ICCS313: Assignment 4

Natthakan Euaumpon
natthakaneuaumpon@gmail.com
November 2019

1: problem1

Let rod(length,price,cost) be a function to solve for the problem using memoization. Then the recurrence would be:

$$rod(length, price, cost) = \begin{cases} 0 & \text{if } smallest\_length > n \geq 0 \\ \max\limits_{1 \leq i \leq n} (rod(n - l_i) + price[i] - cost) & \text{if } n >= smallest\_length \end{cases}$$

the code is:

```
1   def rod(length, price, cost):
2       value = list()
3       value.append(0)
4       m = -1
5       for i in range(1,length):
6           m = price[i]
7           for j in range(1,i-1):
8               m = max(m,price[i]+value[i-j]-cost)
9           value[i] = max
10      return value[length]
```

Prove that this code give optima solution:
Assume for the sake of contradiction that this code does not produce optimal solution. This mean this non-optimal code will produces value a and an optimal code will produce b. Where a ¡ b and a and b are both max value from each code. But the code always give out the maximum possible value. Hence, this code gives out optimal solution.
The time complexity for this algorithm is $O(n^2)$. This is because each loop run O(n) time and it is nested. Other operation only take O(1) time. So the time complexity is $O(n^2)+O(1) = O(n^2)$.

2: Problem2

Let arr be an array which keep track of the maximum sum and arr[i] is the total sum at position i. So, the total sum of the next position will be arr[i+1]. This mean we can choose the max value between the 2 by comparing. Let say arr[i-1] is k and arr[i] is a. Then the max is either $a + k$ or a. So, the recurrence is:

$$arr(i) = \begin{cases} 0 & \text{if } i <= 0 \\ max(arr(k + a, a) & \text{if } i > 0 \end{cases}$$

the code is:

```java
import java.util.HashSet;


public class contiguousSubsequence {
    static int contiguous(int[] a) {
        int length = a.length;
        int max = Integer.MIN_VALUE;
        int[] arr = new int[length];
        for (int i = 0; i < length; i++) {
            if (i > 0){
                arr[i] = Math.max(arr[i-1]+a[i],a[i]);
            }
        }
        for (int i = 0; i < length; i++) {
            if (arr[i] > max){
                max = arr[i];
            }
        }
        return max;
    }
}
```

This code take in array a as an array if integer which is a given input and return the total sum of the contiguous sub-sequence. Both for-loop take O(n) time each and argument inside both loop take constant time. So, the time complexity for this program is O(n).

3: Problem3

Let s be a string input and r be a reverse of s. We can use both string to find the answer. This is because the sting is palindrome. We can find the answer by finding the longest common sub-sequence between s and r. Let length(i,j) be the array which keep track of the maximum length of the common sub-sequence, the recurrence for this code would be:

$$length(i,j) = \begin{cases} 0 & \text{if } i < 0 \vee j < 0 \\ 1 + length(i-1, j-1) & \text{if } s[i] = r[j] \\ max \begin{cases} length(i-1, j) \\ length(i, j-1) \end{cases} & \text{if } s[i] \neq r[j] \end{cases}$$

the code is:

```java
public class Palindrome {
    static String palindrome(String s){
        StringBuilder r = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            r.append(s.charAt(i));
        }
        int nested[][] = new int[s.length()][s.length()];
        for (int i = 0; i < s.length() ; i++) {
            for (int j = 0; j < s.length() ; j++) {
                if (i == 0 || j == 0){
                    nested[i][j] = 0;
                }else if (s.charAt(i-1)== r.charAt(j-1)){
                    nested[i][j] = nested[i-1][j-1]+1;
                }else{
                    nested[i][j] = Math.max(nested[i-1][j], nested[i][j-1]);
                }
            }
        }
        int index = nested[s.length()-1][s.length()-1];
        char[] p = new char[index+1];
        int i = s.length();
        int j = s.length();
        while (i > 0 && j > 0){
            if (index != 0) {
                if (s.charAt(i-1) == r.charAt(j-1)) {
                    p[index-1] = s.charAt(i-1);
                    i--;
                    j--;
                    index--;
                }
            }else if (nested[i-1][j] > nested[i][j-1]){
                i--;
            }else {
                j--;
            }
        }
        return Arrays.toString(p);
    }
}
```

The time complexity for this code is $O(n^2)$. This is because we have one nested for loop which O(n) time each for each loop. We also have one while loop which run within O(n) time complexity. The operation inside and out side the loop all take O(1). So, the time complexity

for this program is $O(n^2)$.

4: Problem4

Natthakan Euaumpon
@natthakaneuaump1