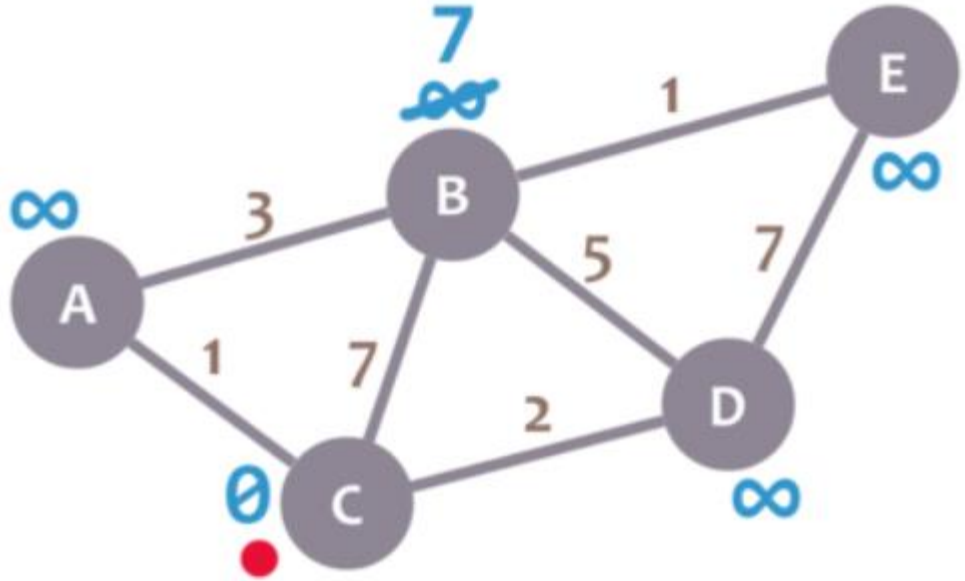
EXAMPLE



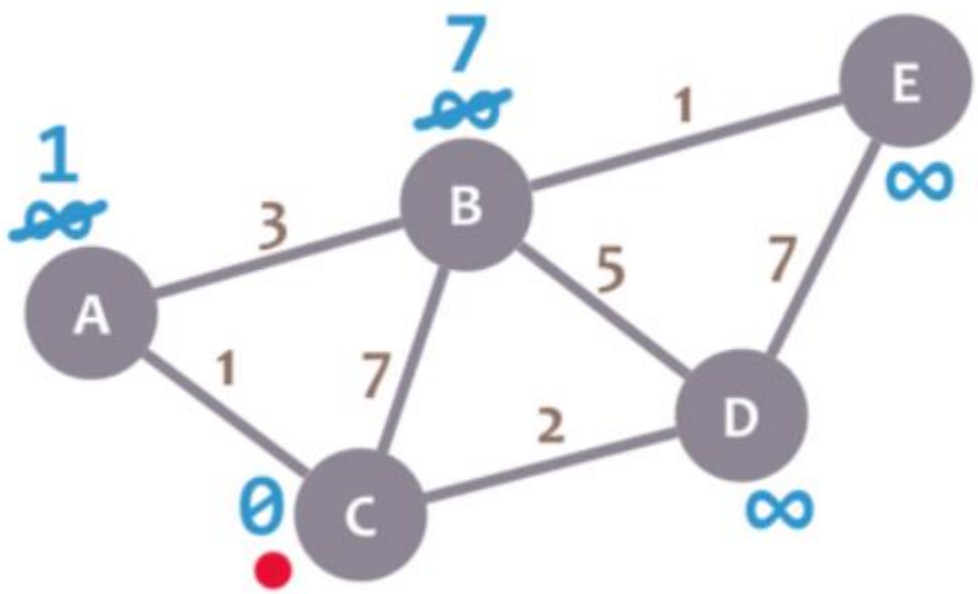
During the algorithm execution, we'll mark every node with its *minimum distance* to node C (our selected node). For node C, this distance is 0. For the rest of nodes, as we still don't know that minimum distance, it starts being infinity (∞):



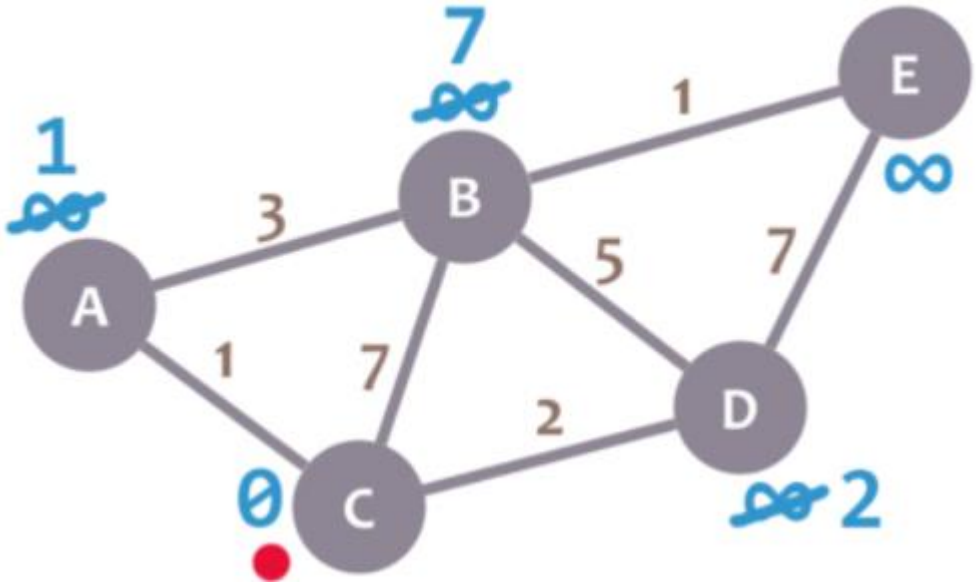
Now, we check the neighbors of our current node (A, B and D) in no specific order. Let's begin with B. We add the minimum distance of the current node (in this case, 0) with the weight of the edge that connects our current node with B (in this case, 7), and we obtain 0 + 7 = 7. We compare that



