1. Your complexity analysis (theoretical) and performance results (empirical) for each algorithm.

	Recursive version	Non-recursive version
Time complexity	O(N^3)	O(N^3)
Space complexity	O(N)	O(N^2)
Performance results: for 4x4 matrix		
case: positive edge	0.000016 seconds	0.000008 seconds
case: negative edge while without negative cycle	0.000015 seconds	0.000006 seconds
case: negative edge and negative cycle(just curious)	0.000026 seconds	0.000016 seconds

- Complexity analysis of recursion implementation:

the recursive stack of floydWarshallRecursive function will hold K items in max where k is the number of vertices of graph; for each vertices, need to calculate and update the cost for edge i-j, and which is a double loop for all the nodes, within this double-loop is the recursion stack; therefore the time complexity is still O(N^3);

for space complexity, it writes the updated distance in place to original graph matrix but the recursion stack will cost no more than O(n) where n is the number of nodes therefore the space complexity is O(N), no extra matrix is used from my implementation.

- Complexity analysis of non-recursion implementation:

the initialization of distance matrix is O(N^2); the triple loop to update distance is O(N^3) where N is the number of nodes and calculate and update the distance is constant operation therefore the time complexity is O(N^3)

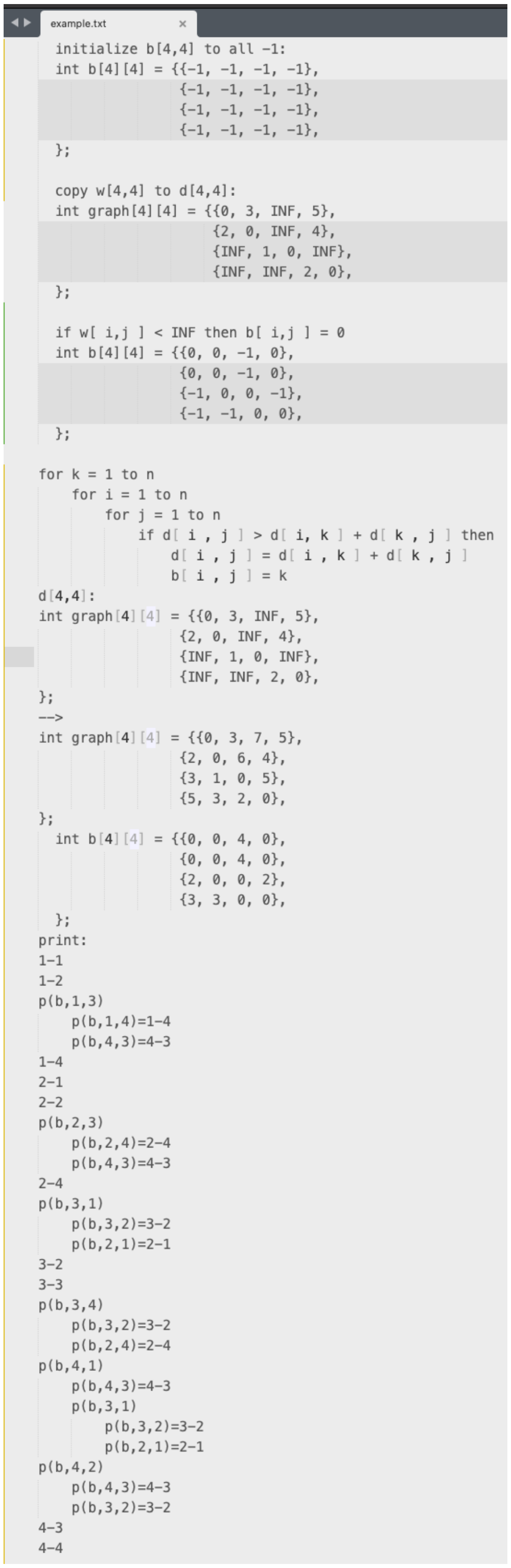
it only uses one extra distance matrix to store the updated distance between pair of nodes therefore its space complexity is O(N^2)

according to the diagram about empirical running time comparison, due to the recursion overhead raised by maintaining the call stack, the running time performance of recursion Floyd Warshall performs worse than iterative version, even though that most of the times recursion solution is easier for programmers to understand and implement, but for this problem, I think iterative implement is more intuitive and performs better.

