Sep 10, 2025

DOUBT SESSION

Ques 1. Valid Commands

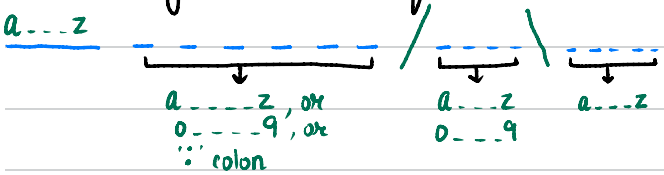
https://maang.in/contests/attempts/56620?problem_id=13183

You have developed a program in a new scripting language. Of course, it requires accurate parsing in order to perform as expected, and it is very cryptic. You wand to determine how many can be made out of your lines of script. To do this, you count all of the substrings that make up a valid command. Each of the valid commands will be in the following format:

- The first letter is a lowercase English letter.
- Next, it contains a sequence of zero or more of the follwing characters: lowercase English letters, digits, and colons.
- Next, it contains a forward slash, '/'.
- Next, it contains a sequence of one or more of the follwing characters: lowercase English letters, and digits.
- Next, it contains a backward slash, '\'.
- Next, it contains a sequence of one or more of the follwing characters: lowercase English letters.

Note: Two substrings, $S[i[1]...j[1]]$ and $S[i[2]...j[2]]$, are said to be distinct if either $i[1] \neq i[2]$ or $j[1] \neq j[2]$.

Composition of valid string according to the given in the ques will be:



Use of two pointer

a / a \ a a b

↑ ↑

If my first character is a lowercase english alphabet then we will fix the first pointer there and use a second pointer and put it on the next char.

We will keep traversing the string using the second pointer as long as our conditions are met (a-z, 0-9, :)

Example:

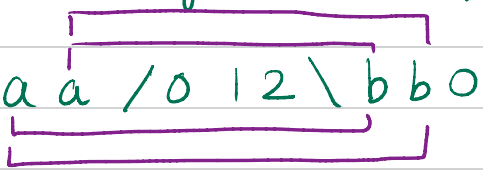
a a : b c c 0 :

↑ ↑

while(s[i] is lower or digit or colon) continue; else check if its a forward slash

The second pointer here will go on till the last char because this is a valid input but if we have a '\' then the string instantly becomes invalid.

Valid substrings in this example:



Ques 2. Robot Vs You

You are playing a game with a robot. The game consists of *N* square cells (*1 x N* table) and *K* boxes of length *A* (one box covers a sequence of consecutive *A* cells). Initially, the robot placed *K* boxes in the square cells, such that **no two boxes intersect or touch each other**.

After that you make a sequence of guesses, in each guess you guess a square cell, and the robot tells whether that cell is covered by a box or it is an empty square cell.

Due to some bug, there was some fault in the robot and **it always tells that the square is empty**. You have to find the **first guess** after which you can be sure that the robot is faulty.

Example: N=6, K=3, A=1

No. of Guesses – 1 2 4

Here,

We have six empty spaces

_ _ _ _ _ _

3 boxes each of size 1

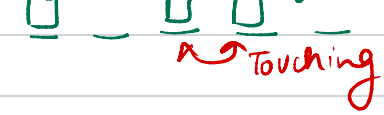
[1] [2] [3]

Now we have put these 3 boxes of size 1 into the 6 empty space such that they don't intersect each other.

One possible arrangement:

Yes

Is this a valid arrangement?



No

If we have n=12, a=3 then how many boxes can we put:

Given n & a, what is the generalized formula to figure out how many boxes can we put into this configuration without it intersecting

$$MaxK = \frac{n+1}{a+1}$$

Ans

For n=12, a=3, $MaxK = \frac{13}{4} = 3$

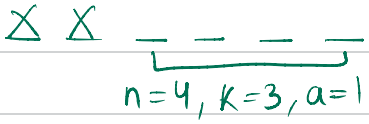
Back to the previous first example of n=6, K=3, a=1
Now, we have also been given indexes which are blocked i.e you cant place the box on it.

→ 1

Can we put K=3 with n=5 without boxes touching each other?

$MaxK = \frac{5+1}{1+1} = \frac{6}{2} = 3$ YES

→ index=2



$MaxK = \frac{4+1}{1+1} = \frac{5}{2} = 2$ No, return.

Given : [1 - - - n] Range
we can know the max Box we can put.
As indexes get blocked, we can create separate ranges.

