3CP02

DESIGN AND ANALYSIS OF ALGORITHMS

QUESTIONS OF ASSIGNMENT

21CP027 JAY KALBI

CONTENTS :

- Gas Station Problem : Greedy Algorithm
- Pascal's Triangle : Dynamic Programming
- Longest Palindrome Substring : Dynamic Programming

❖ GAS STATION PROBLEM :

❑ Problem Description :

- There are **N** gas stations along a circular route, where the amount of gas at station **i** is **arr[i]** . You have a car with an unlimited gas tank and it costs **cost[i]** of gas to travel from station **i** to its next station **(i+1)** . At the beginning of the journey, the tank is empty at one of the gas stations.
- Return the **minimum starting gas station's index** if you need to travel around the circuit once, otherwise return -1.

❑ Problem Note :

→ Completing the circuit means starting at i and ending up at i again.

→ Both input arrays are non-empty and have the same length.

→ Each element in the input arrays is a non-negative integer.

❖ EXAMPLES :

❑ Example 1 :

- Input :

>> gas[ ] = [2 , 3 , 4]

>> cost[ ] = [3 , 4 , 3]

>> Output : -1

- Explanation :

- You can't start at station 0 or 1, as there is not enough gas to travel to the next station.
- Let's start at station 2 and fill up with 4 unit of gas. Gas available in the tank = 0 + 4 = 4
- Travel to station 0. Gas available in tank = 4 – 3 + 2 = 3
- Travel to station 1. Gas available in tank = 3 – 3 + 3 = 3
- You can't travel back to station 2, as it requires 4 unit of gas but you only have 3. Therefore, you can't travel around the circuit once no matter where you start.

❖ EXAMPLES :

❑ Example 2 :

- Input :

>> gas[ ] = [1 , 2 , 3 , 4 , 5]

>> cost[ ] = [3 , 4 , 5 , 1 , 2]

>> Output : 3

- Explanation :

- You can't start at station 0, 1 or 2 as there is not enough gas to travel to the next station.
- Let's start at station 3 (index 3) and fill up with 4 unit of gas. Gas available in the tank = 0 + 4 = 4
- Travel to station 4. Gas available in tank = 4 – 1 + 5 = 8
- Travel to station 0. Gas available in tank = 8 – 2 + 1 = 7
- Travel to station 1. Gas available in tank = 7 – 3 + 2 = 6
- Travel to station 2. Gas available in tank = 6 – 4 + 3 = 5
- Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3. Therefore **return** 3 as starting address.

❖ SOLUTION STEPS :

❑ Create a start to store the valid starting index from where the car could reach all the stations.

❑ For each station i, fill the fuel tank with gas[i] and burn the fuel by cost[i].

❑ If at any point the tank is < 0 then, choose the next index as starting point.

❑ At last, check if the total fuel available at the gas stations is greater than the total fuel burnt during the travel.

❑ Return the start.

PSEUDO CODE :

```
void circle_complete()
{
    for(int i=0;i<size_arr;i++) {
        tank = tank + gas[i] - cost[i];
        if(tank < 0) {
            start = i+1;
            total = total + tank;
            tank = 0;
        }
    }

    if(total + tank < 0) {
        cout<<"Can't travel around the circuit once no matter where you start!!!"<<endl;
    } else {
        cout<<"Starting Index to Travel is : "<<start<<endl;
    }
}
```

❖ PASCAL'S TRIANGLE :

❑ Problem Description :

- Given an integer numRows, return the first numRows of Pascal's triangle.

