**Assignment 2**

**Name:** Sahil Kaundal **UID:** 21BCS8197

**Sem.:** 4th **Section/Group:** 807/B

**Subject:** DSA **Subject Code:** 20CSY-251

**Q1.**

**Illustrate working of the following with an example:**

**i) Floyd Warshall Algorithm**

**ii) Bellman Ford Algorithm**

**Answer.**

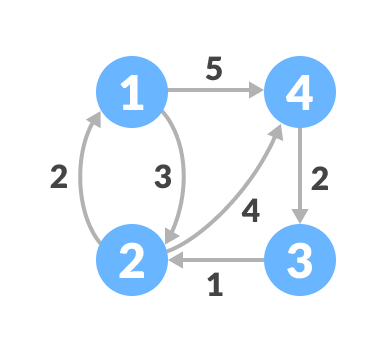
**i)Floyd-warshall:**

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

**Floyd-Warshall Algorithm:**

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

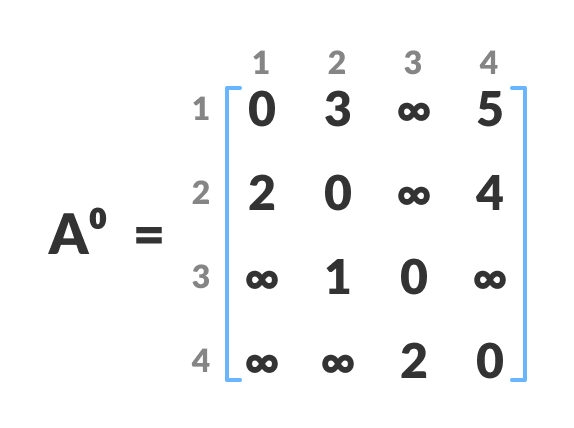
Let the given graph be:

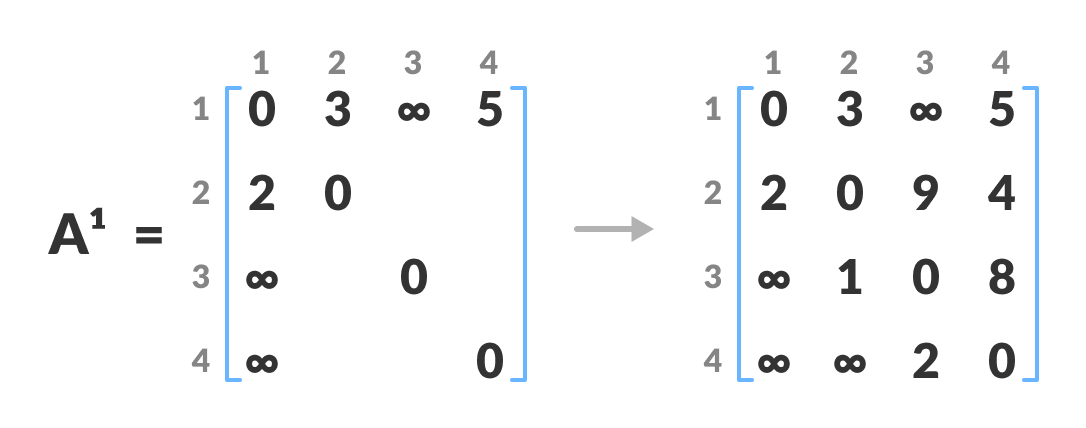
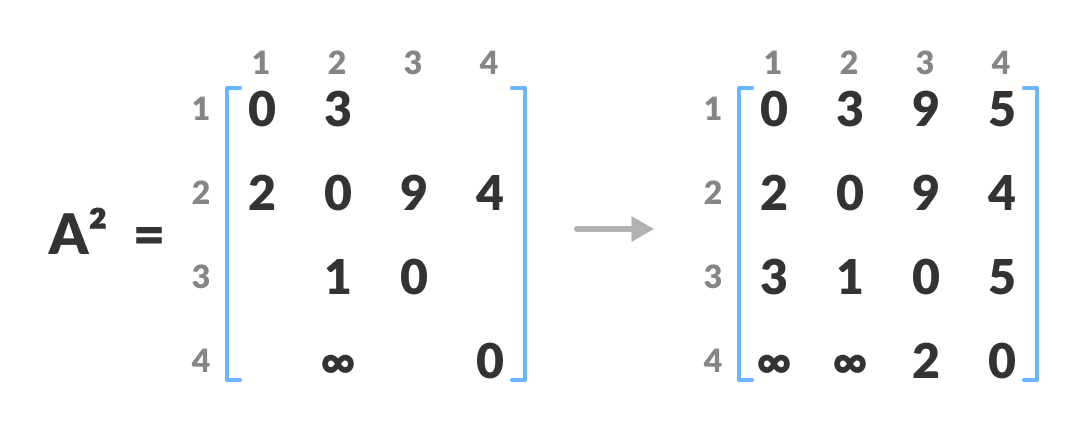