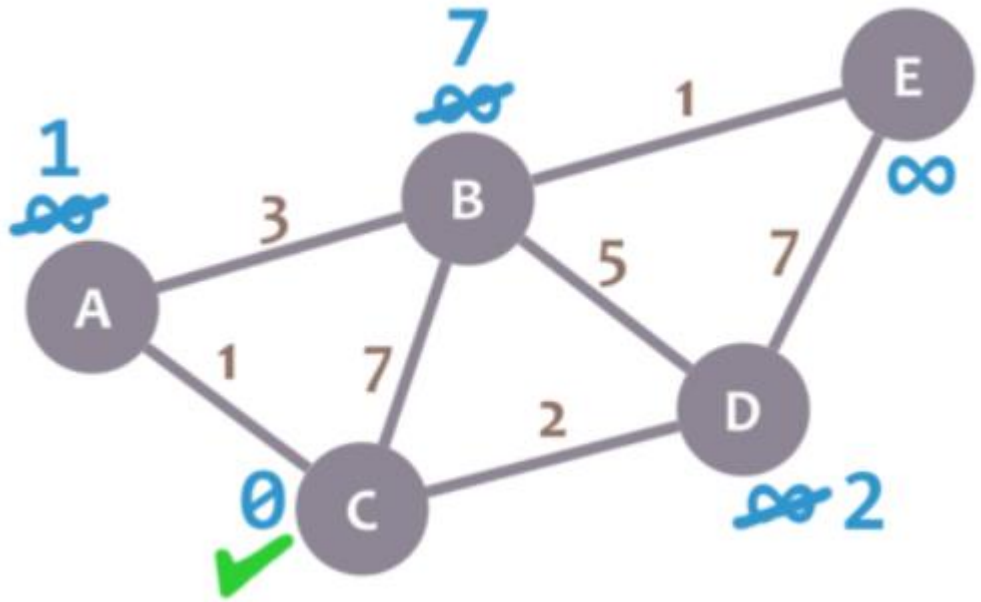
We add 0 (the minimum distance of C, our current node) with 1 (the weight of the edge connecting our current node with A) to obtain 1. We compare that 1 with the minimum distance of A (infinity), and leave the smallest value



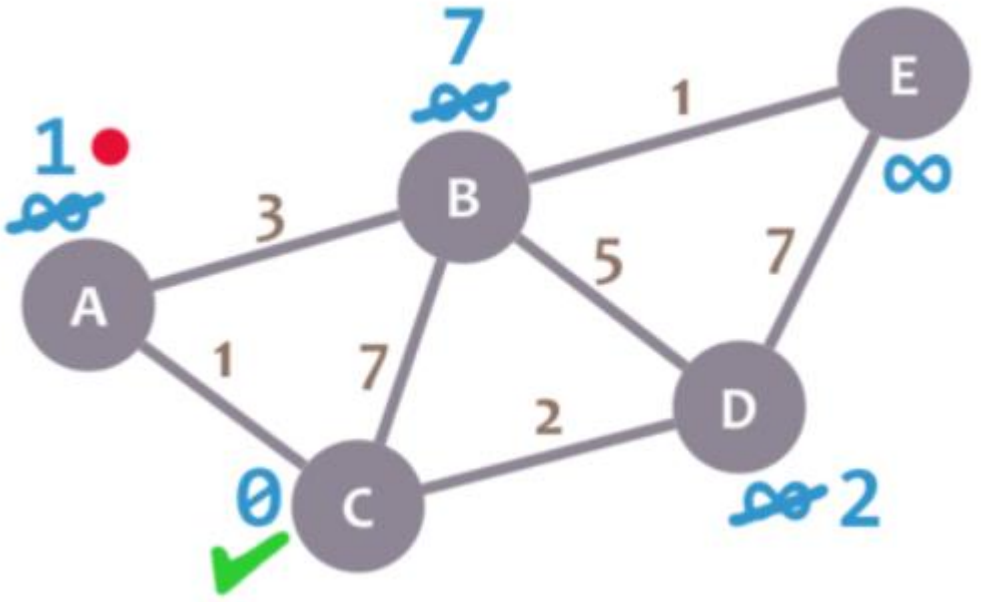
Repeat the same procedure for D



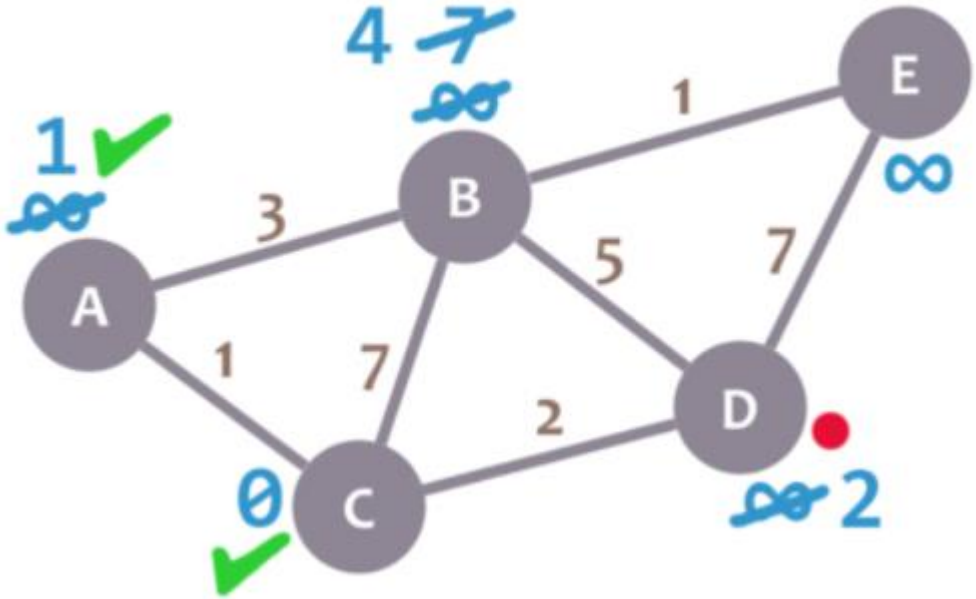
We have checked all the neighbors of C. Because of that, we mark it as *visited*. Let's represent visited nodes with a green check mark



