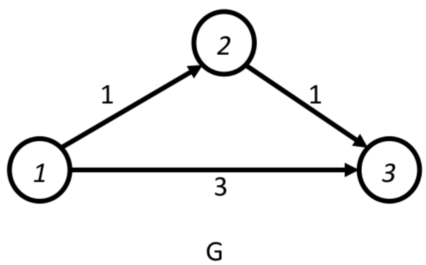
1. *All pairs shortest paths problem*: Given a weighted, directed graph $G = (V, E)$, you are supposed to design an algorithm that outputs an $|V| \times |V|$ matrix A such that $A[i, j]$ contains the length of the shortest path in G from vertex i to vertex j.
   (*For example, the shortest path matrix for the graph given below is on the right.*)



	1	2	3
1	0	1	2
2	∞	0	1
3	∞	∞	0

Figure 1: ∞ in a table entry $A[i, j]$ means that there is no path in the graph from vertex i to vertex j.

You can solve this problem using Dijkstra's algorithm (in case all edge weights are positive) repeatedly on the same graph and different starting vertices.

Question 1: What is the running time of the above algorithm?

We will design an algorithm with better running time using Dynamic Programming idea. For any i, j, k, let $D_k(i, j)$ denote the length of the shortest path from vertex i to vertex j when all the intermediate vertices in the path is from the set $\{1, ..., k\}$.

Given the above definition, we can say that that for all i, j, $D_0(i, j) =$ weight of edge (i, j) in case it exists, otherwise ∞.

Question 2: Write $D_1(., .)$ in terms of $D_0(., .)$.

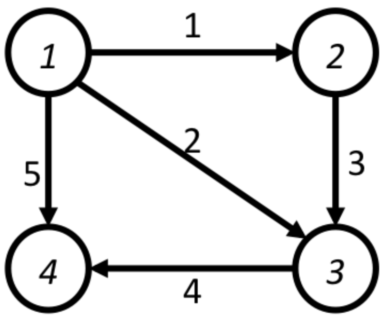
Question 3: Write $D_i(., .)$ in terms of $D_{i-1}(., .)$.

Note that for the output matrix A, $A[i, j] = D_n(i, j)$ for all i, j. So, all we need to do is to figure out a way to compute $D_n(i, j)$ for all i, j. As evident from the recursive formulation in the previous question, we should compute $D_i(., .)$ before computing $D_{i-1}(., .)$. So, the algorithm runs in n passes and in the i^{th} pass it computes $D_i(j, k)$ for all j, k.

Question 4: What is the running time of the above algorithm? Is this better than running Dijkstra's repeatedly?

Question 5: Does this algorithm also work for graphs that have negative weight edges but no negative weight cycles?

Question 6: Consider the graph given below and simulate this algorithm on this graph. That is, fill the tables $D_0(.,.), D_1(.,.), ..., D_4(.,.)$.



	1	2	3	4
1	0	1	2	5
2	∞	0	3	∞
3	∞	∞	0	4
4	∞	∞	∞	0

D_0

	1	2	3	4
1				
2				
3				
4				

D_1

	1	2	3	4
1				
2				
3				
4				

D_2

	1	2	3	4
1				
2				
3				
4				

D_3

	1	2	3	4
1				
2				
3				
4				

D_4

2. You are given a bipartite graph $G = (L, R, E)$ such that $|L| = |R| = n$ and $|E| = m$. You are also given a matching $M \subseteq E$ such that $|M| = (n - 1)$. Your goal is to design an algorithm that takes as input G and matching M and determines whether there exists a perfect matching in the graph G. That is, it should output "yes" if there is a perfect matching in G, otherwise it should output "no".

- There is a simple $O(n \cdot (n+m))$ algorithm for this problem that ignores the matching M. Discuss this algorithm and its running time.
- Design an $O(n + m)$ algorithm and discuss correctness and running time.