CS 333 Project 2 Report

Utku Çelebiöven S017965

Implementation Details

While implementing the solution, I defined the “solve_dynamically” function that takes the number of outlets, the list of outlets and lamps as string arrays. Then the function solves the problem and returns the solution as a string array containing the codes of the lamps which can be connected. To create a table for dynamic programing approach I used the two-dimensional string array data structure.

Dynamic Programming Solution

This code defines a function that employs a dynamic programming methodology to address the issue of finding a wiring that turns on the maximum number of lamps. The function would take a bottom-up method, solving smaller subproblems and accumulating the solutions in an array as it goes. The function first determines if the first outlet can be connected to the first lamp, then if the first two outlets can be connected to the first two lamps, and so on. It quickly determines the maximum number of lamps that can be wired without wires crossing each other by using the results of these calculations, which are stored in a two-dimensional array.

```
private static String[] solve_dynamically(int num_outlets, String[] outlets, String[] lamps) {

    int[][] table = new int[num_outlets][num_outlets];

    for (int i = 0; i < num_outlets; i++) {
        for (int j = 0; j < num_outlets; j++) {
            if (outlets[i].equals(lamps[j])) {
                if(i > 0 && j > 0){
                    table[i][j] = table[i][j - 1] + 1;
                }else{
                    table[i][j] = 1;
                }
            }
            else {
                if(i > 0 && j > 0){
                    table[i][j] = Math.max(table[i][j - 1], table[i - 1][j]);
                }else if(i > 0){
                    table[i][j] = table[i - 1][j];
                }
                else if(j > 0){
                    table[i][j] = table[i][j-1];
                }
            }
        }
    }
```

```
    String[] onlamps = new String[table[num_outlets - 1][num_outlets - 1]];
    int n = 0;
    int j = num_outlets;
    for (int i = num_outlets - 1; i >= 0; i--) {
        if (table[i][j - 1] != 0 && table[i][j - 1] != table[i - 1][j - 1]) {
            onlamps[n] = outlets[i];
            n++;
            j = j - table[i - 1][j - 1];
        }
    }

    return onlamps;

}
```