Ad hoc Problems

Introduction

Ad hoc means for one specific cause or approaching a solution in an unplanned way. In programming, it means working in bits and pieces. For example, there are 2 programmers 1 and 2. They have to complete 4 modules, say W, X, Y ,Z. If there is a plan then they can decide on which module to work on, but in an ad hoc approach they can work in an unplanned manner.

Live Problem 1: Equalize

Link : https://codeforces.com/contest/1037/problem/C

Problem Statement:

You are given two binary strings **a** and **b** of the same length. You can perform the following two operations on the string a:

Swap any two bits at indices i and j respectively ($1 \le i,j \le n$), the cost of this operation is $|i-j|$, that is, the absolute difference between i and j.

Select any arbitrary index i ($1 \le i \le n$) and flip (change 0 to 1 or 1 to 0) the bit at this index. The cost of this operation is 1.

Find the minimum cost to make the string a equal to b. It is not allowed to modify string b.

Explanation:

Agenda: Find the minimum cost of converting string a to string b.

- Cost of flipping a bit: **O(1)**
- Cost of swapping bits from at i and j: **O(|i-j|)**

A = 1 0 0 and B = 0 0 1, if we swap the first and last bit of A that makes A equal to B with a cost of 2 - 0 = **2**. If we had flipped the first and last bits, then our cost would have been 2.

But considering **S = 1 0 1 0 and T = 1 1 0 1**, if **S[i] == T[i]**, then we do nothing at those bits. Now, flipping costs us only 1 and swapping to longer positions will cost more. For example, if for any string we have to change the bits at position 0 and 3, by flipping the two, our cost will only be **2** but if we use a swap operation, then our cost will be **3 - 0 = 3**.

For positions next to each other, the swap operation is a better choice because it will cost us **(i + 1) - i = 1**, but if we flip the individual bits then our cost will be **1 + 1 = 2**. So we swap only with the next element, that is if **S[i + 1] != T[i + 1] && S[i + 1] != S[i]**, then we swap.

Code:

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


int main(){

    int n;
    cin >> n;
    string s, t;
    cin >>s >> t ;
    int cost =0;
    for(int i=0; i<n; ){
        if(s[i] != t[i]){
            //  either swap or flip and increment the cost
            if(i+1 < n && s[i] != s[i+1] && s[i+1] != t[i+1]){
                i+=2;
            } else {
                i+=1;
            }
            cost++;