Initial graph

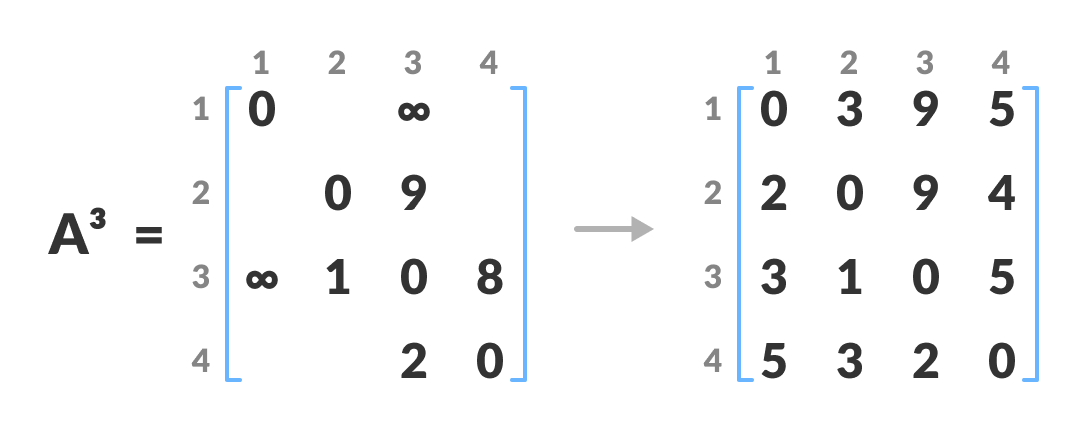
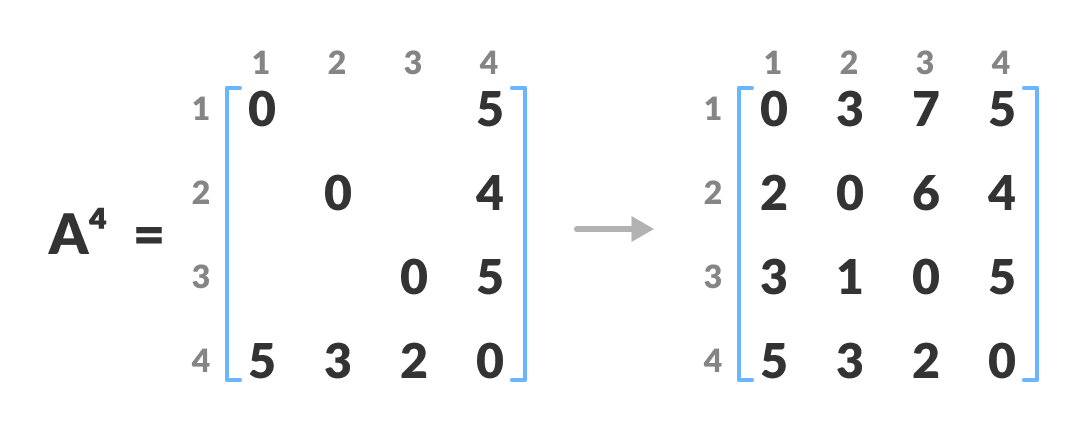
Follow the steps below to find the shortest path between all the pairs of vertices.

1. Create a matrix A0 of dimension n\*n where n is the number of vertices.
2. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.
3. Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.



