InClass Assignment

21AIE203

Data Structure and Algorithms – SEM-III

Professor – Dr. Sachin Sir

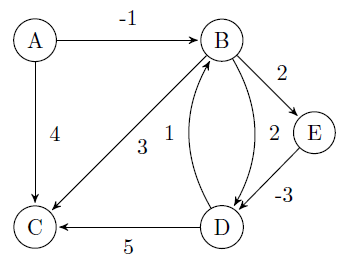
Submitted - Vikhyat Bansal – [CB.EN.U4AIE21076]



Bellman-Ford Algorithm

The Bellman-Ford algorithm is an example of Dynamic Programming.  
It is an algorithm used to get the shortest path where the graph is generally directed and contains positive or negative weight.

For example: Look at the graph below

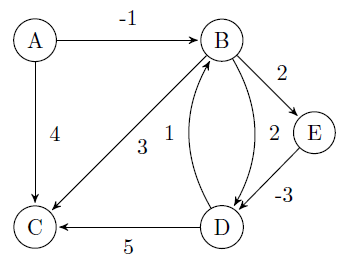


Bellman ford algorithm is used to calculate the shortest paths from a single source vertex to all vertices in the graph. This algorithm also works on graphs with a negative edge weight cycle (It is a cycle of edges with weights that sums to a negative number), unlike Dijkstra which gives wrong answers for the shortest path between two vertices.

Working

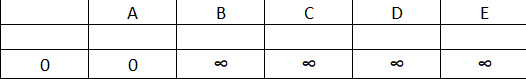
* We can take any node as our source. In this case, it is node A.
* We are going to do some iteration(traversing) using loops in such a way that for iteration-1 we will go to a node using path where there is only one edge and moving on for iteration – 2 we will go to a node using where there are two edges being used.
* We will keep track of the distance in the array whose size is equal to number of vertices and iterations are going to be V-1 because while doing iterations the array can go index out of bounds and display an error if we take more than V-1 iterations.
* Remember initial distance of each vertex in the array should be some infinite value because initial distance between two nodes are unknown.
* If condition can be used to replace each array element with the shortest weight.
* Now, we can do our iteration and store our final result in an array.
* Here, in our explanation we are not displaying the path where the weight is more than the last iteration.

Solution for given graph



**For iteration -1**

**Array – [0 ∞ ∞**  **∞**  **∞]**



From Node A 🡪 B

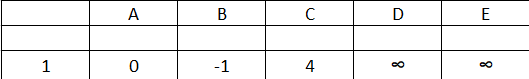
Weight = -1

From Node A🡪 C

Weight = 4

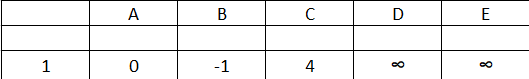
It will stop iteration here because no other node can be accessed using only one edge.

**Array – [0 -1 4 ∞**  **∞]**

****

**For iteration -2**

**Array – [0 -1 4 ∞ ∞]**

****

From Node A 🡪 B

Weight = -1

From Node A🡪 C

Path – A🡪B🡪C

Weight = -1 + 3 = 2 which is less than last iteration calculated distance.

From Node A🡪D

Path – A🡪B🡪D

Weight = -1 + 2 = +1 which is less than infinity.

From Node A🡪E

Path – A🡪B🡪E

Weight = 1 + 2 = +1 which is less than infinity.

**Array – [0 -1 2 1 1]**

****

It will stop iteration here because no other node can be accessed using only two edge.

**For iteration -3**

**Array – [0 -1 2 1 1]**

****

From Node A 🡪 B

Weight = -1

From Node A🡪 C

Path – A🡪B🡪C

Weight = -1 + 3 = 2 which is less than last iteration calculated distance.

From Node A🡪D

Path – A🡪B🡪E🡪D

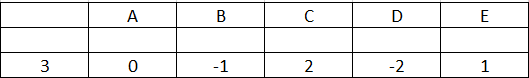
Weight = -1 + 2 - 3 = -2 which is less than 1 calculated in iteration - 2.

From Node A🡪E

Path – A🡪B🡪E

Weight = 1 + 2 = +1 which is less than infinity.

**Array – [0 -1 2 -2 1]**

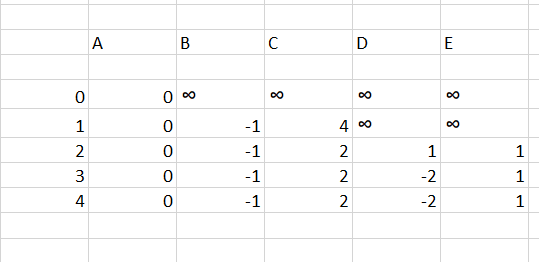
****

It will stop iteration here because no other node can be accessed using only three edge.

**For iteration -4**

It will look for all the possible paths using 4 edges for covering source vertex to destination vertex.

