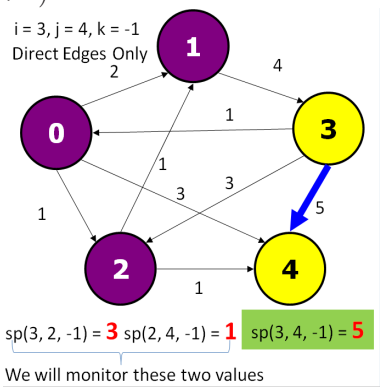
4.5.2 Explanation of Floyd Warshall's DP Solution

We provide this section for the benefit of readers who are interested to know why Floyd Warshall's works. This section can be skipped if you just want to use this algorithm per se. However, examining this section can further strengthen your DP skill. Note that there are graph problems that have no classical algorithm yet and must be solved with DP techniques (see Section 4.7.1).

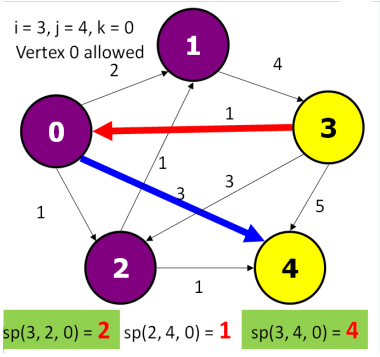


The current content of Adjacency Matrix D at **k = -1**

k = -1	0	1	2	3	4
0	0	2	1	∞	3
1	∞	0	∞	4	∞
2	∞	1	0	∞	1
3	1	∞	3	0	**5**
4	∞	∞	∞	∞	0

Figure 4.20: Floyd Warshall's Explanation 1

The basic idea behind Floyd Warshall's is to gradually allow the usage of intermediate vertices (vertex `[0..`k`]`) to form the shortest paths. We denote the shortest path from vertex i to vertex j using only intermediate vertices `[0..`k`]` as `sp(i,j,k)`. Let the vertices be labeled from 0 to V-1. We start with direct edges only when $k = -1$, i.e. `sp(i,j,-1)` = weight of edge (i, j). Then, we find shortest paths between any two vertices with the help of restricted intermediate vertices from vertex `[0..`k`]`. In Figure 4.20, we want to find `sp(3,4,4)`—the shortest path from vertex 3 to vertex 4, using any intermediate vertex in the graph (vertex `[0..4]`). The eventual shortest path is path 3-0-2-4 with cost 3. But how to reach this solution? We know that by using only direct edges, `sp(3,4,-1) = 5`, as shown in Figure 4.20. The solution for `sp(3,4,4)` will *eventually* be reached from `sp(3,2,2)+sp(2,4,2)`. But with using only direct edges, `sp(3,2,-1)+sp(2,4,-1) = 3+1 = 4` is still > 3.

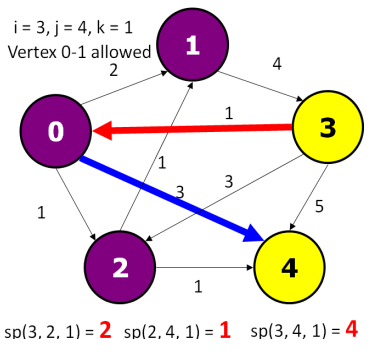


The current content of Adjacency Matrix D at **k = 0**

k = 0	0	1	2	3	4
0	0	2	1	∞	3
1	∞	0	∞	4	∞
2	∞	1	0	∞	1
3	1	**3**	**2**	0	**4**
4	∞	∞	∞	∞	0

Figure 4.21: Floyd Warshall's Explanation 2

Floyd Warshall's then gradually allow $k = 0$, then $k = 1$, $k = 2 \ldots$, up to $k = V$-1. When we allow $k = 0$, i.e. vertex 0 can now be used as an intermediate vertex, then `sp(3,4,0)` is reduced as `sp(3,4,0) = sp(3,0,-1) + sp(0,4,-1) = 1+3 = 4`, as shown in Figure 4.21. Note that with $k = 0$, `sp(3,2,0)`—which we will need later—also drop from 3 to `sp(3,0,-1) + sp(0,2,-1) = 1+1 = 2`. Floyd Warshall's will process `sp(i,j,0)` for all other pairs considering only vertex 0 as the intermediate vertex but there is only one more change: `sp(3,1,0)` from ∞ down to 3.

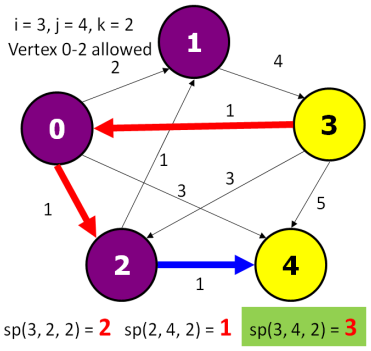


The current content of Adjacency Matrix D at **k = 1**

k = 1	0	1	2	3	4
0	0	2	1	6	3
1	∞	0	∞	4	∞
2	∞	1	0	5	1
3	1	3	2	0	4
4	∞	∞	∞	∞	0

Figure 4.22: Floyd Warshall's Explanation 3

When we allow $k = 1$, i.e. vertex 0 and 1 can now be used as intermediate vertices, then it happens that there is no change to `sp(3,2,1)`, `sp(2,4,1)`, nor to `sp(3,4,1)`. However, two other values change: `sp(0,3,1)` and `sp(2,3,1)` as shown in Figure 4.22 but these two values will not affect the final computation of the shortest path between vertex 3 and 4.