[1 - - - x-1] [x+1 - - - n]

'a' no of box + 'b' no. of box

If a+b ≥ K, move to next index else Return.

Min Max

Set<pair<int, int>> {1, 10}

Break at 5

lowerbound {5, 0}

Insert {1, 4} {6, 10}

Break at 6

lowerbound {6, 0}

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll boxes_fit(ll len, ll A) {
    if (len < A) return 0;
    return (len + 1) / (A + 1);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int T;
    if (!(cin >> T)) return 0;
    while (T--) {
        ll N, K, A;
        cin >> N >> K >> A;
        int M;
        cin >> M;
        vector<ll> x(M);
        for (int i = 0; i < M; ++i) cin >> x[i];

        // set of free intervals [l, r], sorted by l
        set<pair<ll,ll>> intervals;
        intervals.insert({1, N});
```

```cpp
// set of free intervals [l, r], sorted by l
set<pair<ll,ll>> intervals;
intervals.insert({1, N});

ll total = boxes_fit(N, A);
if (total < K) {
    cout << -1 << '\n';
    continue;
}

int answer = -1;
for (int i = 0; i < M; ++i) {
    ll xi = x[i];

    // find interval that might contain xi
    // find first interval with start > xi
    auto it = intervals.upper_bound({xi, (ll)9e18});
    if (it == intervals.begin()) {
        // no interval starts <= xi, so xi not inside any interval
        // nothing changes
    } else {
        --it; // candidate interval
        ll L = it->first, R = it->second;
        if (L <= xi && xi <= R) {
            // remove current contribution
            ll len = R - L + 1;
            total -= boxes_fit(len, A);

            // erase the interval and insert left and right pieces (if any)
            intervals.erase(it);
            if (L <= xi-1) {
                intervals.insert({L, xi-1});
                total += boxes_fit((xi-1)-L+1, A);
            }
            if (xi+1 <= R) {
                intervals.insert({xi+1, R});
                total += boxes_fit(R-(xi+1)+1, A);
            }
        }

        if (total < K) {
            answer = i+1; // 1-based index of guess
            cout << answer << '\n';
            break;
        }
    }
    if (answer == -1) cout << -1 << '\n';
}
return 0;
}
```

Approach is already discussed.
Here is code :

```cpp
void helper(int n, int k)
{
    //base case
    // cout << n << " " << k << endl;
    if (len[n] < k)
    {
        cout << ".";
        return;
    }
    if (n == 0)
    {
        cout << st[k];
        return;
    }

    //ith string = prev + (i-1)string + nextt + (i-1)thstring + lst

    //lyinh in st part
    if (k < prevv.size())
    {
        cout << prevv[k];
        return;
    }
    k -= prevv.size();

    //lying in (i-1)th part (first)
    if (k < len[n - 1])
    {
        helper(n - 1, k);
        return;
    }
    k -= len[n - 1];

    if (k < nextt.size())
    {
        cout << nextt[k];
        return;
    }
    k -= nextt.size();

    //lying in (i-1)th part (second)
    if (k < len[n - 1])
    {
        helper(n - 1, k);
        return;
    }
    k -= len[n - 1];
    if(k>1)cout<<".";
    else cout << lst[k];
    return;
}
void solve()
{
    int n, k;
    cin >> n >> k;
    helper(n, k - 1);
}
```

Ques 3. Nephren gives a riddle

https://codeforces.com/problemset/problem/896/A

Nephren is playing a game with little leprechauns.

She gives them an infinite array of strings, $f_{0\ldots\infty}$.

f_0 is "What are you doing at the end of the world? Are you busy? Will you save us?".

She wants to let more people know about it, so she defines f_i = "What are you doing while sending "f_{i-1}"? Are you busy? Will you send "f_{i-1}"?" for all $i \geq 1$.

For example, f_1 is

"What are you doing while sending "What are you doing at the end of the world? Are you busy? Will you save us?"? Are you busy? Will you send "What are you doing at the end of the world? Are you busy? Will you save us?"?". Note that the quotes in the very beginning and in the very end are for clarity and are not a part of f_1.

It can be seen that the characters in f_i are letters, question marks, (possibly) quotation marks and spaces.

Nephren will ask the little leprechauns q times. Each time she will let them find the k-th character of f_n. The characters are indexed starting from 1. If f_n consists of less than k characters, output '.' (without quotes).

Can you answer her queries?

AlgoZenith