1. Fill each cell with the distance between ith and jth vertex
2. Now, create a matrix A1 using matrix A0. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.
3. Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).
4. That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j].
5. In this step, k is vertex 1. We calculate the distance from source vertex to destination vertex through this vertex k.
6. Calculate the distance from the source vertex to destination vertex through this vertex k
7. For example: For A1[2, 4], the direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (ie. from vertex 2 to 1 and from vertex 1 to 4) is 7. Since 4 < 7, A0[2, 4] is filled with 4.
8. Similarly, A2 is created using A1. The elements in the second column and the second row are left as they are.
9. In this step, k is the second vertex (i.e. vertex 2). The remaining steps are the same as in **step 2**.
10. 

Calculate the distance from the source vertex to destination vertex through this vertex 2

1. Similarly, A3 and A4 is also created.
2. Calculate the distance from the source vertex to destination vertex through this vertex 3
3. Calculate the distance from the source vertex to destination vertex through this vertex 4
4. A4 gives the shortest path between each pair of vertices.

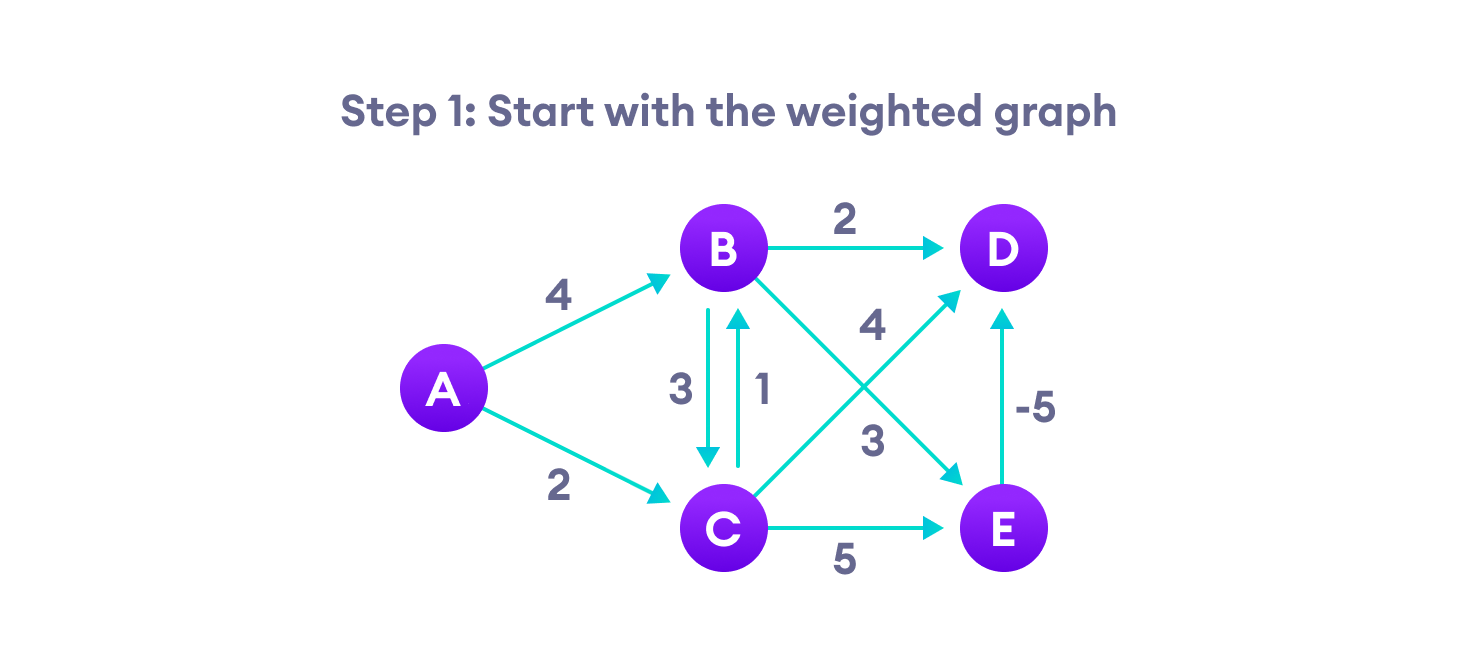
**ii)Bellman-Ford:**

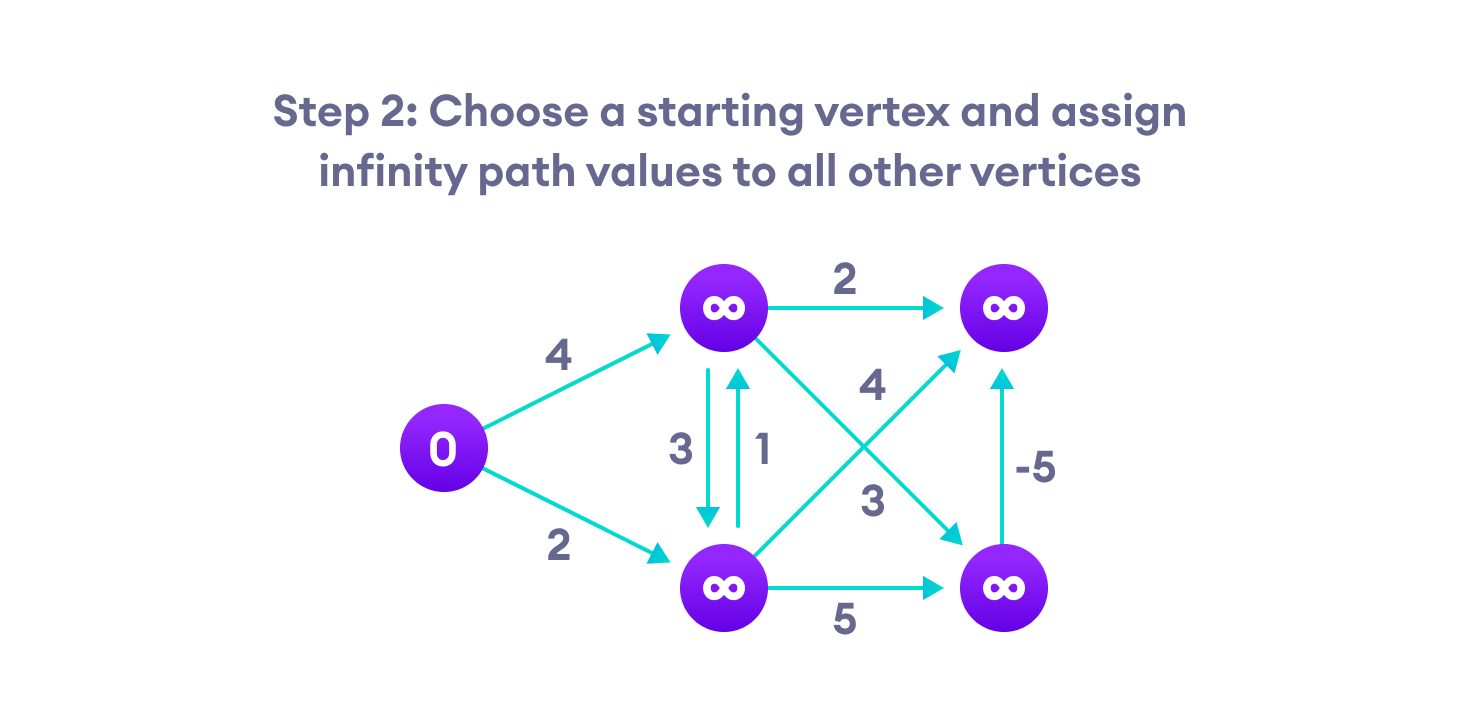
The bellman-Ford algorithm helps us find the shortest path from a vertex to all other vertices of a weighted graph.

