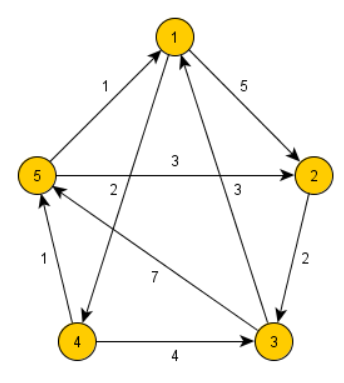
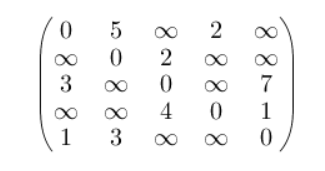
**Floyd-Warshall Algorithm**

Floyd-Warshall algorithm is a procedure, which is used to find the shortest (longest) paths among all pairs of nodes in a graph, which does not contain any cycles of negative length. The main advantage of Floyd-Warshall algorithm is its simplicity.

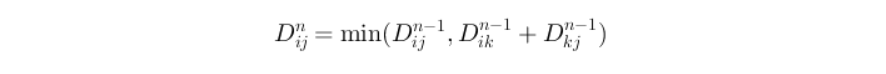


**Graph and matrix of lengths**

**Description:**

Floyd-Warshall algorithm uses a matrix of lengths D\_{0} as its input. If there is an edge between nodes i and j, than the matrix D\_{0} contains its length at the corresponding coordinates. The diagonal of the matrix contains only zeros. If there is no edge between edges i and j, than the position (i,j) contains positive infinity. In other words, the matrix represents lengths of all paths between nodes that does not contain any intermediate node.

In each iteration of Floyd-Warshall algorithm is this matrix recalculated, so it contains lengths of paths among all pairs of nodes using gradually enlarging set of intermediate nodes. The matrix D\_{1}, which is created by the first iteration of the procedure, contains paths among all nodes using exactly one (predefined) intermediate node. D {2} contains lengths using two predefined intermediate nodes. Finally the matrix D {n} uses n intermediate nodes.

This transformation can be described using the following recurrent formula:

Because this transformation never rewrites elements, which are to be used to calculate the new matrix, we can use the same matrix for both D^{i} and D^{i+1}.

The Floyd-Warshall algorithm is a shortest path algorithm for [graphs](https://brilliant.org/wiki/graphs/). Like the [Bellman-Ford algorithm](https://brilliant.org/wiki/bellman-ford-algorithm/) or the [Dijkstra's algorithm](https://brilliant.org/wiki/dijkstras-short-path-finder/), it computes the shortest path in a graph. However, Bellman-Ford and Dijkstra are both single-source, shortest-path algorithms. This means they only compute the shortest path from a single source. Floyd-Warshall, on the other hand, computes the shortest distances between every pair of vertices in the input graph.

**Function:**

At the heart of Floyd-Warshall is this function:

This function returns the shortest path from A to C using the vertices from 1 to K in the graph. The vertices are individually numbered 1, 2,...k

There is a base case and a [recursive](https://brilliant.org/wiki/recursion-problem-solving/) case. The base case is that the shortest path is simply the weight of the edge connecting A and C

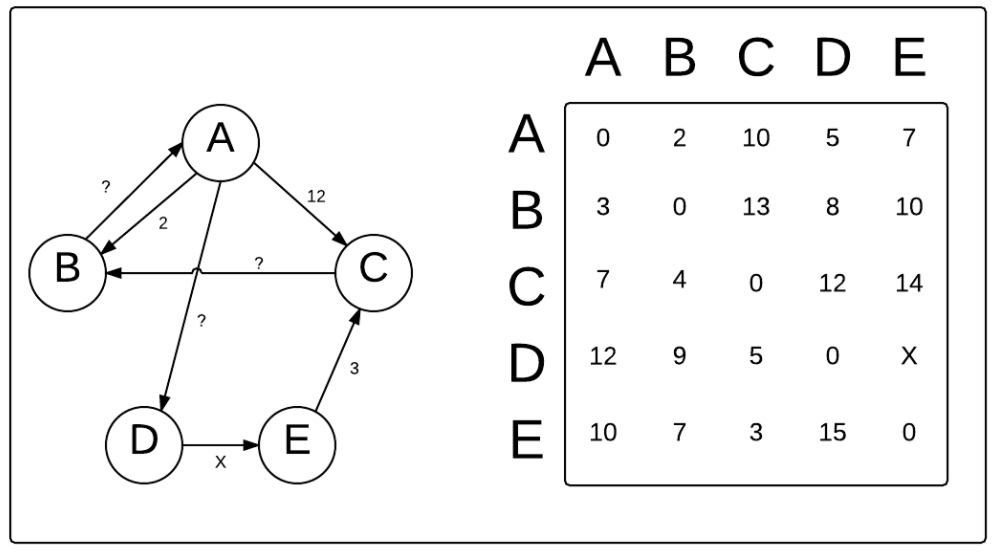
The recursive case will take advantage of the [dynamic programming](https://brilliant.org/wiki/problem-solving-dynamic-programming/) nature of this problem. There are two possible answers for this function. Either the shortest path between and  is the shortest known path, or it is the shortest known path from  to some vertex (let's call it ) plus the shortest known path from  Z to J.

Basically, what this function setup is asking this: "Is the vertex K an intermediate of our shortest path (any vertex in the path besides the first or the last)?"

If  is not an intermediate vertex, then the shortest path from  to  using the vertices in  is also the shortest path using the vertices in

If K is an intermediate vertex, then the path can be broken down into two paths, each of which uses the vertices in {1,2,…,k-1}  i to j make a path that uses all vertices in  {1,2,…,k}  That is because the vertex K  is the middle point. This is illustrated in the image below.

**Graph and distance Matrix:**



**The Floyd-Warshall algorithm can be described by the following pseudo code:**

Create a |V| x |V| matrix, M, that will describe the distances between vertices

For each cell (i, j) in M:

if i == j:

M[i][j] = 0

if (i, j) is an edge in E:

M[i][j] = weight(i, j)

else:

M[i][j] = infinity

for k from 1 to |V|:

for i from 1 to |V|:

for j from 1 to |V|:

if M[i][j] > M[i][k] + M[k][j]:

M[i][j] = M[i][k] + M[k][j]