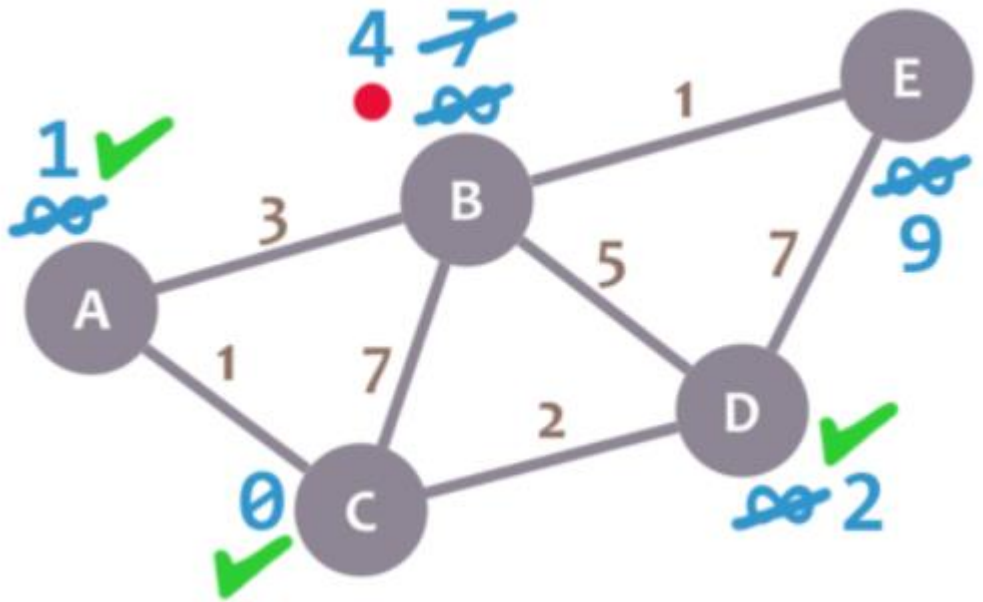
We now need to pick a new *current node*. That node must be the unvisited node with the smallest minimum distance (so, the node with the smallest number and no check mark). That's A. Let's mark it with the red dot



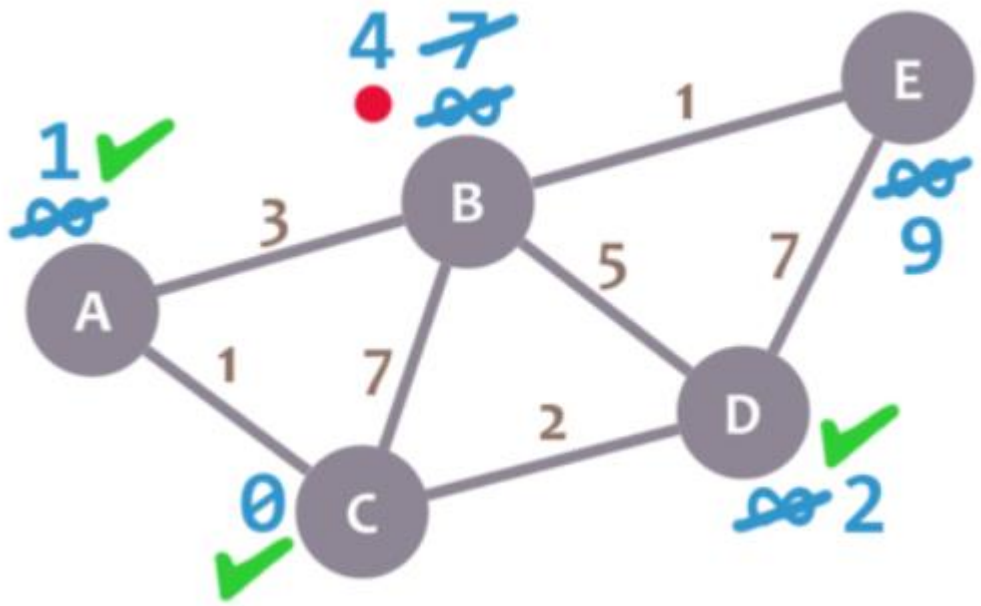
And now we repeat the algorithm. We check the neighbors of our current node, ignoring the visited nodes. For B, we add 1 (the minimum distance of A, our current node) with 3 (the weight of the edge connecting A and B) to obtain 4. We compare that 4 with the minimum distance of B (7) and leave the smallest value



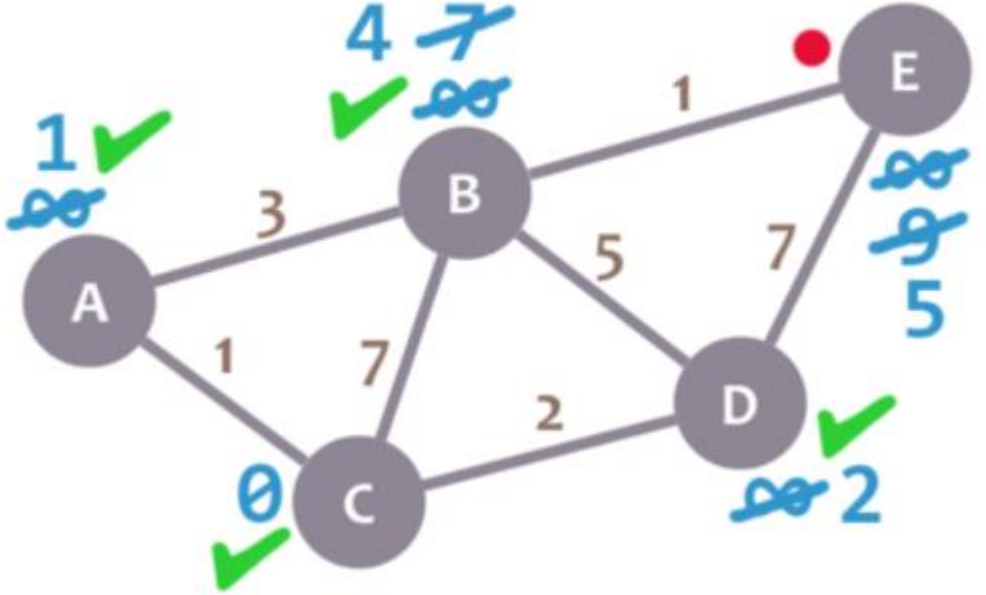
We mark A as visited and pick a new current node: D, which is the non-visited node with the smallest current distance



