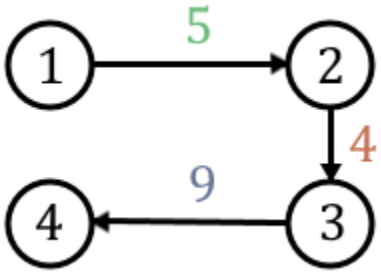
CIS 360 Lab #5 Appendix

Note: Following the textbook, [1..n] indexing is presented; most implementation languages use [0..n-1].

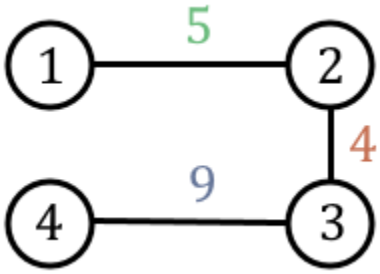
Adjacency Matrix - represents graphs using an array (*examples below*). Used by algorithms 3.4, 3.5.

W[i][j] = distance from vertex i to vertex j.

Directed vs undirected graphs:



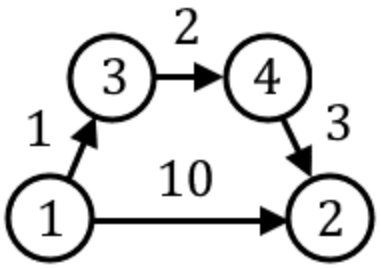
W	1	2	3	4
1	0	5	∞	∞
2	∞	0	4	∞
3	∞	∞	0	9
4	∞	∞	∞	0



W	1	2	3	4
1	0	5	∞	∞
2	5	0	4	∞
3	∞	4	0	9
4	∞	∞	9	0

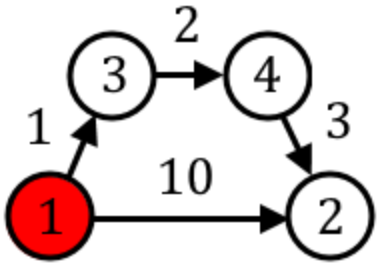
Floyd's algorithm

The algorithm works by adding intermediate vertices one-by-one and checking if these intermediate vertices improve any of the current shortest paths. $D^{(k)}$ represents the shortest paths using (1, 2, … , k) as intermediate vertices.



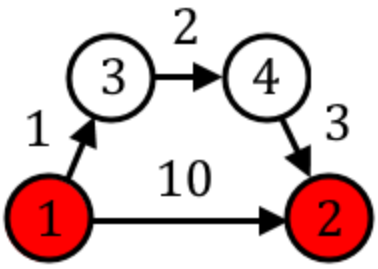
$D^{(0)}$	1	2	3	4
1	0	10	1	∞
2	∞	0	∞	∞
3	∞	∞	0	2
4	∞	3	∞	0

$D^{(0)}$ = W.

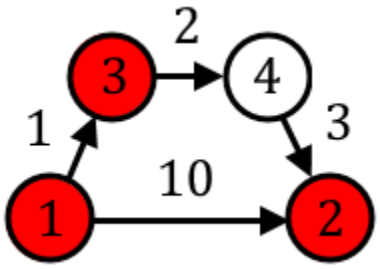


$D^{(1)}$	1	2	3	4
1	0	10	1	∞
2	∞	0	∞	∞
3	∞	∞	0	2
4	∞	3	∞	0

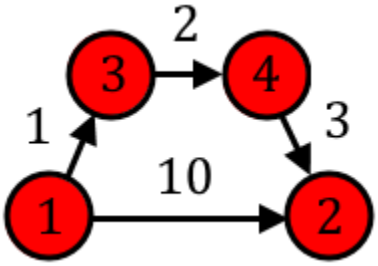
No paths can be improved using vertex 1.



$D^{(2)}$	1	2	3	4
1	0	10	1	∞
2	∞	0	∞	∞
3	∞	∞	0	2
4	∞	3	∞	0
No paths can be improved using vertex 2.				



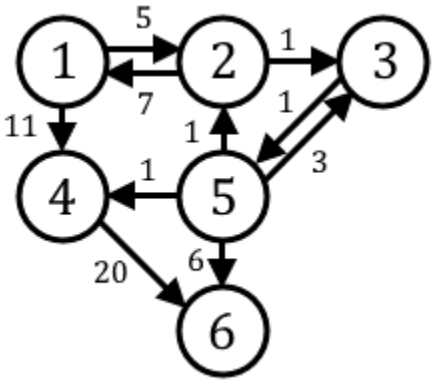
$D^{(3)}$	1	2	3	4
1	0	10	1	3
2	∞	0	∞	∞
3	∞	∞	0	2
4	∞	3	∞	0
The path from vertices 1 to 4 is improved using vertex 3 (∞→3).				



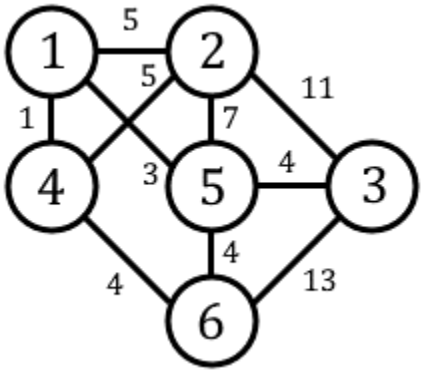
$D^{(4)}$	1	2	3	4
1	0	6	1	3
2	∞	0	∞	∞
3	∞	5	0	2
4	∞	3	∞	0
The path from vertices 1 to 2 is improved using vertex 4 (10→6). The path from vertices 3 to 2 is improved using vertex 4 (∞→5).				

Test your solution on these 2 graphs:

Graph A: directed, n = 6.



Graph B: undirected, n = 6.



Pseudo code (from Neapolitan section 3.2):

W[][] is the original adjacency matrix. W[i][j] is the distance from vertex i to vertex j.

D[][] contains our calculated shortest distances. D[i][j] contains the shortest distance from vertex i to vertex j.

P[][] is used to print the shortest paths. P[i][j] contains the last intermediate vertex on the shortest path from vertex i to vertex j (or 0 if there are no intermediate vertices).

Algorithm 3.4

```
void floyd2 (int n,
             const number W[][],
                   number D[][],
                   index  P[][])
{
  index i, j, k;

  for (i = 1; i <= n; i++)
     for (j = 1; j <= n; j++)
        P[i][j] = 0;
  D = W;
  for (k = 1; k <= n; k++)
     for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
           if (D[i][k] + D[k][j] < D[i][j]){
              P[i][j] = k;
              D[i][j] = D[i][k] + D[k][j];
           }
}
```

Algorithm 3.5

```
void path (index q, r)
{
  if (P[q][r] != 0){
     path(q, P[q][r]);
     cout << ''v'' << P[q][r];
     path(P[q][r], r);
  }
}
```