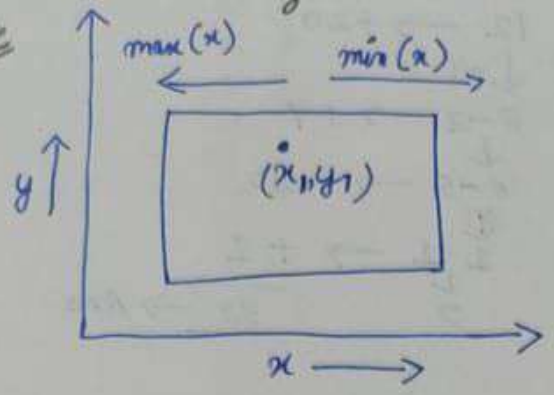
Contest Discussion

Snuke's coloring

2.



Given : $x > x_i = 2$ (Let Green color)
$x < x_i = 1$ (Let Yellow ")
$y < y_i = 3$
$y > y_i = 4$

Approach:-

1. start with full white rectangles
2. For each operation (x_i, y_i, a_i) :
   - → if $a_i = 1$: paint left → update $x\_min = \max(x\_min, x_i)$
   - → if $a_i = 2$: $x\_max = \min(x\_max, x_i)$
   - → if $a_i = 3$: $y\_min = \min(y\_min, y_i)$
   - → if $a_i = 4$: $y\_max = \max(y\_max, y_i)$
3. Final area = $\max(0, x_{max} - x_{min}) * \max(0, y\_max - y\_min)$

code:-

```
void solve(){
  int x, int y;
  cin >> x >> y;
  int n;
  cin >> n;
  int left=0, right=x, up=y, down=0;
  for (int i=0; i<n; i++){
    int xx, yy;
    cin >> xx >> yy;
    int a;
    cin >> a;
    if (a==1){
      left = max(left, );
    }
    else if (a==2){
      right = min(right, xx);
    }
    else if (a==3){
      down = max(down, yy);
    }
    else{
      up = min(up, yy);
    }
  cout << left << " " << right << endl << up << " " << down << endl;
  if (left > right || up < down){
    cout << 0 << endl;
    return; }
```

cout << (right - left) * (up - down);
}

1. Candy Distribution

Approach 1: Using a for loop. T.C ⇒ O(N)

For every child, check the neighbouring values & inc count based on cond^n

Approach 2: Using Formula $\frac{n(n+1)}{2}$ T.C ⇒ O(1)

If the distribution follows inc. or dec. sequence then sum = $\frac{n(n+1)}{2}$

3. Equation Solver

Find the no. of real sol^n for : $ax^2 + bx + c = 0$

$D = b^2 - 4ac$

If a == 0 → 2 sol^n (sol^n become linear)
↳ output 1, in this case.

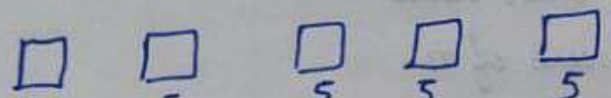
If D > 0 → 2 sol^n

If D = 0 → 1 sol^n

If D < 0 → No sol^n

4. Colour it

Rule : No two adjacent blocks can have the same colour.



1st block : K choices

Each subsequent block : K-1 choices

∴ Total Ways = $K \times (K-1)^{N-1}$.

5. Weird sum

Given K → subarray length, l → starting index, ~~n~~ → No. of elements

Approach :-

K = 4, l = 2, n = 7

Brute

1 2 3 4 5 6 7 8 9 10

2+3+4+5

3+4+5+6

4+5+6+7

We can do this using for loops.

T.C : O(n²)

Better Approach :-

1. Precompute Prefix sum.
2. For each valid array calculate the sum using prefix sum.

	①	②	③	④
P	(1)	(1+2)	(1+2+3)	(1+2+3+4)
PP	(1)	(1) + (1+2)	(1) + (1+2) + (1+2+3)	(1) + (1+2) + (1+2+3) + (1+2+3+4)

Pattern Analysis :

Inc. Pattern → 1 ---- K

Stagnant → K - --- K+1

Decreasing → K+1 ---- N-1

T.C : O(N)

6. Time Complexity

Approach :-

1. Parse the ~~nesting~~ nesting of for loops from code or string.
2. Use a variable depth to track current nest level
3. Use a map to keep track of the powers of n.

```
map<int,int>mp;
for(int i=0;i<s.size();i++)
{
    if(s[i]=='f')
    {
        flag=0;
        depth++;
        i+=2;
    }
    else if(s[i]=='e')
    {
        if(depth==0)
        {
            cout<<"Compile Error\n";
            return;
        }
        if(flag==0)
        {
            mp[depth]++;
            flag=1;
            depth--;
        }
        else
        {
            depth--;
        }
        i+=5;
    }
}
if(depth>0)
{
    cout<<"Compile Error\n";
    return;
}
for(auto i:mp)
{
    cout<<i.first<<" "<<i.second<<endl;
}
}
```

7. Best Train connection

1. Convert all train times to min.
2. Use two nested for loops & check connection.