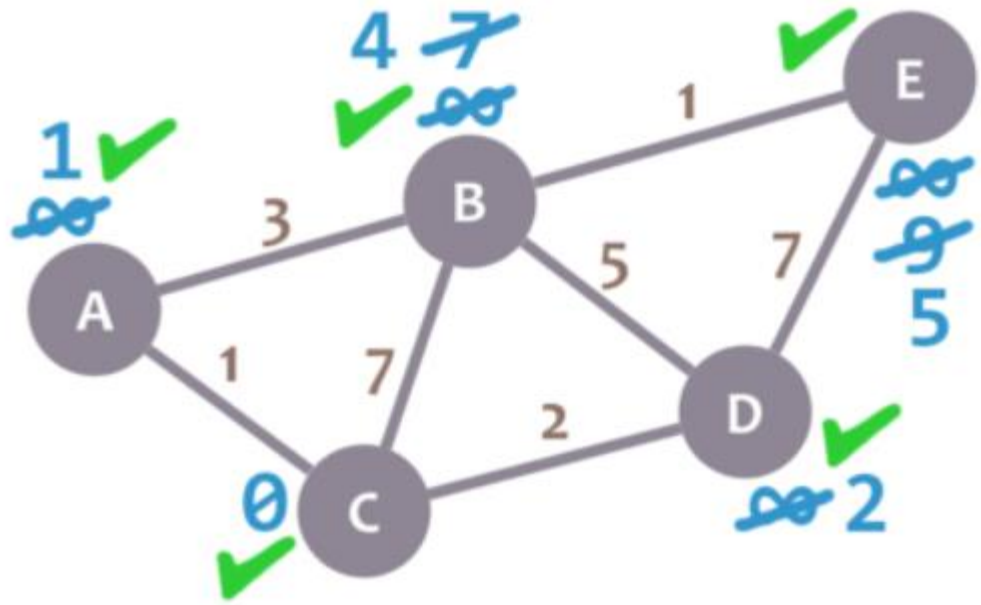
We repeat the algorithm again. This time, we check B and E. For E we compare it with the minimum distance of E (infinity) and leave the smallest one (9).We mark D as visited and set our current node to B



We only need to check E. 4 + 1 = 5, which is less than E's minimum distance (9), so we leave the 5. Then, we mark B as visited and set E as the current node



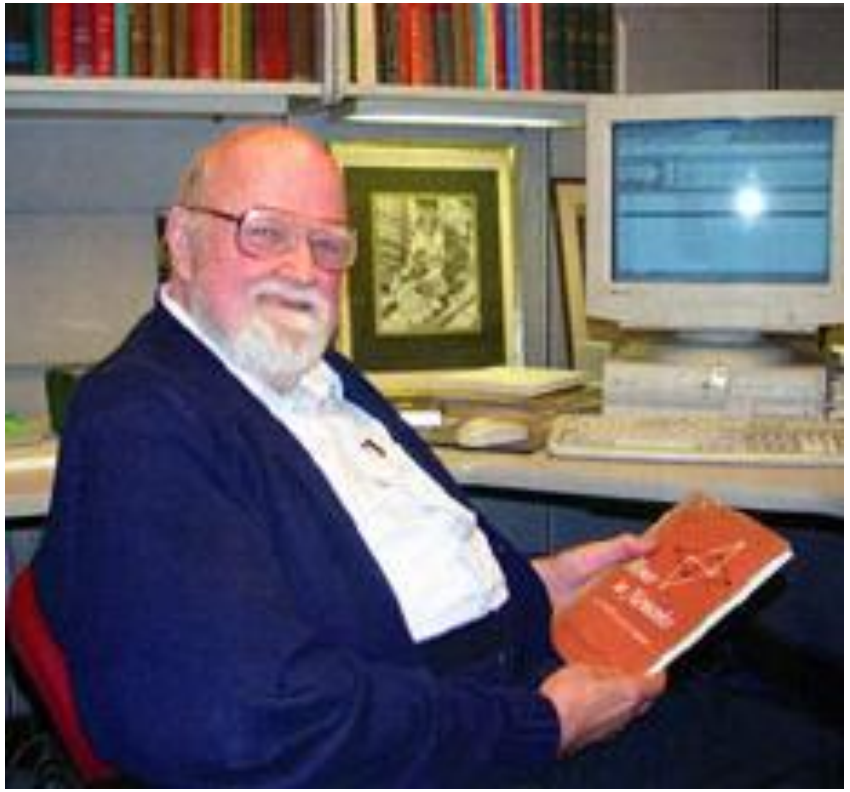
E doesn't have any non-visited neighbors, so we don't need to check anything. We mark it as visited.



As there are not unvisited nodes, we're done! The minimum distance of each node now actually represents the minimum distance from that node to node C (the node we picked as our initial node)

BELLMAN-FORD ALGORITHIM

The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel (1955), but is instead named after Richard Bellman and Lester Ford Jr., who published it in 1958 and 1956, respectively.



