ICCS 313: Assignment 4

Tawan Chaeyklinthes u5980963

Date : 12 Nov 2018

Problem 1

Assuming that we have $P_0, P_1, ..., P_k$ which indicates prices for the rod. Let $value(x)$ be the maximum value of a rod of length x. We can then write the recurrence as follows:

$$value(x) = \begin{cases} 0, & \text{if } x \leqslant 0. \\ \max_{i \leqslant k}(P_i + value(x - i) - cut(i, x)), & \text{if } x > 0. \end{cases}$$

Let $cut(i, x)$ be an indicator to whether or not are we cutting the rod. If we are cutting the rod, return c ,otherwise, return 0.

$$cut(i, x) = \begin{cases} 0, & \text{if } i == x. \\ c, & \text{otherwise.} \end{cases}$$

This can be translated into code as such:

```
def rod_cutting(n,P):
    values = [0]*(n+1)
    for x in range(1,n+1):
        for i in range(len(P)):
            val = values[x-i] + P[i]
            if(i != x):
                val -= c
            values[x] = max(val,values[x])
    return values[n]
```

Problem 2

Let array $best\_seq[x]$ keep the best contiguous subsequence from elements index $0, 1, ..., x$.

The algorithm is to keep track of the running sum while updating the $best\_seq$ .If the running sum is < 0, for instance at index i , reset the sum.

This can be written in code as follows:

```
def max_sequence(S):
    best_seq = [-INF]
    current_seq = []
    sum = 0
    for i in range(len(S)):
        sum += S[i]
        current_seq.append(S[i])
        if(sum > sum(best_seq)):
            best_seq = current_seq.copy()
```

```
    if(sum < 0):
      sum = 0
      current_seq = []
  return best_seq
```

Problem 3

Let $longest\_p(i,j)$ be the function that gets the longest palindrome of range i to j. We can write a recurrence relation as follows:

$$longest\_p(i,j) = \begin{cases} \text{empty string}, & \text{if } i > j. \\ s[i], & \text{if } i == j. \\ longest \begin{cases} longest\_p(i+1,j) \\ longest\_p(i,j-1) \end{cases} \underset{\text{if } s[i]==s[j]}{+ longest\_p(i+1,j-1)}, & \text{otherwise.} \end{cases}$$

This can be translated into codes as follow:

```
def longest_p(i,j):
  if(i == j):
    return s[i]
  if(i > j):
    return ""
  if(memo[i][j] != null):
    return memo[i][j]
  else:
    longest_palin_without_j = longest(i,j-1)
    longest_palin_without_i = longest(i+1,j)
    longest_palin = longest(longest_palin_without_j,
                            longest_palin_without_i)
    if(s[i] == s[j]):
      longest_palin = s[i] +longest(i+1,j-1) + s[j]
    memo[i][j] = longest_palin
    return longest_palin
```

There are $O(n^2)$ sub-problems and each costing $O(1)$. Therefore the total running time is $O(n^2)$.