        } else {
            i++;
```

```
        }
    }
    cout << cost;
    return 0;
}
```

Live Problem 2: Rectangular Area

Problem Statement:

You are given N rectangles, which are centered in the center of the Cartesian coordinate system and their sides are parallel to the coordinate axes. Each rectangle is uniquely identified with its width (along the x-axis) and height (along the y-axis). Navdeep has coloured each rectangle in a certain colour and now wants to know the area of the coloured part of the paper. Please refer to the sample test case 1 and image used in it for better understanding.

Explanation:

Sample Input 1 :

3

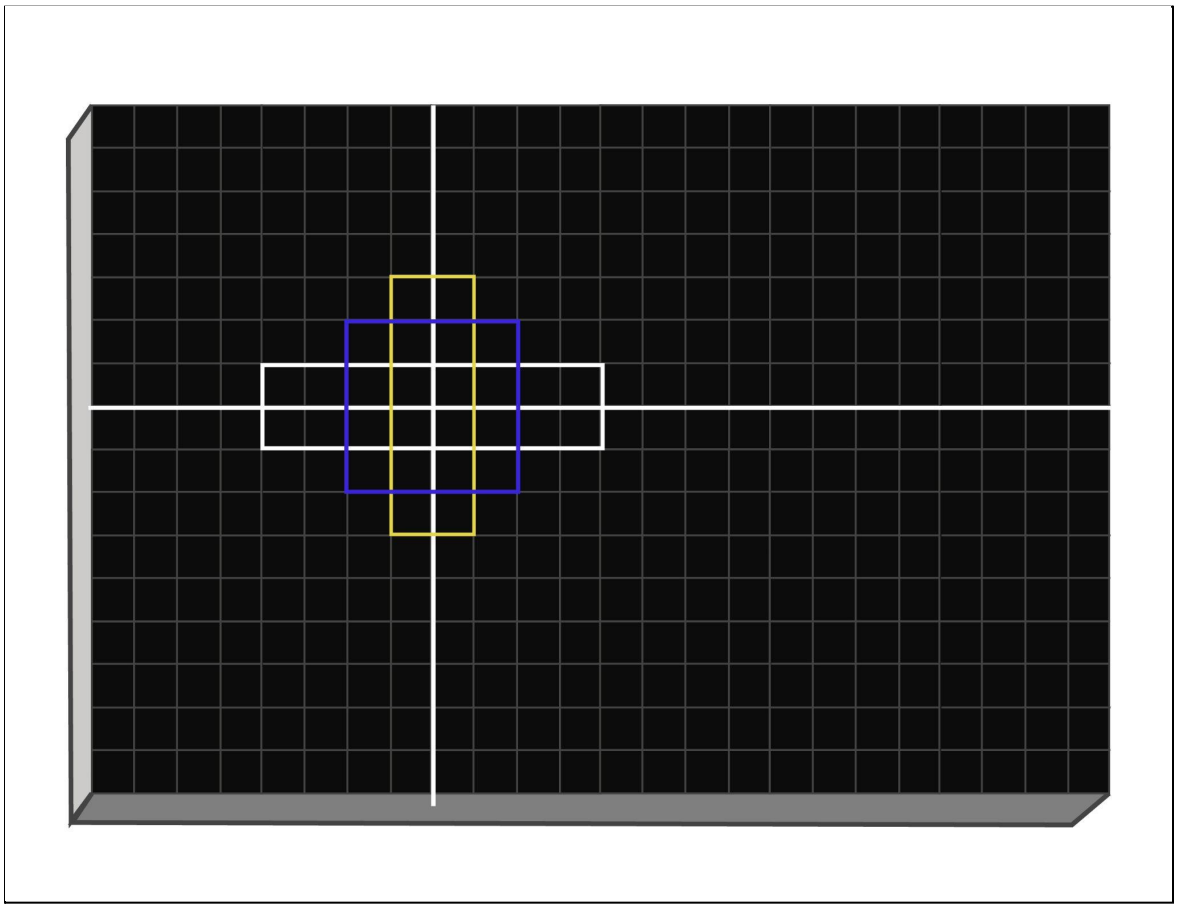
8 2 (represented by white coloured rectangle)

4 4 (represented by blue coloured rectangle)

2 6 (represented by yellow coloured rectangle)

Sample Output 1 :

28



In other words, he wants to know the number of unit squares that belong to at least one rectangle. This can also be seen in the image. There are 28 unit squares, which come in the bounds of at least one rectangle.

We need not calculate the area of the entire figure, rather we start from the rightmost part of the figure and start calculating the area according to the height. Once we reach the origin we can multiply our answer by 2 to calculate the area of the entire figure.

Steps:

1. Create a vector that stores the heights.
2. Start from the rightmost point and iterate towards the left one unit at a time till you reach the origin.

3. If height of the left point/ unit is less than the previous right one, then height[i] = height[i+1]

Code:

```
/*
    Time complexity: O(N + M)
    Space complexity: O(M)
    where N is the number of rectangles
    and M is the range of X coordinates of the rectangle
*/
#include <bits/stdc++.h>
using namespace std;
long long int rectArea(pair<int, int> arr[], int n) {
    int height[100002];
    for (int i = 0; i < 100002; i++) {
        height[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        height[arr[i].first / 2] = max(height[arr[i].first / 2],arr[i].second);
    }
    long long int ans = 0;
    for (int i = 100000; i > 0; i--) {
        height[i] = max(height[i], height[i + 1]);
        ans += height[i];
    }
    return ans * 2;
}
int main() {
    int n;
    cin >> n;
    int a, b;
    pair<int, int> arr[n];
    for (int i = 0; i < n; i++) {
        cin >> a >> b;
        arr[i] = make_pair(a, b);
    }
    cout << rectArea(arr, n) << endl;
}
```