BASIC IDEA

Input: Graph and a source vertex *source*

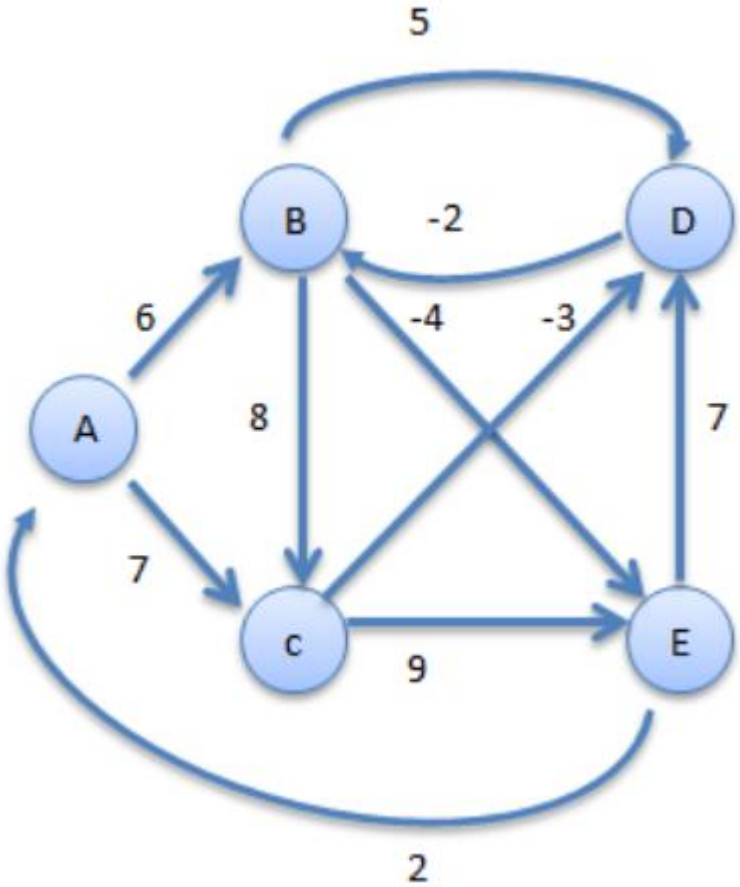
Output: Shortest distance to all vertices from *source*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

BASIC APPROACH

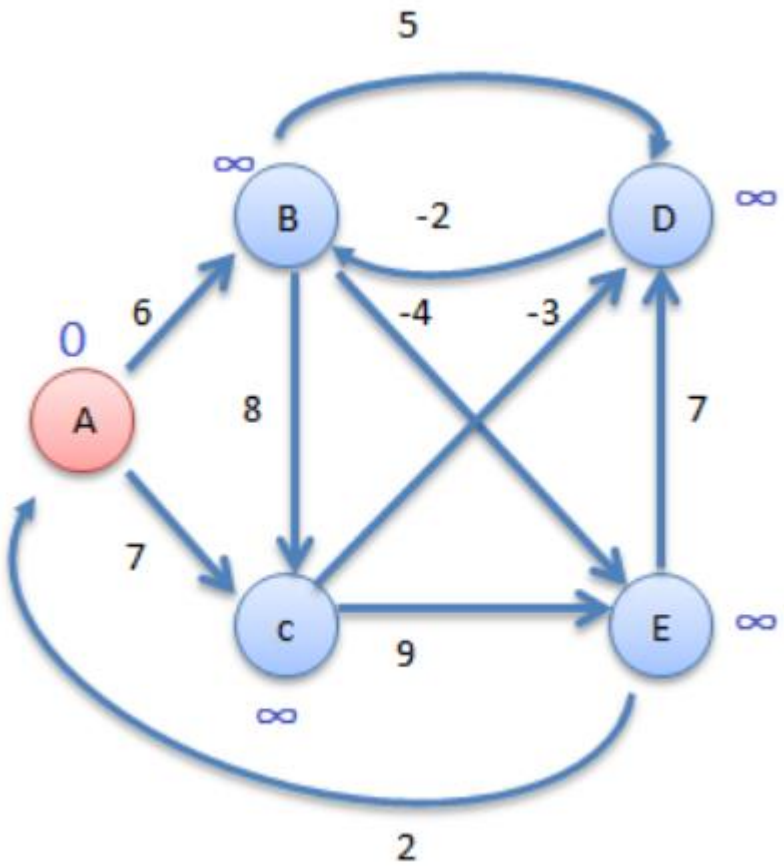
1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0.

2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
…..**a)** Do following for each edge u-v
……If dist[v] > dist[u] + weight of edge uv, then update dist[v]
…….dist[v] = dist[u] + weight of edge uv

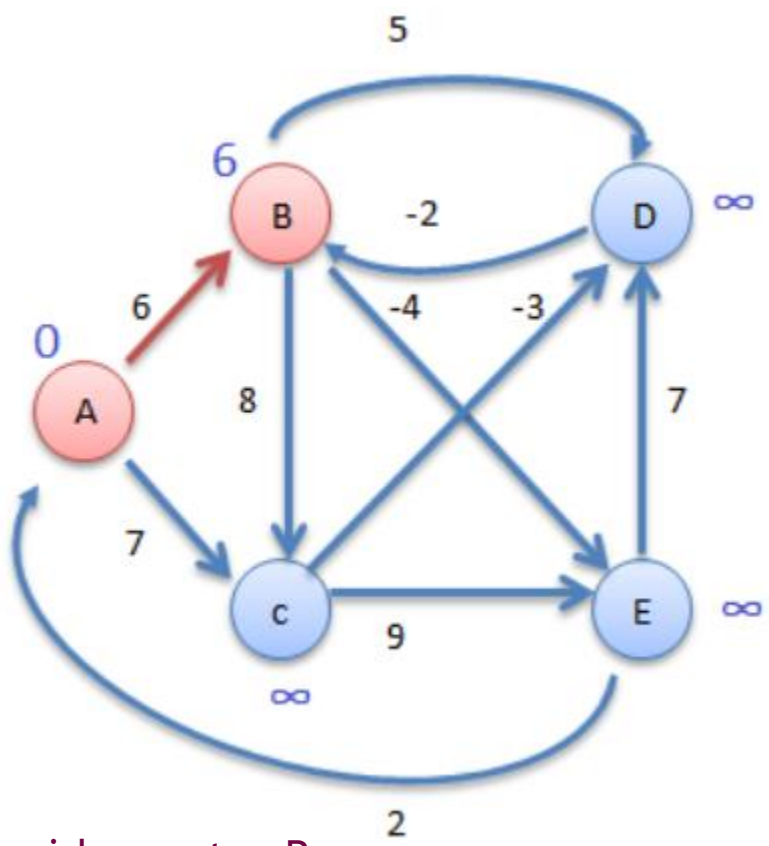
3) This step reports if there is a negative weight cycle in graph. Do following for each edge u-v
If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn’t contain negative weight cycle. **If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle**



