CS 331: Algorithms and Complexity (Spring 2024)
Unique Number: 50930, 50935 50940, 50945

Assignment 5

Due on Tuesday, 5 March, by 11.59pm

Problem 1

(8 pts)

(a) (6 pts)

A naive solution would be to split into 3 cases, one for each of the 3 possible operations
(1) Adding a gap to the first string
(2) Adding a gap to the second string
(3) Including characters in both strings
OPT(i, j) = $\min(\alpha_{x_i y_i} + OPT(i-1, j-1), \delta + OPT(i-1, j), \delta + OPT(i, j-1))$
This yields a time complexity of $O(3^{m+n})$
Example: (CH, EN)
First layer: (C, E), (C, _), (_, E)
Second layer: (CH, EN), (CH, E_), (C_, EN), (CH, EN), (CH, __), (C_, _E), (_C, EN), (_C, E_), (__, EN)

(b) (2 pts)

1.

_	A	L	G	O
T	1	2	3	4
E	2	2	3	4
S	3	3	3	4
T	4	4	4	4

$Alignment_1$: ALGO
$Alignment_2$: TEST

2. The cost is the value in the bottom right corner, 4
Cost is 4 since there are 4 sets of characters that are different

Problem 2

(12 pts)

(a) Indices (i, j) store the minimum number of cuts needed to make the substring from i to j a set of palindromes.

```
def min_palindrome(s):
    sections = [∞] * len(s)
    for j in [1, len(s)]:
        min_sections = j
        for i in [1, j]:
            if isPali(i, j):
                min_sections = min(min_sections, sections[i - 1] + 1)
        sections[j] = min_sections
```

I'll test by running it on the string 'coffee'
Each iteration runs with a fixed endpoint j and a increasing start point i.

First iteration: $(1 \to 1, 1)$

1	∞	∞	∞	∞	∞

Second iteration: $(1 \to 2, 2)$

1	2	∞	∞	∞	∞

Third iteration: $(1 \to 3, 3)$

1	2	3	∞	∞	∞

Fourth iteration: $(1 \to 4, 4)$

1	2	3	3	∞	∞

Fifth iteration: $(1 \to 5, 5)$

1	2	3	3	4	∞

Sixth iteration: $(1 \to 6, 6)$

1	2	3	3	4	4

(b)

Problem 3

(10 pts)

(a) Since we can't include the direct children of a manager, then we need to include subtrees 3 layers deep.
This isn't greedy since local best choices don't always lead to the global best choice.

(b) I used memoization to store the maximum enjoyment for each person.

```
Map(person, enjoyment) = {}
def max_enjoyment(person):
    if person is None:
        return 0
    if person in Map:
        return Map(person)
```

Problem 3 continued on next page. . .

```
    ce = 0
    for child in person.children:
        ce += max_enjoyment(child)

    gce = 0
    for child in person.children:
        for grandchild in child.children:
            gce += max_enjoyment(grandchild)
    Map(tree, max(ce, gce + person.enjoyment))
    return max(ce, gce + person.enjoyment)
```

Time complexity is $O(n)$