Algorithms Lab (CSLR41)

106119100

End Semester Laboratory Examination

106119100 - Rajneesh Pandey

Question 1 :

1. **Find missing term in a sequence in log(n) time :** Given a sequence of numbers such that the difference between the consecutive terms is constant, find missing term in it in O(log (n)) time. (Sample Input : [5, 7, 9, 11, 15], Output: 13)

Approach

As we are asked to perform a search on a sorted array, we should always think of a **binary search algorithm**.

We can easily solve this problem in **O(log(n))** time by modifying the binary search algorithm.
The idea is to check the difference of middle element with its left and right neighbor. If the difference is not equal to the common difference, then we know that the missing number lies between the middle element and its left or right neighbor.
If the difference is the same as common difference of the sequence, reduce our search space and the left sub array **arr[low…mid-1]** or right sub array **arr[mid+1…high]** depending upon if the missing element lies on the left sub array or not.

Code and Input / Output

```cpp
// Question 1 : Find missing term in a sequence in log(n) time
// 106119100 Rajneesh Pandey

#include <bits/stdc++.h>
using namespace std;

int findMissingTerm(int arr[], int n)
{
    // search space is `arr[low…high]`
    int low = 0, high = n - 1;

    // calculate the common difference between successive elements
    int d = (arr[n - 1] - arr[0]) / n;

    // loop till the search space is exhausted
    while (low <= high)
    {
        // find the middle index
        int mid = high - (high - low) / 2;

     // check the difference of middle element with its right neighbor
        if (mid + 1 < n && arr[mid + 1] - arr[mid] != d)
        {
            return arr[mid + 1] - d;
        }

     // check the difference of middle element with its left neighbor
        if (mid - 1 >= 0 && arr[mid] - arr[mid - 1] != d)
        {
            return arr[mid - 1] + d;
        }

        // if the missing element lies on the left subarray, reduce
        // our search space to the left subarray `arr[low…mid-1]`
        if (arr[mid] - arr[0] != (mid - 0) * d)
        {
            high = mid - 1;
        }
        // if the missing element lies on the right subarray, reduce
        // our search space to the right subarray `arr[mid+1…high]`
        else
        {
            low = mid + 1;
        }
    }
}
```

```cpp
int main()
{

    int n;
    cout << "----------------------------" << endl;
    cout << "Enter the size of the array : " << endl;
    cin >> n;
    int array[n];
    cout << "Enter the Values in the array : " << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> array[i];
    }
    cout << "Missing Term is : " << endl;
    cout << findMissingTerm(array, n) << endl;
    cout << "----------------------------" << endl;

    return 0;
}
```

Input / Output

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Programs> ./Question1
----------------------------
Enter the size of the array :
5
Enter the Values in the array :
5 7 9 11 15
Missing Term is :
13
----------------------------
PS D:\Programs>
```

Question 2:

2. **Maximal Rectangle:** Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.
   Sample Input :

0	1	1	0	1
1	1	0	1	0
0	1	1	1	0
1	1	1	1	0
1	1	1	1	1
0	0	0	0	0

Output: 9

Approach

We start from the first row, and move downward;

- **height[i]** record the current number of countinous '1' in column i;
- **left[i]** record the left most index j which satisfies that for any index k from j to i, **height[k] >= height[i];**
- right[i] record the right most index j which satifies that for any index k from i to j, **height[k] >= height[i];**
- we need to update **maxArea** with value (**height[i] * (right[i] - left[i] + 1));**
  we know initially, height array contains all 0, so according to the definition of left and right array, left array should contains all 0, and right array should contain all **n-1**.
- the idea for updating the right boundary is similar, we just need to iterate from right to left

Code and Input / Output

```cpp
//Question 2 : Maximal Rectangle
// 106119100 Rajneesh Pandey

#include <bits/stdc++.h>
using namespace std;

int maxRectangle(vector<vector<char>> &matrix)
{
    if (matrix.empty())
        return 0;
    int m = matrix.size();
    int n = matrix[0].size();
    int left[n], right[n], height[n];
    //initializing the arrays
    fill_n(left, n, 0);
    fill_n(right, n, n);
    fill_n(height, n, 0);
    int maxArea = 0;
    for (int i = 0; i < m; i++)
    {
        int cur_left = 0, cur_right = n;
        for (int j = 0; j < n; j++)
        { // compute height (can do this from either side)
            if (matrix[i][j] == '1')
                height[j]++;
            else
                height[j] = 0;
        }
        for (int j = 0; j < n; j++)
        { // compute left (from left to right)
            if (matrix[i][j] == '1')
                left[j] = max(left[j], cur_left);
            else
            {
                left[j] = 0;
                cur_left = j + 1;
            }
        }
        // compute right (from right to left)
        for (int j = n - 1; j >= 0; j--)
        {
            if (matrix[i][j] == '1')
                right[j] = min(right[j], cur_right);
            else
            {
                right[j] = n;
                cur_right = j;
            }
        }
```