Consider A as source vertex

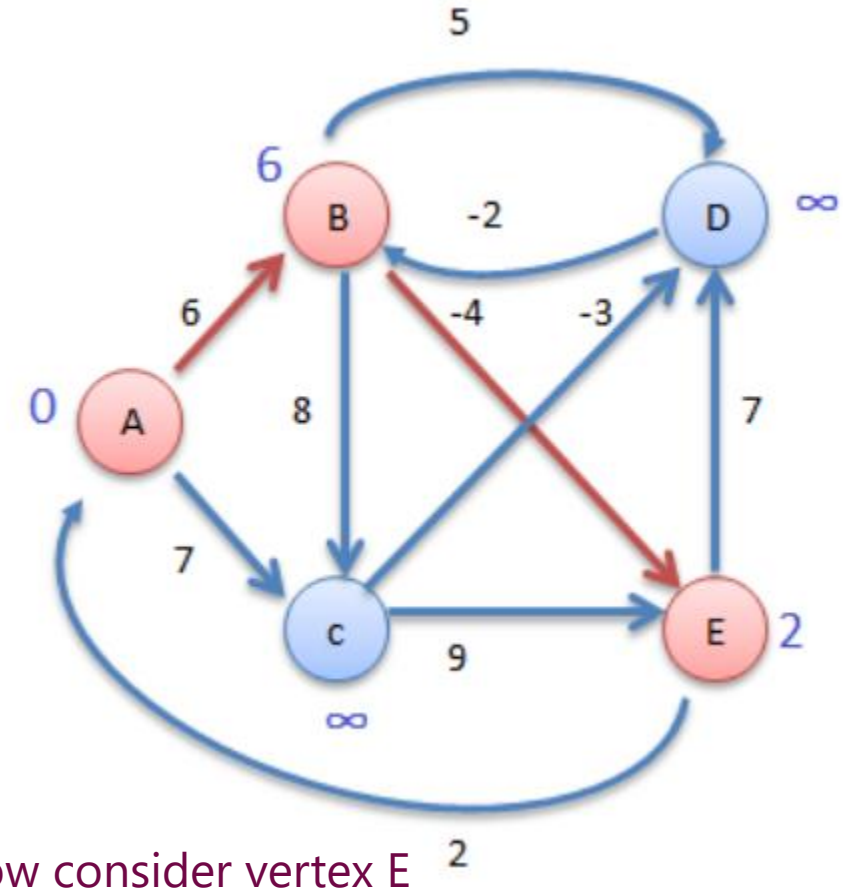


No. of Nodes	A	B	C	D	E
Distance	0	**6**	7	∞	∞
Distance From	A	A	A		



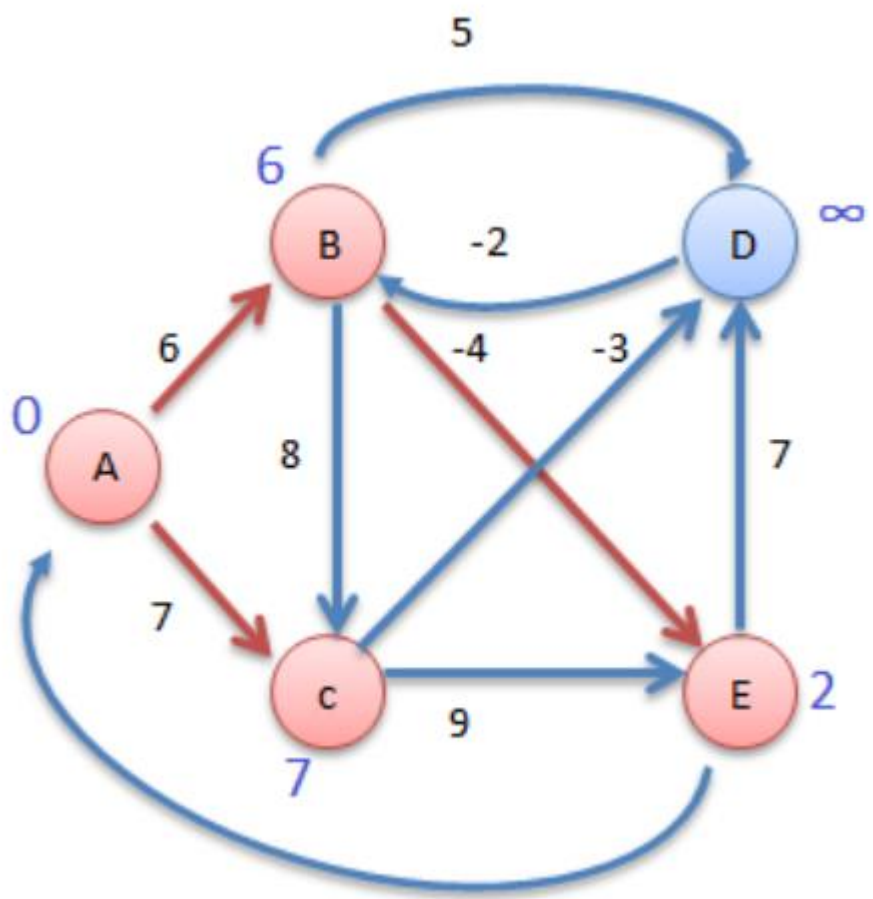
Now consider vertex B

No. of Nodes	A	B	C	D	E
Distance	0	6	7	11	2
Distance From	A	A	A	B	B



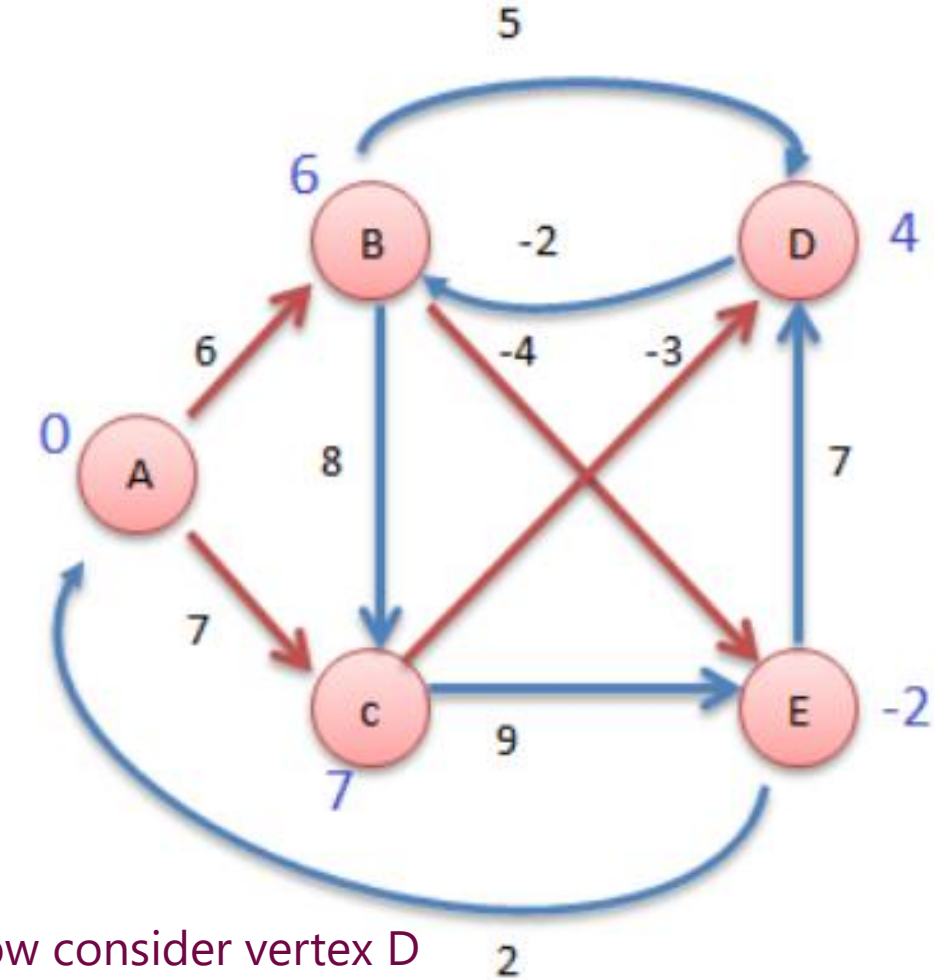
Now consider vertex E

No. of Nodes	A	B	C	D	E
Distance	0	6	7	9	2
Distance From	A	A	A	E	B



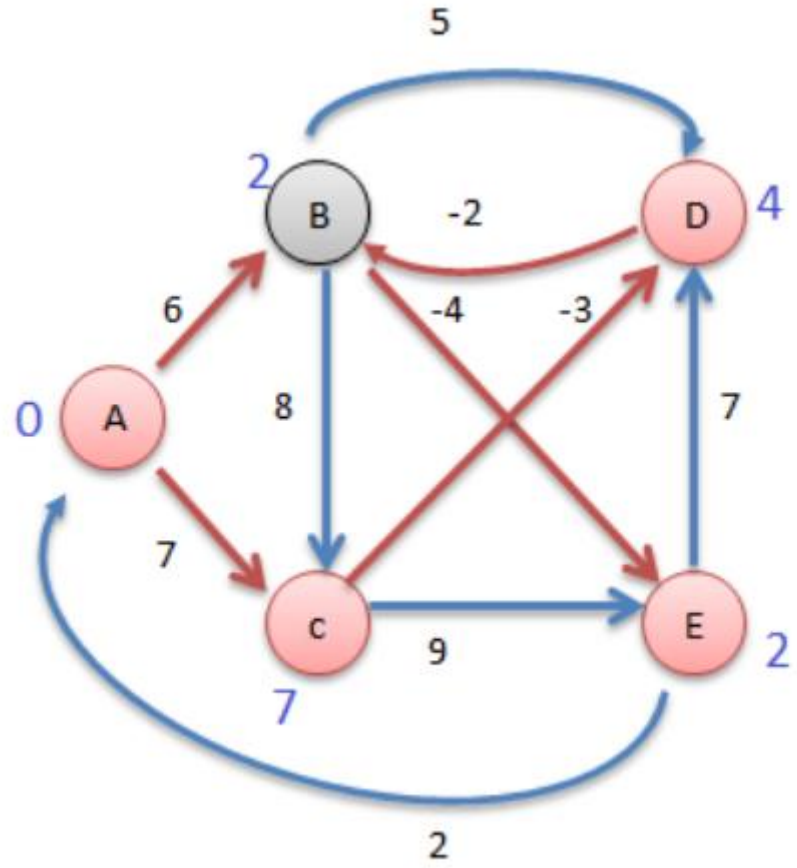
Now consider vertex C