The current content of Adjacency Matrix D at **k = 2**

k = 2	0	1	2	3	4
0	0	2	1	6	2
1	∞	0	∞	4	∞
2	∞	1	0	5	1
3	1	3	2	0	3
4	∞	∞	∞	∞	0

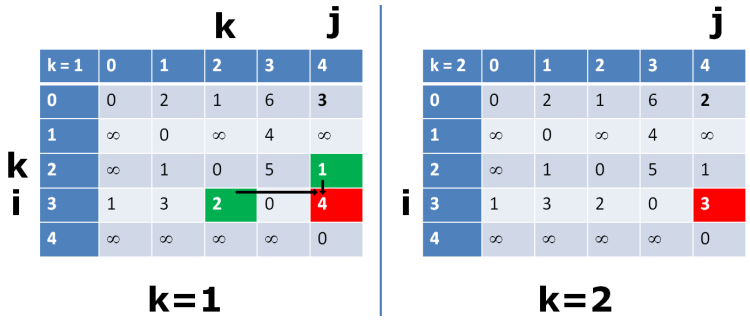
Figure 4.23: Floyd Warshall's Explanation 4

When we allow $k = 2$, i.e. vertex 0, 1, and 2 now can be used as the intermediate vertices, then `sp(3,4,2)` is reduced again as `sp(3,4,2) = sp(3,2,2)+sp(2,4,2) = 2+1 = 3` as shown in Figure 4.23. Floyd Warshall's repeats this process for $k = 3$ and finally $k = 4$ but `sp(3,4,4)` remains at 3 and this is the final answer.

Formally, we define Floyd Warshall's DP recurrences as follow. Let $D^k_{i,j}$ be the shortest distance from i to j with only `[0..k]` as intermediate vertices. Then, Floyd Warshall's base case and recurrence are as follow:

$D^{-1}_{i,j} = weight(i, j)$. This is the base case when we do not use any intermediate vertices.
$D^k_{i,j} = min(D^{k-1}_{i,j}, D^{k-1}_{i,k} + D^{k-1}_{k,j}) = min(\text{not using vertex } k, \text{using vertex } k)$, for $k \geq 0$.

This DP formulation must be filled layer by layer (by increasing k). To fill out an entry in the table k, we make use of the entries in the table k-1. For example, to calculate $D^2_{3,4}$, (row 3, column 4, in table $k = 2$, index start from 0), we look at the minimum of $D^1_{3,4}$ or the sum of $D^1_{3,2} + D^1_{2,4}$ (see Table 4.3). The naïve implementation is to use a 3-dimensional matrix `D[k][i][j]` of size $O(V^3)$. However, since to compute layer k we only need to know the values from layer k-1, we can drop dimension k and compute `D[i][j]` 'on-the-fly' (the space saving trick discussed in Section 3.5.1). Thus, Floyd Warshall's algorithm just need $O(V^2)$ space although it still runs in $O(V^3)$.



k = 1	0	1	2	3	4
0	0	2	1	6	3
1	∞	0	∞	4	∞
2	∞	1	0	5	1
3	1	3	2	0	4
4	∞	∞	∞	∞	0

k = 2	0	1	2	3	4
0	0	2	1	6	2
1	∞	0	∞	4	∞
2	∞	1	0	5	1
3	1	3	2	0	3
4	∞	∞	∞	∞	0

Table 4.3: Floyd Warshall's DP Table

4.5.3 Other Applications

The main purpose of Floyd Warshall's is to solve the APSP problem. However, Floyd Warshall's is frequently used in other problems too, as long as the input graph is small. Here we list down several problem variants that are also solvable with Floyd Warshall's.

Solving the SSSP Problem on a Small Weighted Graph

If we have the All-Pairs Shortest Paths (APSP) information, we also know the Single-Source Shortest Paths (SSSP) information from any possible source. If the given weighted graph is small $V \leq 400$, it may be beneficial, in terms of coding time, to use the four-liner Floyd Warshall's code rather than the longer Dijkstra's algorithm.

Printing the Shortest Paths

A common issue encountered by programmers who use the four-liner Floyd Warshall's without understanding how it works is when they are asked to print the shortest paths too. In BFS/Dijkstra's/Bellman Ford's algorithms, we just need to remember the shortest paths spanning tree by using a 1D `vi p` to store the parent information for each vertex. In Floyd Warshall's, we need to store a 2D parent matrix. The modified code is shown below.

```
// inside int main()
  // let p be a 2D parent matrix, where p[i][j] is the last vertex before j
  // on a shortest path from i to j, i.e. i -> ... -> p[i][j] -> j
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      p[i][j] = i;                          // initialize the parent matrix
  for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++) // this time, we need to use if statement
        if (AdjMat[i][k] + AdjMat[k][j] < AdjMat[i][j]) {
          AdjMat[i][j] = AdjMat[i][k] + AdjMat[k][j];
          p[i][j] = p[k][j];                    // update the parent matrix
        }
//---------------------------------------------------------------------------
// when we need to print the shortest paths, we can call the method below:
void printPath(int i, int j) {
  if (i != j) printPath(i, p[i][j]);
  printf(" %d", j);
}
```