2. Why don’t we take the recursion out of the funky printing function too, since we went to all the trouble to get rid of it in the algorithm? Please explain your answer!

refer to example.txt file and graph picture which have the example for the print function

the funky printing function first skip self-loop and directed-connected nodes where it just prints two nodes; then it assigns the shortest path mid node k as the value of b[i,j] and recursively print path from i->k and k->j thus it print the shortest path among all nodes.



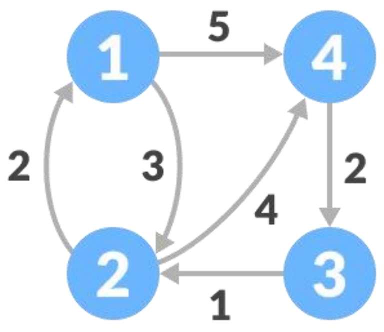
example.txt

```
 initialize b[4,4] to all -1:
 int b[4][4] = {{-1, -1, -1, -1},
               {-1, -1, -1, -1},
               {-1, -1, -1, -1},
               {-1, -1, -1, -1},
 };

 copy w[4,4] to d[4,4]:
 int graph[4][4] = {{0, 3, INF, 5},
                   {2, 0, INF, 4},
                   {INF, 1, 0, INF},
                   {INF, INF, 2, 0},
 };

 if w[ i,j ] < INF then b[ i,j ] = 0
 int b[4][4] = {{0, 0, -1, 0},
               {0, 0, -1, 0},
               {-1, 0, 0, -1},
               {-1, -1, 0, 0},
 };

for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            if d[ i , j ] > d[ i, k ] + d[ k , j ] then
                d[ i , j ] = d[ i , k ] + d[ k , j ]
                b[ i , j ] = k
d[4,4]:
int graph[4][4] = {{0, 3, INF, 5},
                {2, 0, INF, 4},
                {INF, 1, 0, INF},
                {INF, INF, 2, 0},
};
-->
int graph[4][4] = {{0, 3, 7, 5},
                {2, 0, 6, 4},
                {3, 1, 0, 5},
                {5, 3, 2, 0},
};
 int b[4][4] = {{0, 0, 4, 0},
               {0, 0, 4, 0},
               {2, 0, 0, 2},
               {3, 3, 0, 0},
 };
print:
1-1
1-2
p(b,1,3)
    p(b,1,4)=1-4
    p(b,4,3)=4-3
1-4
2-1
2-2
p(b,2,3)
    p(b,2,4)=2-4
    p(b,4,3)=4-3
2-4
p(b,3,1)
    p(b,3,2)=3-2
    p(b,2,1)=2-1
3-2
3-3
p(b,3,4)
    p(b,3,2)=3-2
    p(b,2,4)=2-4
p(b,4,1)
    p(b,4,3)=4-3
    p(b,3,1)
        p(b,3,2)=3-2
        p(b,2,1)=2-1
p(b,4,2)
    p(b,4,3)=4-3
    p(b,3,2)=3-2
4-3
4-4
```



3. Now that you know a little more about this algorithm, if you were given a graph with only positive edge weights, would you use this algorithm, or just apply Dijkstra's to every node in the graph? Please explain your answers!

I will prefer to use Floyd-Warshall

since the time complexity of applying Dijkstra would be O(V*ElogV), considering O(E)~O(V^2) it would cost O(V^3*logV) slower than Floyd-Warshall's O(V^3);

for the space complexity, if use Dijkstra on all nodes and each node to calculate the shortest path, it would cost O(V^3) space while Floyd-Warshall can be implemented in space(just recursion space cost O(V) or use extra matrix (V^2) see above to refer space complexity about Floyd-Warshall);

but it also depends on whether I want to find single-source shortest path(use Dijsktra) or all-pairs shortest path(use Floyd-Warshall)

for algorithm perspective, Dijsktra is using greedy while Floyd-Warshall is using dynamic programming;

Floyd-Warshall is easier to implement and more suitable to use for distributed systems;

but according to research, Dijkstra performs better for sparse graph than Floyd-Warshall

Distributed system application using Floyd-Warshall: http://parallelcomp.github.io/Floyd.pdf