No. of Nodes	A	B	C	D	E
Distance	0	6	7	4	2
Distance From	A	A	A	C	B



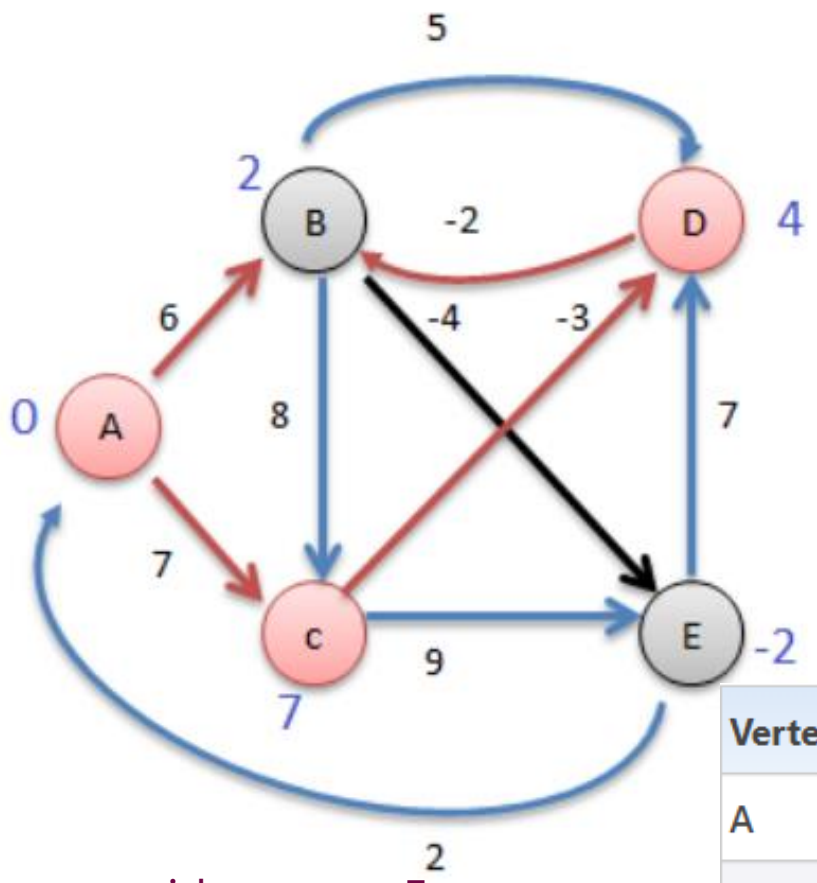
Now consider vertex D

No. of Nodes	A	B	C	D	E
Distance	0	2	7	4	2
Distance From	A	D	A	C	B



Now consider vertex B

No. of Nodes	A	B	C	D	E
Distance	0	2	7	4	-2
Distance From	A	D	A	C	B



Now consider vertex E

No. of Nodes	A	B	C	D	E
Distance	0	2	7	4	-2
Distance From	A	D	A	C	B

Vertex	Distance from A
A	0
B	2
C	7
D	4
E	-2

APPLICATION

DJISTRA'S ALGORITHIM