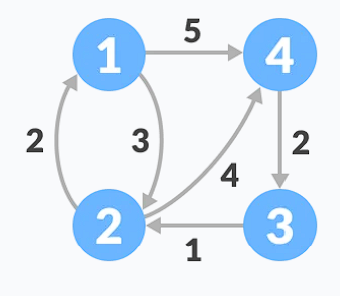
Final Array Table



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).

For example: Look at the graph below.

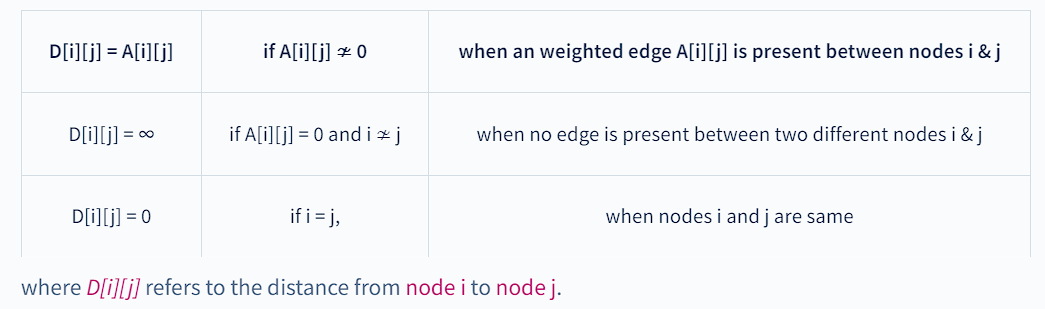


In other words, the Floyd-Warshall algorithm is an ideal choice for finding the length of the shortest path across every pair of nodes in a graph data structure.

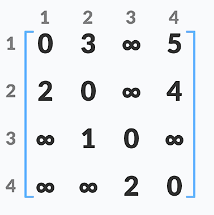
Floyd-Warshall is preferred but at the same time is used a lot less because of time complexity.

Working

* Create a corresponding distance matrix D having the same dimension as A which satisfies the following conditions:



* Matrix will something look like this



* Now, to solve this further what we have to do is create different matrix depending upon the number of vertex.
* We take some random variable k starting from 1 till n,where n Is the number of vertex.
* Depending upon the value of k, we will avoid doing changes in that particular row and column.

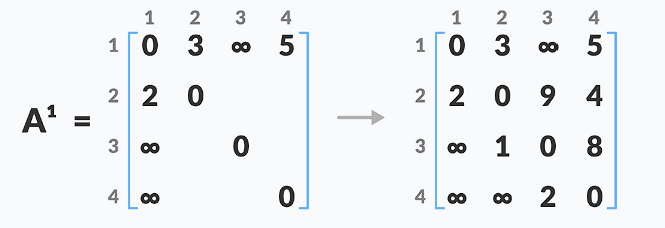
For ex: if k = 1, we will not do any change in row 1 and column 1 and keep the diagonals as it is because we do not have any loop in our graph.



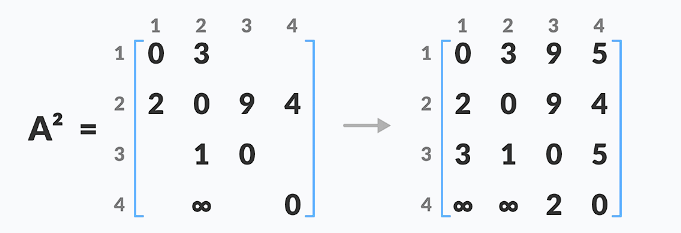
* When we have a value in k, we will try to replace infinity or some weight for one node to another node in such a way that kth node is coming in between.
* When for some value of k, we have gone through all elements of matrix we can increase the value of k by one and repeat the last two steps.
* At last value of k, we will have some matrix which is going to be our last matrix.

Solving it

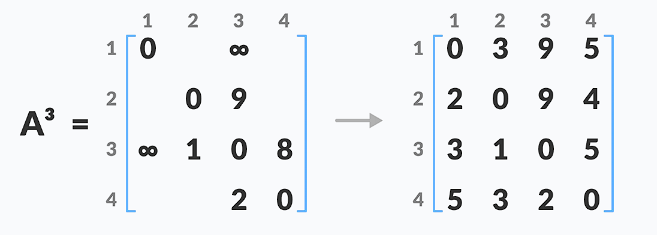
K = 1



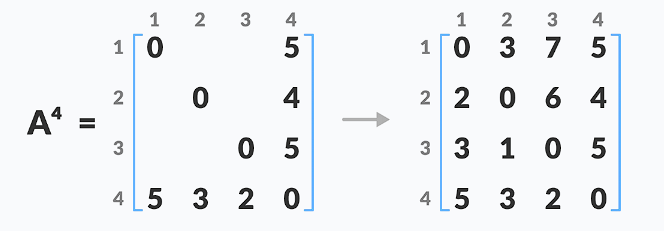
K = 2



K = 3



K = 4



We have our final matrix with shortest path with each pair of vertices.

THANK YOU!!