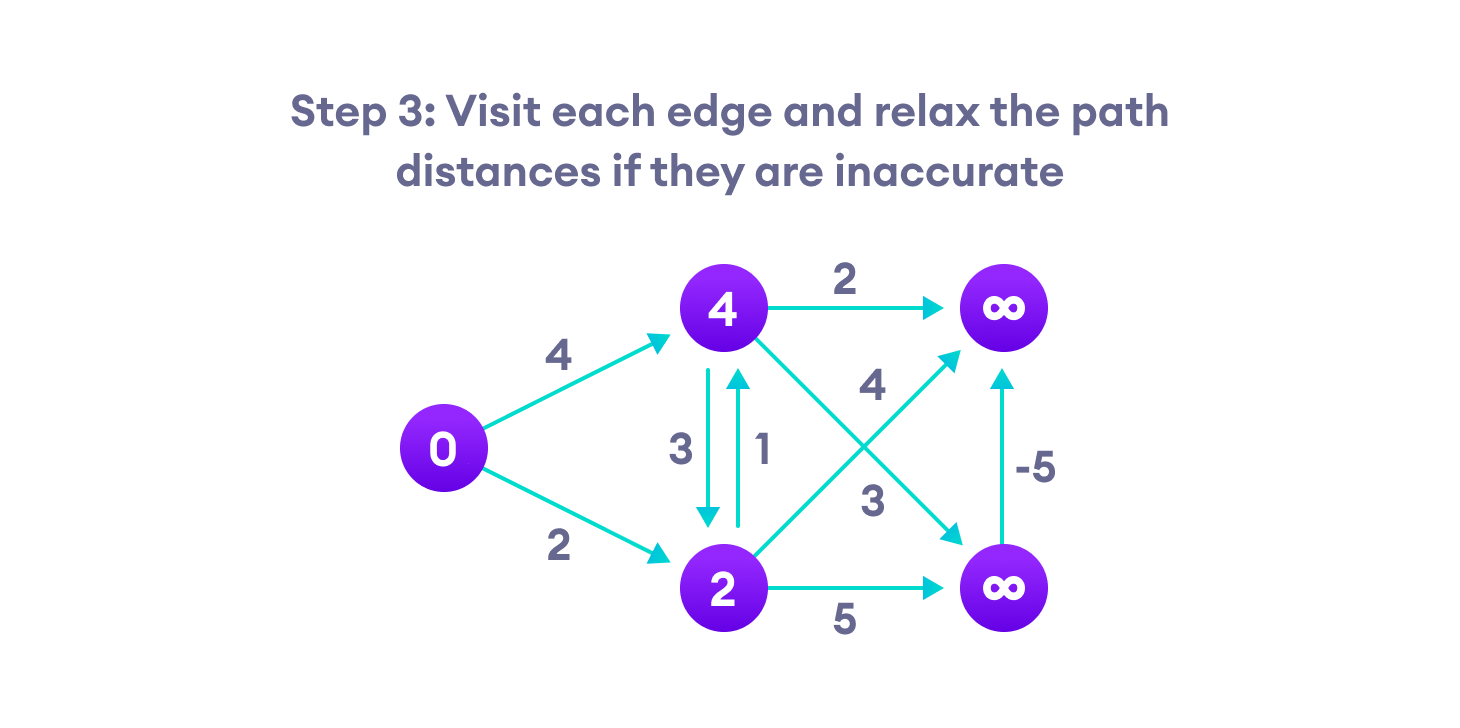
Bellman-Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices. Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths.

