The solution provided is a Python script that solves a shortest path problem using the Bellman-Ford algorithm combined with dynamic programming. The problem aims to find the minimum cost to visit all bathhouses in a grid. Each cell of the grid has a height, and the moving cost between cells depends on the height difference.

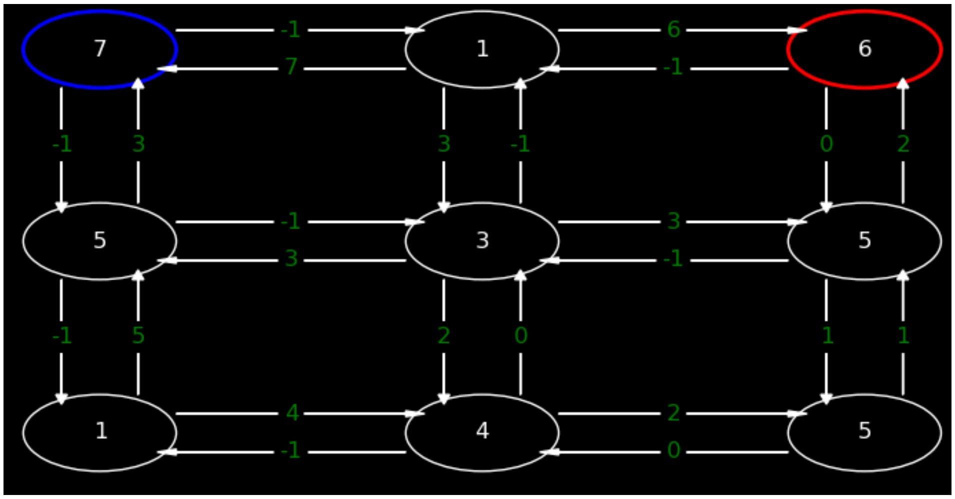
First, the script reads the grid from a file using the read_grid function. It then creates a graph representation of the grid using the create_graph function. This graph is a dictionary where each key is a tuple representing a cell's coordinates, and the value is a list of tuples. These tuples represent the neighboring cells and the costs to move to them.

The bellman_ford function implements the Bellman-Ford algorithm to find the shortest path from a source cell to a target cell in the graph. The shortest_paths function utilizes the Bellman-Ford algorithm to determine the shortest paths from a source cell to all other cells in the grid. The shortest_path_to_baths function uses these shortest paths to find routes from a source cell to all bathhouses.

The min_cost_to_visit_all_baths function employs dynamic programming to calculate the minimum cost to visit all bathhouses. It uses a bit mask to represent the state of visited bathhouses and a 2D array dp to store the minimum costs. The recursive equation is dp[visited | (1 << next)][next] = min(dp[visited | (1 << next)][next], dp[visited][current] + shortest[current][baths[next]]). Here, visited is the bit mask of visited bathhouses, current is the index of the current bathhouse, next is the index of the next bathhouse to visit, and shortest is a dictionary that stores the shortest paths from each bathhouse to all others.

The time complexity of this algorithm is O(n^2 * 2^n), where n is the number of bathhouses. This complexity comes from the 2^n possible states for the visited bathhouses, and for each state, checking all pairs of bathhouses is necessary.

I also used matplotlib to help me visualize the bathhouses and the connections between the nodes with their respective weights. If you have matplotlib installed feel free to try it out. Here is the test case example that was in the paper:


7
-1
7
1
6
-1
6
-1
3
3
-1
0
2
5
-1
3
3
3
-1
5
-1
5
2
0
1
1
1
4
-1
4
2
0
5