CS333Project 2

Berkecan Koçyiğit S0210632

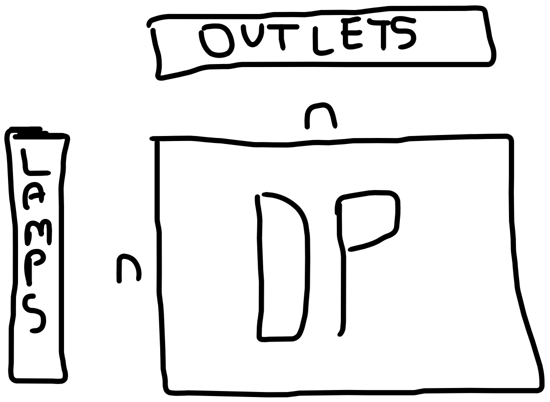
I have one main function that handles everything. Main function consists of:

- Scanner
- 2 for loops for creating lamps and outlets arrays.
- 1 for loop for creating DP array.
- 2 nested for loops for dynamic programming.
- Printing max lamp count.
- 1 for loop for printing lamps which can be connected to outlets.

I took numbers of outlets/lamps form user with scanner. Then I created lamps and outlet arrays with sizes of n. Then I assigned each item to outlet for one line then same to second line for lamps.

After initialization I created 2-dimensional array called DP and initialized with zeros. This is a longest common subsequence problem.

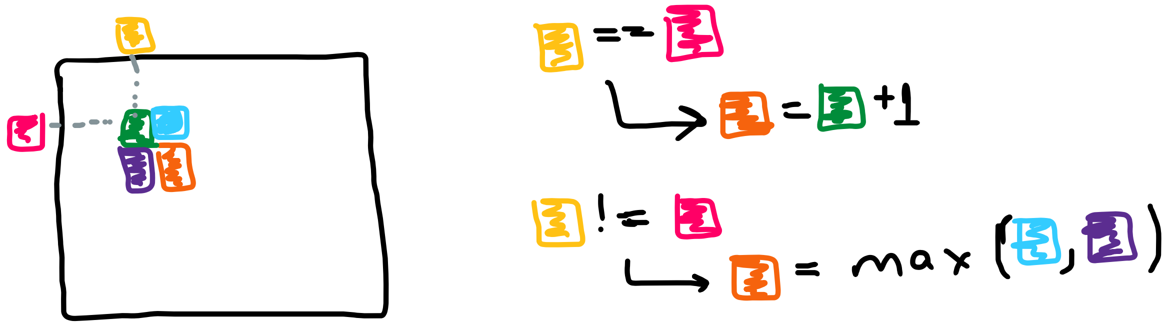
I check every DP[n][n]/outlets[n]/lamps[n] with two nested for loops. You can think DP array as:



$(\text{LAMPS}) \cdot (\text{OUTLETS})^T$

(figuratively)

But different from dot product I check for every i-1 and j-1 outlet lamp sequence and if those are same I assign DP[i][j] with increment of DP[i-1][j-1]. If not, I took biggest of DP[i-1][j] and DP[i][j-1].



LCS(X, Y) =

1. if X[n] == Y[n]
   then LCS(X[1...n-1], Y[1...n-1]) + 1
2. else
   LCS(X[1...n-1], Y[1...n])
   LCS(X[1...n], Y[1...n-1])

After doing this calculation for each nXn, I got maximum amount of connections count in DP[n][n].

After printing maximum number. I have a while loop which backtracks DP array to find lamps that can be connected. To find the sockets that can be connected, the code initializes the current row and column indexes to the last row and column in the DP array, respectively. It then loops until the current row or column index is zero, and in each iteration, it compares the current outlet code and lamp code. If the codes match, it prints the current socket/lamp pair and moves to the previous row and column in the DP array. If the codes do not match, it moves to the previous row or column in the DP array depending on the value in the current element in the DP array. This allows the code to trace back through the DP array and find the sockets that can be connected.

The code has O(n^2) time complexity because it uses a two-dimensional DP (dynamic programming) array of size n x n, and it loops through the elements of the two sequences twice, once to fill in the DP array and once to find the longest common subsequence by comparing the outlet codes and lamp codes.

In the worst case, where the two sequences have no elements in common, the algorithm will have to compare every element of the first sequence with every element of the second sequence, resulting in O(n^2) comparisons. In the best case, where all elements of the two sequences are the same, the algorithm will only have to compare the first element of the first sequence with the first element of the second sequence, resulting in only O(1) comparisons.

Overall, the time complexity of the LCS algorithm is O(n^2), making it efficient for relatively small sequences, but potentially impractical for very large sequences.