

Homework 8

Brian Knotten, Brett Schreiber, Brian Falkenstein

September 13, 2018

21

If it is the case that both the husband and the wife have the same list, then there is not enough information to determine if a fair partition is possible.

On both the husband's and the wife's turns they will greedily select the highest available item on their respective lists. They will alternate between the husband and the wife according to the following pattern: H, W, WH, WHHW, ... where each step of the pattern is defined as the inverse of the prior sequence.

Assuming each item on the list has a distinct value, the highest item on a list has value at least $\Sigma/n + \epsilon$ where Σ = the total value of all items on the list, n = the number of items on the list, and ϵ = some small constant value. It follows that all subsequent items on the list must have value at most $\Sigma/n + \sigma$ where σ is a smaller constant than that of the item directly above it on the list.

Because the algorithm always takes the highest available item on each list it will always produce the highest value combination of items using the above values for both lists. If both the husband and the wife are not satisfied with this combination, then we do not have enough information to determine if a fair partition is possible.

$OPT(I)$ has H take G_m at time k .

$A(I)$ has H take G_h at time k .

G_h has higher value than G_m because $A(I)$ took it and OPT and A agreed up to this point.

Construct $OPT'(I)$ by having $OPT'(I)$ take G_h instead of G_m .

$OPT'(I)$ obviously agrees for one more step and increases H 's overall value so it is still over 50%.

2

```
int longestCommonSubsequence(A, l, B, m, C, n) {
    int LCS[l + 1][m + 1][n + 1];

    for(int i = 0; i <= l; i++) {
        LCS[i][0][0] = 0;
    }

    for(int i = 0; i <= m; i++) {
        LCS[0][i][0] = 0;
    }

    for(int i = 0; i <= n; i++) {
        LCS[0][0][i] = 0;
    }

    for(int i = 1; i <= l; i++) {
        for(int j = 1; j <= m; j++) {
            for(int k = 1; k <= n; k++) {
                if(A[i - 1] == B[j - 1] && A[i - 1] == C[k - 1]) {
                    LCS[i][j][k] = LCS[i - 1][j - 1][k - 1] + 1;
                } else {
                    LCS[i][j][k] = max(LCS[i][j][k - 1], LCS[i][j - 1][k], LCS[i - 1][j][k]);
                }
            }
        }
    }
}
```

```

    }
}

return LCS[l, m, n];
}

```

3

Given strings A and B of length N and M respectively:

```

dyn_array = [N+1][M+1]

for i in range(0, N + 1):
    dyn_array[0, i] = i

for i in range(0, M + 1):
    dyn_array[i, 0] = i

for i in range(0, N + 1):
    for j in range(0, M + 1):
        if A[i] == B[j]:
            dyn_array[i][j] = dyn_array[i-1][j-1] + 1
        else:
            dyn_array[i][j] = min(dyn_array[i-1][j], dyn_array[i][j-1])

return dyn_array[N+1][M+1]

```

a.

	null	z	x	y	y	z	z
null	0	1	2	3	4	5	6
z	1	1	2	3	4	5	6
z	2	2	3	4	5	5	6
y	3	3	4	4	5	6	7
x	4	4	4	5	6	7	8
z	5	5	5	6	7	7	8
y	6	6	6	6	7	8	9

b.

To find the length, simply look at the last index in the array (index $[N, M]$).

c.

Starting at the bottom right, trace backwards. If the letters do not match, and index $[i-1, j-1]$ does not contain a value of one less than that at $[i, j]$, then take both letters (the i 'th letter in A , and the j 'th letter in B). If the letters do match, move to index $[i-1, j-1]$.