GIS (Geographic Information System), which needs to determine the shortest path from point A to point B on a road map (a weighted graph actually), can make a good use of Dijkstra's shortest path algorithm

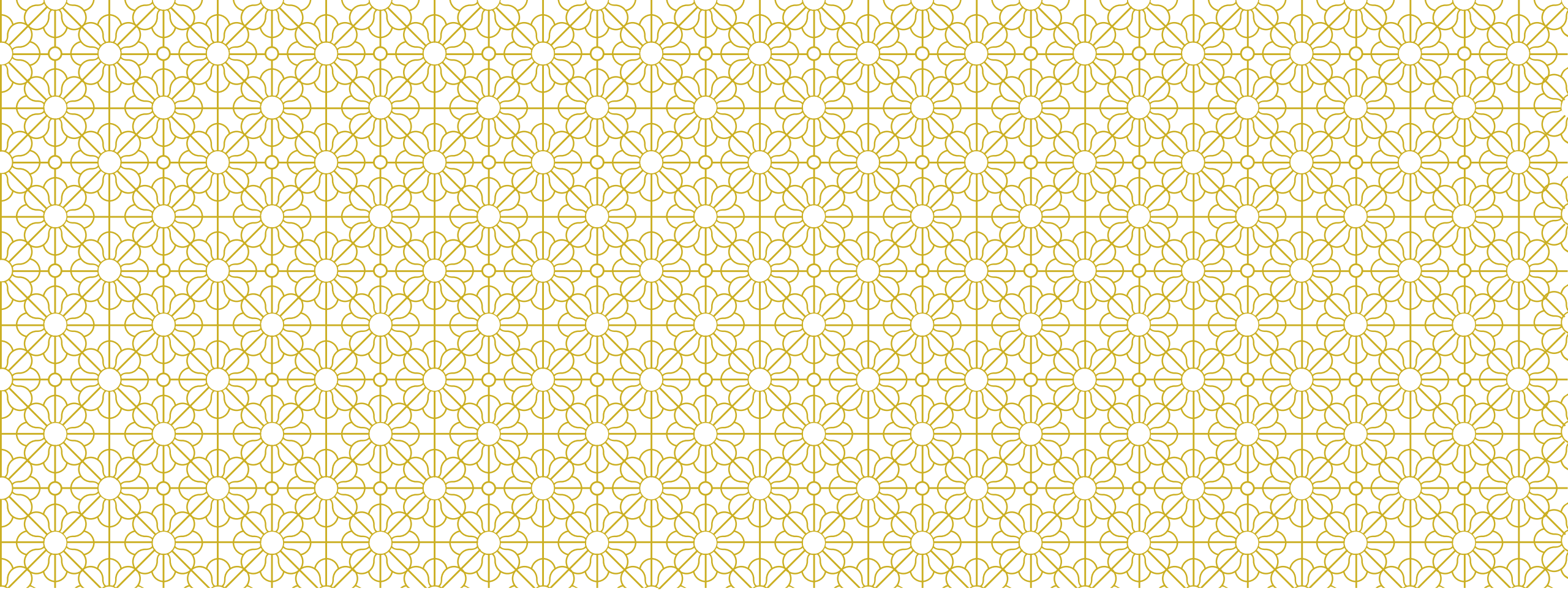
BELLMAN-FORD ALGORITHIM

Distance-vector routing protocol

- – Repeatedly relax edges until convergence
- – Relaxation is local!

On the Internet:

- – Routing Information Protocol (RIP)
- – Interior Gateway Routing Protocol (IGRP)



THANK YOU