All Pairs Shortest Path:

Reading:

- "The Floyd-Warshall algorithm" 25.2 CLRS
- "All Pairs Shortest-Paths" 7.2 GT

When we looked at greedy algorithms, we saw Dijkstra's algorithm, which found all shortest paths from a single source vertex in $O\left([|V| + |E|] \log |V|\right)$ time. To find the shortest path between every pair of vertices, we could run Dijkstra's algorithm once from every vertex. If the graph is dense (i.e. $|E| \in \Theta\left(|V|^2\right)$), then the runtime of this approach is

$$|V| \times O\left(\left[|V| + |V|^2\right] \log |V|\right) = O\left(|V|^3 \log |V|\right)$$

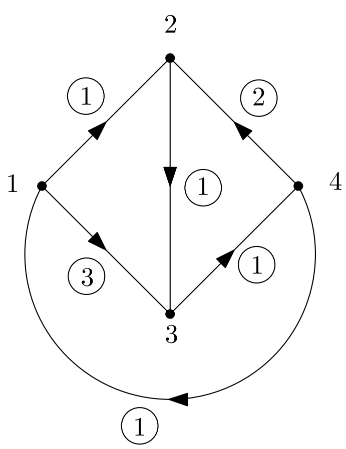
We now look at a clever dynamic programming solution that solves this problem for the general case where there are negative edge weights, but no negative cost cycles. Its beauty is its simplicity: The algorithm is one "if" statement nested within three "for" loops. Plus, it has a runtime of just $O\left(|V|^3\right)$.

Preliminaries:

To simplify our presentation, we assume that the vertices are labelled from $1 \ldots n$. Any order of the vertices will do. We also assume that the graph is represented with a $n \times n$ adjacency matrix M such that

$$M[i,j] = \begin{cases} \infty & \text{if there is no edge from } i \text{ to } j \\ \text{weight of the edge from } i \text{ to } j & \text{otherwise} \end{cases}$$

Example:

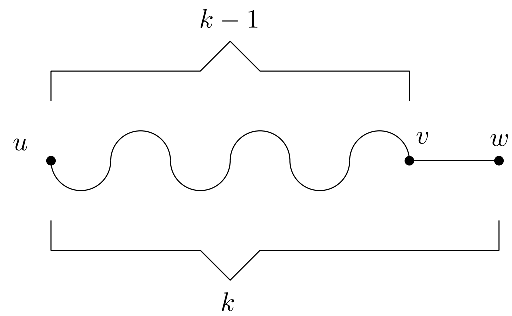


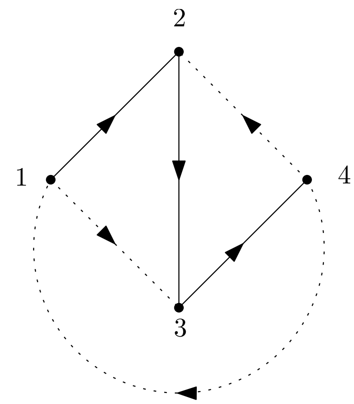
M

j

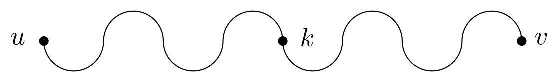
		1	2	3	4
	1	∞	1	3	
i	2	∞	∞	1	
	3	∞	∞	∞	
	4	1	2	∞	

The key to creating a dynamic programming algorithm is observing some optimal substructure. Recall the Bellman-Ford algorithm: The shortest u, w-path using k edges is a shortest path using $k-1$ edges plus one other edge.

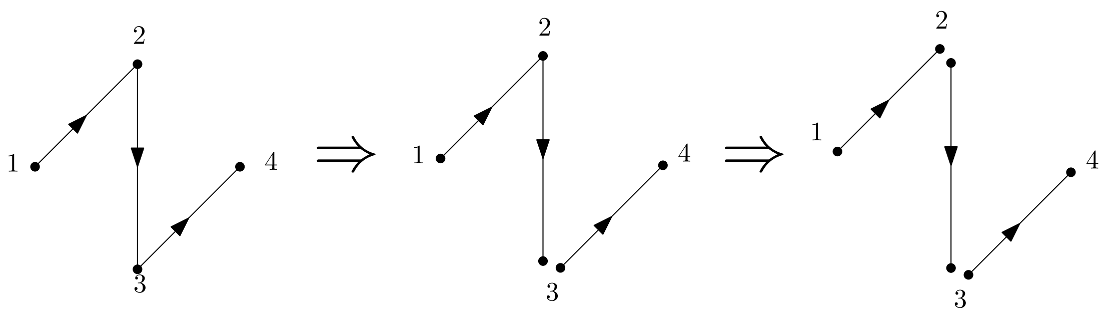




Consider the $1,4$-path $\langle 1,2,3,4\rangle$ above. We call the vertices of the path other than the endpoints *internal vertices* (vertices 2 and 3 are internal). Without negative cost cycles, there is a shortest u,v-path where every vertex occurs at most once (i.e. we can prune cycles because they do not make the path shorter). Using this observation, we split such a path at its greatest internal vertex k.



The maximum internal node on the u,k-subpath is less than k. Similarly for the k,v-subpath. This breaks the path into "smaller" subpaths.



Let $D_k\left[i,j\right]$ be the length of the shortest i,j-path with internal vertices in the range of $1\ldots k$. Then

$$D_k\left[i,j\right] = \max\left\{\begin{array}{c} D_{k-1}\left[i,j\right], \\ D_{k-1}\left[i,k\right] + D_{k-1}\left[k,j\right] \end{array}\right\}$$

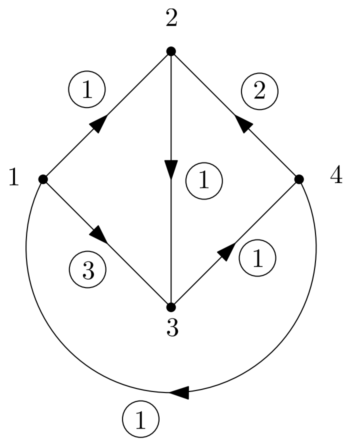
because the shortest i,j-path with internal vertices from $1\ldots k$ either does not go through k (first case) or it does not (second case). Our base case is $D_0 = M$.

Notice that D_k only depends on D_{k-1}, so we can save space by using one matrix for all D_i.

```
Algorithm Floyd-Warshall(M,n)
  D ← M

  for k ← 1 to n do
    for i ← 1 to n do
      for j ← 1 to n do
        D[i,j] ← max { D[i,j],
                       D[i,k] + D[k,j] }
  return D
```

Example:



D_0

j

		1	2	3	4
	1	∞	1	3	∞
i	2	∞	∞	1	∞
	3	∞	∞	∞	1
	4	1	2	∞	∞

D_1

j

		1	2	3	4
	1	∞	1	3	∞
i	2	∞	∞	1	∞
	3	∞	∞	∞	1
	4	1			∞

D_2

j

		1	2	3	4
	1	∞	1		∞
i	2	∞	∞	1	∞
	3	∞	∞	∞	1
	4	1	2		∞

D_3

j

		1	2	3	4
	1	∞	1	2	
i	2	∞	∞	1	
	3	∞	∞	∞	1
	4	1	2	3	∞

D_4

j

		1	2	3	4
	1		1	2	3
i	2			1	2
	3				1
	4	1	2	3	∞