HW6 - S15
Warmuth
Justin Wong (jujwong)
5/19/15

1 Determine the longest common subsequence of x=(1,0,0,1,0,1,0,1) and y=(0,1,0,1,1,0,1,1,0)

		1	0	0	1	0	1	0	1
	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1
1	0	1	1	1	2	2	2	2	2
0	0	1	2	2	2	3	3	3	3
1	0	1	2	2	3	3	4	4	4
1	0	1	2	2	3	3	4	4	5
0	0	1	2	3	3	4	4	5	5
1	0	1	2	3	4	4	5	5	6
1	0	1	2	3	4	4	5	5	6
1	0	1	2	3	4	5	5	6	6

2 Shuffle

To determine if Z, size r, is a shuffle of X, size n, and Y, size m, we will create a table of n x m size of booleans. The index at S[i][j] will only be true if the first i+j charcters of Z are a shuffle of the first i characters of X and the first j characters of Y. S will hold all values that we compute and for all possible sub-problems, but each one is only calculated one time. We fill the array from the smaller values, starting at the top left corner of the array, we eventually want to find S[n][m] at the bottom right.

First we check the length of Z, to make sure that it is equal to the length of n + m, if not than it obviously can not be the shuffle of X and Y. If this passes then we will mark S[0][0] as true as an initialization.

Now we will check the values of each characters using for loops:

```
for(int i = 1; i ¡ n; ++i)
    S[i][0] = S[i-1][0] AND (Z[i-1] == X[i-1])
for(int j = 1; j ¡ m; ++j)
    S[0][j] = S[0][j-1] AND (Z[j-1] == Y[j-1])
for(int i = 1; i ¡ n; ++i)
  for(int j = 1; j ¡ m; ++j)
    S[i][j] = ((Z[i+j-1] == X[i-1]) AND S[i-1][j]) OR
      ((Z[i+j-1] == Y[j-1]) AND S[i][j-1])
    return S[n-1][m-1]
```

The runtime is O(nm), it does constant work of checking characters if they are matching and must iterate through the entire n x m array.

3 Chessboard The goal is to move from the bottom edge to the top edge According to the rules, that each step is either 1. imeddiately above, 2. one up and one to the left if not in the leftmost column, 3. one up and one right if not already in rightmost column.

Our subproblem is how to get from row 1 to a square (i,j) and make it the most profitable. Each step we take, we will earn d + p(x,y), where d is the dollars earned for that step previous. So d[i,j] is the most profit we can earn on our way to (i,j), d[1,j] = 0 for all j = 1,2,...,n. Thus for 1 ¿ i ¿ n we have this recurrence:

$$d[i,j] = max \begin{cases} Rule1 : d[i-1, j+1] + p(n(i-2) + j + 1, 1) : j < n \\ Rule2 : d[i-1, j] + p(n(i-2) + j, 2) : always \\ Rule3 : d[i-1, j-1] + p(n(i-2) + j - 1, 3) : j > 1 \end{cases}$$

We are trying to get to space (i,j) in the most profitable way for all possible (i,j) pairs. There will be a total of n^2 subproblems, and we will keep track of these in a table d of n x n size, we will also use another table r to keep track of which were used for each move made. For j = 1 to n

Initialize both tables d[1,j] and w[1,j] = 0.

for i = 2 to n

for j = 1 to n

if rule 1: d[i-1, j+1] + p(n(i-2) + j + 1, 1) ¿ d[i,j] and set d[i,j] = d[i-1, j+1] + p(n(i-2) + j + 1, 1) and w[i,j] = 1

if Rule 2: d[i-1,j] + p(n(i-2) + j, 2) ¿ d[i,j] set d[i,j] = d[i-1,j] + p(n(i-2) + j, 2) and w[i,j] = 2

if Rule 3: d[i-1,j-1] + p(n(i-2) +j -1, 3) ¿ d[i,j]

set d[i,j] = d[i-1,j-1] + p(n(i-2) +j -1, 3) and w[i,j] = 3

To recover the solution: find which column on row n is part of an optimal solution and check w for what rule was used to move to the next space.

This algoritm will run in O(n^2) due to the size of the array being n^2 and the number of times each loop, of the nest loop, is run O(n) times.