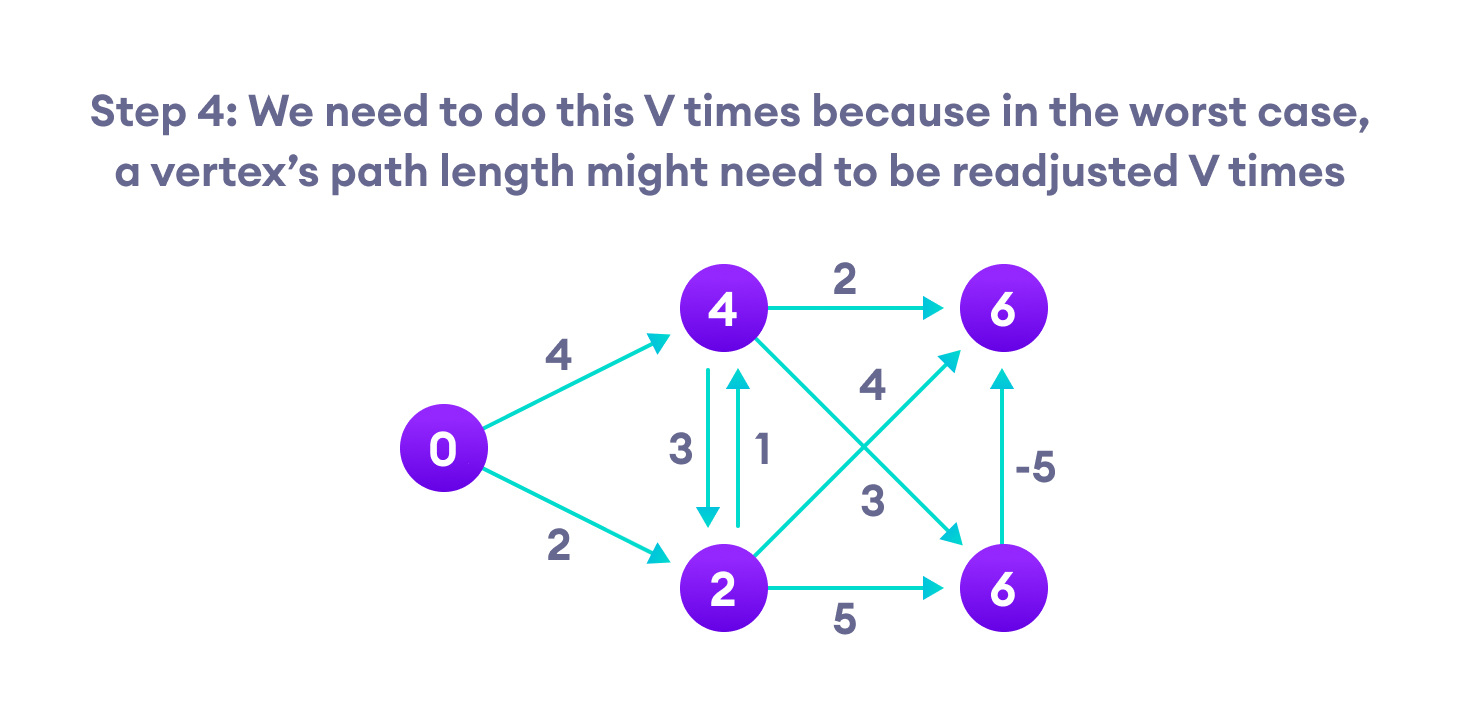
Bellman ford algorithm is a single-source shortest path algorithm. This algorithm is used to find the shortest distance from the single vertex to all the other vertices of a weighted graph. There are various other algorithms used to find the shortest path like Dijkstra algorithm, etc. If the weighted graph contains the negative weight values, then the Dijkstra algorithm does not confirm whether it produces the correct answer or not. In contrast to Dijkstra algorithm, bellman ford algorithm guarantees the correct answer even if the weighted graph contains the negative weight values.

