

# **International Coding Hub Labor Day 2020**

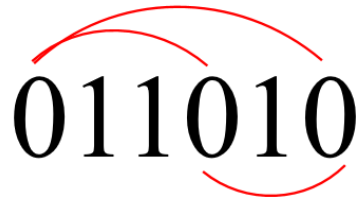
## **Editorial**

ICH ADMINISTRATION

September 6, 2020

## §1 Binary Substring

A valid substring is defined by having zeroes as its leftmost and rightmost characters. Hence the number of valid substrings is equal to the number of ways to choose those two zeroes from our pool of  $m$  zeroes in the binary string:  $\binom{m}{2} = \frac{m(m+1)}{2} - m$


  
 011010

```

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

int main(){
    string binary;
    cin>>binary;
    long long m = 0;
    for (int i = 0; i<binary.length(); i++){
        if (binary[i] == '0')
            m++;
    }
    cout<<(m*(m+1)/2) - m<<endl;
    return 0;
}
  
```

Complexity:  $O(n)$

## §2 String Comparison

We need to find the length of smallest substring of  $n$  which contains all unique characters of  $m$ . To do this, we can brute force every substring of  $n$ , and check whether that substring contains all characters of  $m$ . If so, we can check if we need to update our minimum substring length. To keep the unique characters of  $m$ , and each substring of  $n$ , we can use two sets and compare them each time we choose a substring of  $n$  (**intersect**).

```

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

bool intersect(set<char> a, set<char> b){
    //check if everything in a is in b
    for(char c:a) if(!(b.count(c))) return 0;
    return 1;
}

int main(){
    string n,m; cin>>n>>m;
    set<char> M;
    for(char c:m) M.insert(c);
    int mn=401;
    for(int i=0; i<n.length(); i++){
        set<char> s;
        for(int j=i; j<n.length(); j++){
            s.insert(n[j]);
            if(intersect(M, s)){
                mn=min(mn, j-i+1); break;
            }
        }
    }
    if(mn==401) cout<<"N/A"<<endl;
    else cout<<mn<<endl;

    return 0;
}
    
```

Complexity: Roughly  $O(|n|^2|m|\log|n|)$  where  $|n|$  is the length of string  $n$  and  $|m|$  is the length of string  $m$ .

### §3 Powerful Subarray

The main observation is that if value  $x$  has maximum occurrences in the optimal subarray  $s$ , the subarray should have  $x$  as both its leftmost and rightmost values, those being the only two occurrences of  $x$  in  $s$ .

Proof: We know that there must be one value with at least 2 occurrences in  $s$ , for  $s$  to be powerful. Hence, the optimal (shortest) subarray should have exactly 2 occurrences of a value, and 1 occurrence of all other values (if there are any). To construct this subarray, if there are 2 locations where the same value occurs, the subarray can have those locations as its left and rightmost positions, to minimize the number of other elements with occurrences of 1.

Implementation/solution: we can sweep through the array to find the minimum distance between any two identical values. For each value in the array  $a$ , we can store the most recent occurrence of it (`occ`).

```

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

int main(){
    int n; cin>>n;
    int a[n]; int occ[n+1];
    for(int i=0; i<n; i++) cin>>a[i];
    if(n==1) cout<<-1<<endl;
    else{
        fill(occ, occ+n+1, -1);
        int mn=2e5+1;
        for(int i=0; i<n; i++){
            if(occ[a[i]]!=-1){
                mn=min(mn, i-occ[a[i]]); occ[a[i]]=i;
            }
            else occ[a[i]]=i;
        }
        if(mn==2e5+1) cout<<-1<<endl;
        else cout<<mn+1<<endl;
    }
    return 0;
}
    
```

Complexity:  $O(n)$

## §4 Mouse

Since  $n, m \leq 10^3$ , we cannot recursively brute force all paths. Instead, we can use dynamic programming with  $dp[i][j]$  being the number of valid paths to cell  $[i][j]$ , and  $dp[n][m]$  being the answer. So the dp state is  $dp[i][j]=dp[i-1][j]+dp[i][j-1]$  (for  $i, j > 1$ ). In other words, the number of paths to cell  $[i][j]$  is the number of paths to the cell directly above + number of paths to the cell directly to the left.

	1	2	3	4
1				
2				
3				
4				

1	1	1	1
1	0	0	1
1	1	1	2
0	1	2	4

Above is the dp table for the sample input. The number of paths to cells with mousetraps is 0, as no valid path can include a mousetrap.

```

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

const int nmmax=1e3; const int MOD=1e9+7;
bool grid[nmmax][nmmax]; int n,m,k;
int dp[nmmax][nmmax];
int main(){
    cin>>n>>m>>k;
    for(int i=0; i<k; i++){
        int u,v; cin>>u>>v;
        grid[u-1][v-1]=1;
    }
    if(!grid[0][0]) dp[0][0]=1;
    for(int i=1; i<n; i++)
        if(!grid[i][0])
            dp[i][0]=dp[i-1][0];
    for(int i=1; i<m; i++)
        if(!grid[0][i])
            dp[0][i]=dp[0][i-1];
    for(int i=1; i<n; i++)
        for(int j=1; j<m; j++)
            if(!grid[i][j])
                dp[i][j]=(dp[i-1][j]+dp[i][j-1])%MOD;
    cout<<dp[n-1][m-1]<<endl;
    return 0;
}
    
```

Complexity:  $O(nm)$

## §5 Teams

We can sort by  $x$  coordinate and sweep across using a left and right pointer to find the minimum valid size. We start with  $\text{left}=\text{right}=\text{leftmost room location}$ . When a team ID is not present within our window, we increase the right pointer. When all team ID's are present within our window, we update the minimum window size, and increase the left pointer.

```

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

struct room{
    int x; int id;
    bool operator<(room const& other) {
        return x < other.x;
    }
};

int main(){
    int n; cin>>n; room hotel[n];
    for(int i=0; i<n; i++){
        cin>>hotel[i].x>>hotel[i].id;
    }
    sort(hotel, hotel+n);
    int mn=hotel[n-1].x-hotel[0].x;
    int l=0; int r=0;
    set<int> cid;//unique id's in the current window
    map<int,int> idcnt;//number of each id in current window
    idcnt[hotel[0].id]=1;
    while(r<n){
        if(cid.size()<n){
            r++;
            if(r<n){
                cid.insert(hotel[r].id);
                idcnt[hotel[r].id]++;
            }
        }
        else{
            idcnt[hotel[l].id]--;
            if(idcnt[hotel[l].id]==0)
                cid.erase(hotel[l].id);
            mn=min(mn, hotel[r].x-hotel[l].x);
            l++;
        }
    }
    cout<<mn<<endl;
    return 0;
}
    
```

Complexity:  $O(n \log n)$

Source: USACO 2011 November: Cow Lineup (Silver)