❑ Problem Note :

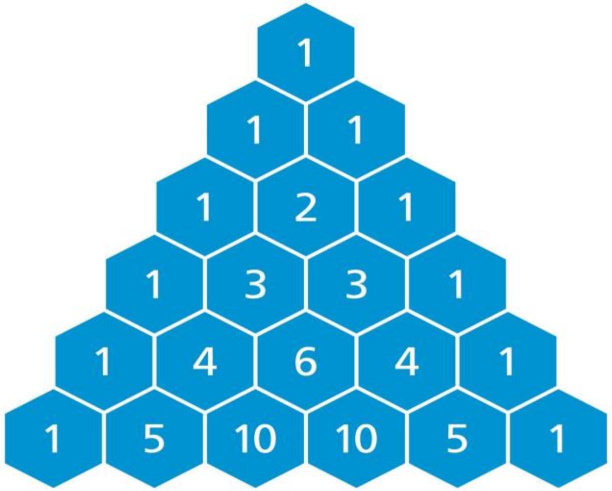
→ 1 <= numRows <= 30

❑ Examples :

- Input :

>> numRows = 5

>> Output : [[1], [1,1] , [1,2,1] , [1,3,3,1] , [1,4,6,4,1]]



❖ EXPLANATION :

- ❑ For the i^{th} row, there are i elements, where i >= 1.
- ❑ The corner elements of each row are always equal to 1.
- ❑ All the other (i, j) elements of the triangle are equal to the sum of $(i-1,\ j-1)^{th}$ and $(i-1,\ j)^{th}$ element.

❑ FORMULA :

dp[i][j] = dp[i-1][j-1] + dp[i-1][j]

where i > 1

❖ TABULATION :

i / j	j=0	j=1	j=2	j=3	j=4	j=5
i=0	0	1	0	0	0	0
i=1	0	1				
i=2	0	1	1			
i=3	0	1	2	1		
i=4	0	1	3	3	1	
i=5	0	1	4	6	4	1

❖ PSEUDO CODE :

❑ Initialize a matrix triangle[n][n] with 0

```
void calculate_tringle()
{
    for(int i=0;i<n;i++) triangle[i][0] = 0;
    for(int i=0;i<n;i++) triangle[0][i] = 0;

     triangle[0][1] = 1;

     for(int i=1;i<n;i++) {
         for(int j=1;j<i+1;j++) {
             triangle[i][j] = triangle[i-1][j] + triangle[i-1][j-1];
         }
     }
 }
```

❖ LONGEST PALINDROME SUBSTRING :

❑ Problem Description :

- Given a string s, return the longest palindrome substring in s.

❑ Problem Note :

→ s consist of only digits and English letters (lower-case and/or upper-case).

→ 1 <= s.length <= 1000

❑ Examples :

- Input :

>> s = ”babad”

>> Output : ”bab”

- Input :

>> s = ”cbbd”

>> Output : ”bb”

❖ EXPLANATION :

❑ Manacher’s Algorithm :

- The left side of a palindrome is a mirror image of its right side.
- Odd length palindrome will be centered on a letter and even length palindrome will be centered in between two letters (thus there will be total 2n+1 letters).

❑ Steps :

→ Initialize the lengths array to the numbers of possible center.

→ Set the current center to the first letter.

→ Loop while the current center is valid :

  → Expand to the left and right simultaneously until we find the largest palindrome around this center.

  → Fill in the appropriate entry in the longest palindrome length array.

  → Iterate through the longest palindrome lengths array backwards and fill in the corresponding values to the right of entry for the current center until an unknown value is encountered.

  → Set the new center to the index of this unknown value.

→ Return the longest substring.

❖ PSEUDO CODE :

- ❑ Initialize a string res with empty and resLen with 0.

```
void Calculate_Palindrome()
{
   int l,r;
   for(int i=0;i<text.length();i++) {
       //odd Length
       l=r=i;
       while(l>=0 and r<text.length() and text[l] == text[r]) {
           if((r-l+1) > resLen) {
               res="";
               for(int k=l;k<=r;k++) {
                    res += text[k];
               }
               resLen = r - l + 1;
           }
           l -= 1;
           r += 1;
       }
    }
   cout<<"Res : "<<res<<endl;
   cout<<"Reslen :"<<resLen<<endl;
}
```

```
//Even Length
l=i; r=i+1;
while(l>=0 and r<text.length()
       and text[l] == text[r]) {
    if((r-l+1) > resLen) {
        res="";
        for(int k=l;k<=r;k++) {
            res += text[k];
        }
        resLen = r - l + 1;
    }
    l -= 1;
    r += 1;
}
```

THANK YOU