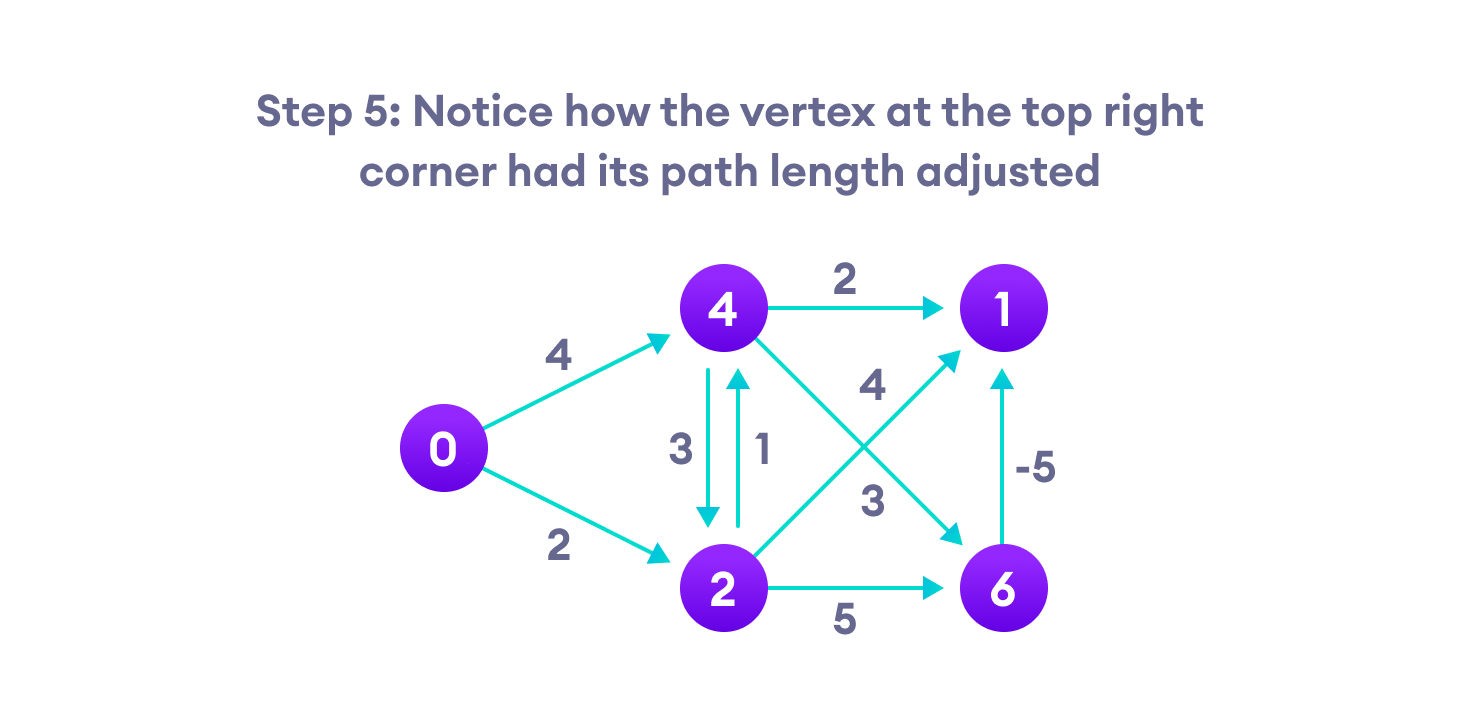
By doing this repeatedly for all vertices, we can guarantee that the result is optimized.

