Homework 7

Wendi Chen

1 Bonus: Edit distance

1.1 Problem analysis

As the book describes, the individual costs of the copy and replace operations are less than the combined costs of the delete and insert operations. Similarily, we can also assert that the individual cost of copy is less than the individual cost of replace, otherwise the copy operation would not be used.

It's naturally to find that this is a problem that can be divided into subproblems that overlap. So dynamic programming is a good way to solve it. We assume $c[i, j](0 \le i \le m, 0 \le j \le n)$ represents the least operation cost of tansforming $x[1..i]$ to $y[1..j]$. At the same time, we assume $cost[i](1 \le i \le 6)$ represents the cost of **Copy, Replace, Delete, Insert, Twiddle, Kill** respectively and $op[i, j](0 \le i \le m, 0 \le j \le n)$ for recording the last operation when tansforming $x[1..i]$ to $y[1..j]$.

1.2 Pseudo-code

Algorithm 1 Pseudocode of the Dynamic Programming to Solve Edit Distance

Input: x, y: two input strings
Output: c, op: two 2-D arrays num: the number of characters in x that has been examined

create $c[0..m, 0..n], op[0..m, 0..n]$
create $current\_cost[0..6]$
set $c[0, 0] = 0$
set $op[0, 0] = 0$
set $num = 0$
for $i = 1$ to m **do**
    $c[i, 0] = cost[3] + c[i-1, 0]$
    $op[i, 0] = 3$
end for
for $i = 1$ to n **do**
    $c[0, n] = cost[4] + c[0, i-1]$
    $op[0, i] = 4$
end for
if $n == 0$ and $cost[6] < c[m, 0]$ **then**
    $c[m, 0] = cost[6]$
    $op[m, 0] = 6$
end if

```
for i = 1 to m do
    for j = 1 to n do
        for k = 1 to 6 do
            current_cost[k] = ∞
        end for
        if x[i] == y[j] then
            current_cost[1] = cost[1] + c[i − 1][j − 1]
        else
            current_cost[2] = cost[2] + c[i − 1][j − 1]
        end if
        current_cost[3] = cost[3] + c[i − 1][j]
        current_cost[4] = cost[4] + c[i][j − 1]
        if 2 ≤ i and 2 ≤ j and x[i − 1] == y[j] and x[i] == y[j − 1] then
            current_cost[5] = cost[5] + c[i − 2][j − 2]
        end if
        if i == m and j == n then
            for k = 1 to m − 1 do
                if cost[6] + c[k, n] < current_cost[6] then
                    current_cost[6] = cost[6] + c[k, n]
                    num = k
                end if
            end for
        end if
        c[i, j] = ∞
        for k = 1 to 6 do
            if current_cost[k] < c[i, j] then
                c[i, j] = current_cost[k]
                op[i, j] = k
            end if
        end for
    end for
end for
return c, op, num
```

Algorithm 2 Pseudocode of Printing an Optimal Operation Sequence

Input: *op*: operation 2-D array *num*: the number of characters in x that has been examined

```
function PRINT_SEQ(op, num, i, j)
    if op[i, j] == 0 then
        return
    else if op[i, j] == 1 then
        PRINT_SEQ(op, num, i-1, j-1)
        print copy
    else if op[i, j] == 2 then
        PRINT_SEQ(op, num, i-1, j-1)
        print replace
    else if op[i, j] == 3 then
        PRINT_SEQ(op, num, i-1, j)
        print delete
```

```
        else if op[i, j] == 4 then
            PRINT_SEQ(op, num, i, j-1)
            print insert
        else if op[i, j] == 5 then
            PRINT_SEQ(op, num, i-2, j-2)
            print twiddle
        else if op[i, j] == 6 then
            PRINT_SEQ(op, num, num, j)
            print kill
        end if
    end function
```

1.3 Complexity

From these two nested loops we can get the time complexity and the space complexity of the whole algorithm are both $O(mn)$.