```
// compute the area of rectangle (can do this from either side)
        for (int j = 0; j < n; j++)
            maxArea = max(maxArea, (right[j] - left[j]) * height[j]);
    }
    return maxArea;
}

int main()
{

    int n, m;
    cout << "----------------------------" << endl;
    cout << "Enter the Dimensions of the matrix : " << endl;
    cin >> n >> m;
    vector<vector<char>> matrix(n, vector<char>(m));
    cout << "Enter the Values in the matrix : " << endl;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
        {
            cin >> matrix[i][j];
        }
    cout << "Maximum area of Rectangle is : " << endl;
    cout << maxRectangle(matrix) << endl;
    cout << "----------------------------" << endl;

    return 0;
}
```

Input / Output:

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Programs> ./Question2
----------------------------
Enter the Dimensions of the matrix :
6 5
Enter the Values in the matrix :
0 1 1 0 1
1 1 0 1 0
0 1 1 1 0
1 1 1 1 0
1 1 1 1 1
0 0 0 0 0
Maximum area of Rectangle is :
9
----------------------------
PS D:\Programs>
```

Question 3 :

3. **Fitting Shelves Problem:** Given length of wall w and shelves of two lengths m and n, find the number of each type of shelf to be used and the remaining empty space in the optimal solution so that the empty space is minimum. The larger of the two shelves is cheaper so it is preferred. However cost is secondary and first priority is to minimize empty space on wall. (Sample Input : w = 24 m = 3 n = 5, Output : 3 , 3 , 0 We use three units of both shelves and 0 space is left.)

   Note: The input will be given by the user at the time of execution for all the questions.

Approach

We will try all possible combinations of shelves that fit within the length of the wall.

To implement this approach along with the constraint that larger shelf costs less than the smaller one, starting from 0, we increase no of larger type shelves till they can be fit. For each case we calculate the empty space and finally store that value which minimizes the empty space. if empty space is same in two cases we prefer the one with more no of larger shelves

Code and Input / Output :

```cpp
// Question 3 : Fitting Shelves Problem
// 106119100 Rajneesh Pandey

#include <bits/stdc++.h>
using namespace std;

void minSpacePreferLarge(int wall, int m, int n)
{
    // for simplicity, Assuming m is always smaller than n
    // initializing output variables
    int num_m = 0, num_n = 0, min_empty = wall;

    // p and q are no of shelves of length m and n
    // rem is the empty space
    int p = wall / m, q = 0, rem = wall % m;
    num_m = p;
    num_n = q;
    min_empty = rem;
    while (wall >= n)
    {
        // place one more shelf of length n
        q += 1;
        wall = wall - n;
        // place as many shelves of length m
        // in the remaining part
        p = wall / m;
        rem = wall % m;

        // update output variablse if curr
        // min_empty <= overall empty
        if (rem <= min_empty)
        {
            num_m = p;
            num_n = q;
            min_empty = rem;
        }
    }
    cout << num_m << " " << num_n << " " << min_empty << endl;
}

// Driver code
int main()
{
    int wall, m, n;
    cout << "---------------------------------" << endl;
    cout << "Enter the value of wall(w) : " << endl;
    cin >> wall;
    cout << "Enter the value of m : " << endl;
    cin >> m;
```

```
    cout << "Enter the value of n : " << endl;
    cin >> n;
    cout << "Number of each type of shelf to be used and the remaining
 empty space : " << endl;
    minSpacePreferLarge(wall, m, n);
    cout << "----------------------------------------" << endl;
    return 0;
}
```

Input/Output:

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Programs> ./Question3
--------------------------------
Enter the value of wall(w) :
24
Enter the value of m :
3
Enter the value of n :
5
Number of each type of shelf to be used and the remaining empty space :
3 3 0
----------------------------------------
PS D:\Programs>
```