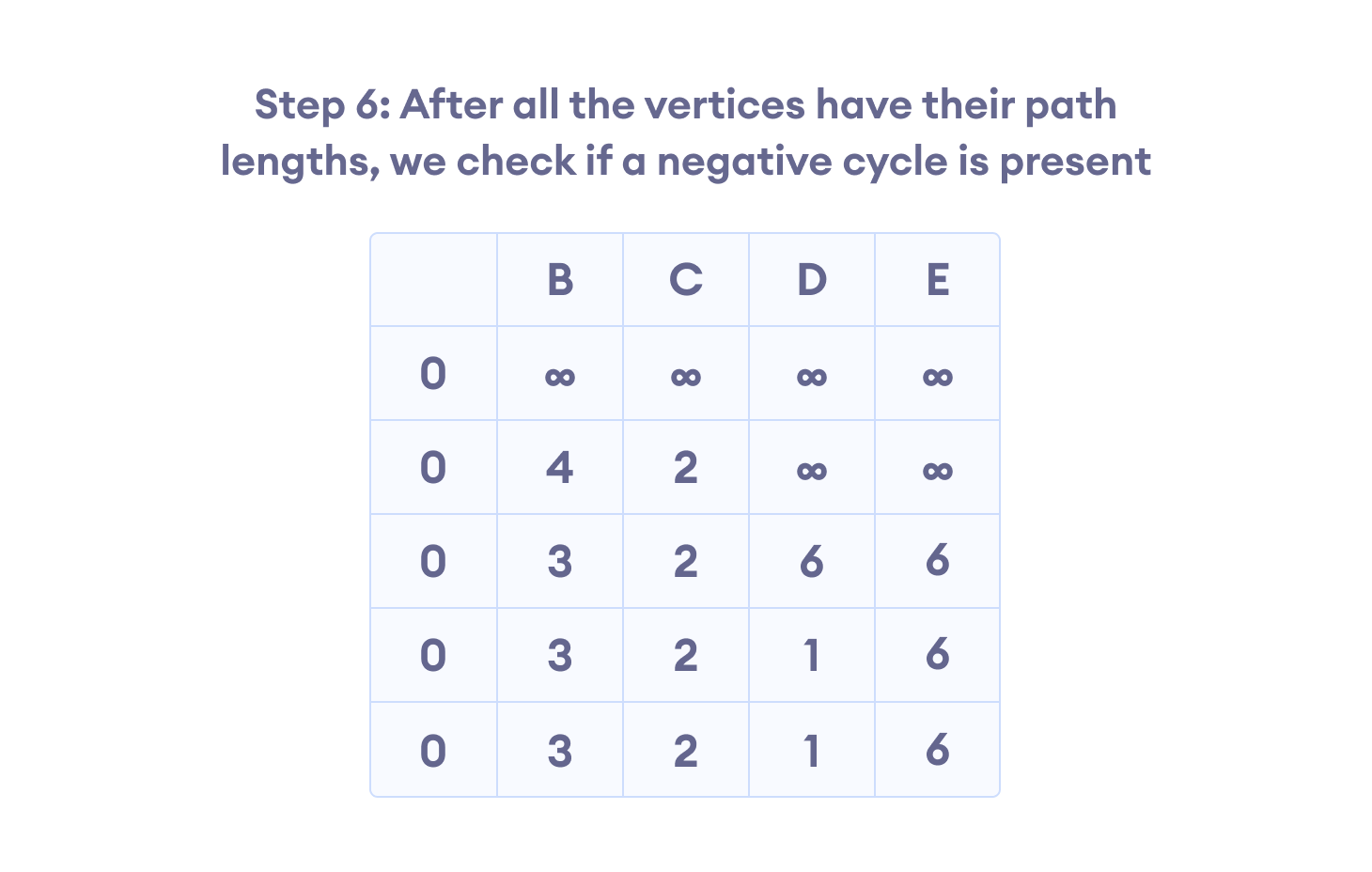
Step-1 for Bellman Ford's algorithm

Step-2 for Bellman Ford's algorithm

Step-3 for Bellman Ford's algorithm

Step-4 for Bellman Ford's algorithm

Step-5 for Bellman Ford's algorithm

Step-6 for Bellman Ford's algorithm

**Q2.**

**Compare the contrast between a binary and binary search tree.**

**Answer.**

|  |  |  |
| --- | --- | --- |
| **Basis for comparison** | **Binary tree** | **Binary search tree** |
| **Definition** | A binary tree is a non-linear data structure in which a node can have utmost two children, i.e., a node can have 0, 1 or maximum two children. A binary search tree is an ordered binary tree in which some order is followed to organize the nodes in a tree. | A [Binary search tree](https://www.javatpoint.com/binary-search-tree) is a tree that follows some order to arrange the elements, whereas the binary tree does not follow any order. In a Binary search tree, the value of the left node must be smaller than the parent node, and the value of the right node must be greater than the parent node. |
| **Structure** | The structure of the binary tree is that the first node or the topmost node is known as the root node. Each node in a binary tree contains the left pointer and the right pointer. The left pointer contains the address of the left subtree, whereas right pointer contains the address of right subtree. | The binary search tree is one of the types of binary tree that has the value of all the nodes in the left subtree lesser or equal to the root node, and the value of all the nodes in a right subtree are greater than or equal to the value of the root node. |
| **Operations** | The operations that can be implemented on a binary tree are insertion, deletion, and traversal. | Binary search trees are the sorted binary trees that provide fast insertion, deletion and search. Lookups mainly implement binary search as all the keys are arranged in sorted order. |
| **types** | Four types of binary trees are Full Binary Tree, Complete Binary Tree, Perfect Binary Tree, and Extended Binary Tree. | There are different types of binary search trees such as AVL trees, Splay tree, Tango trees, etc. |

**Binary tree:**

* There is no relative order to how the nodes should be organized.
* It’s basically a hierarchical data structure that is a collection of elements called nodes.
* It is used for fast and efficient lookup of data and information in a tree structure.

[**Binary search tree**](https://www.javatpoint.com/binary-search-tree)**:**

* It follows a definitive order to how the nodes should be organized in a tree.
* It’s a variant of the binary tree in which the nodes are arranged in a relative order.77
* It is mainly used for insertion, deletion, and searching of elements.