International Open Programming Contest, Editorial

IIT Kanpur

March 25, 2014

Summary

It was a great contest. It had 20 problems. 2 of them remained unsolved. Highest ranked team solved 16 problems. So we feel it was a balanced problem set.

Problem A: Fun with Flooring Factorial

Difficulty

Easy, requires understanding of multiplying two long number modulo m.

Problem

Find whether $\lfloor (n!/b) \rfloor$ is even or not ?

Solution

Answer is yes if $n!$ % $2b$ is $< b$. Otherwise it is always No.

Let us just try to take the case when b = 2, then as $\lfloor (2/2) \rfloor = 1$ and $\lfloor (1/2) \rfloor = 0$ and $\lfloor (4/2) \rfloor = 2$. Now calculate 2 % 4 = 2, 1 % 4 = 1, 4 % 4 = 0, So for 1st case answer is No, Yes and No and condition above stated is true.

Proof for general case:
Let us for now consider $a = n!$.
Let us assume $a = x$ * $b + k$, where $0 \leq k \leq b$.
If x is even our answer is Yes otherwise No.
Now consider the case when x is even. Write $a = (x/2)$ * $2b + k$. So this means a % will give k which lies between $[0, b)$.
Now consider the case when x is odd. Write $a = ((x-1)/2)$ * $2b + (k+b)$. So this means a % $2b$ will give k + b which lies between $[b, 2b)$.

Now You problem reduces to find $n!$ % $2b$, as b is fit in long (Java), long long in (C++). For finding $n!$ % $2b$, you will need to multiply two long number modulo long numbers. If you multiply both of them directly, then our product will overflow. You can use bigInteger (Java) for this purpose. But for this problem, it will be a TLE for our problem.

Now there are two ways for multiplying for two long number a, b modulo a long number m. For this you can use the usual method used in doing exponentiation of $a^b \% m$, You just have to use addition instead of multiplication. Write m in binary representation. For understanding the complete process, please look into the my code and also look for modular exponentiation.

Also read comment of yeputons on QPOLYSUM codechef editorial to get a $\mathbf{O}(1)$ solution for multiplying for two long number a, b modulo a long number m.

Implementation

```java
/* author: Praveen Dhinwa */
import java.io.*;
import java.util.InputMismatchException;
import java.util.Scanner;

public class Main {
    public static void main (String[] args) {
        Scanner in = new Scanner (System.in);
        int testCase = in.nextInt();
        while (testCase -- > 0) {
            long a = in.nextLong();
            long b = in.nextLong();
            long ans = solve(a, 2 * b);
            if (ans >= 0 && ans < b) System.out.println ("Even");
            else System.out.println ("Odd");
        }
    }

    private static long solve(long a, long b) {
        long res = 1;
        for (int i = 1; i <= (int) a; i++) {
            res = multiply (res, i, b);
        }
        return res;
    }

    private static long multiply(long a, int b, long mod) {
        long res = 0;
        while (b > 0) {
         // (b & 1) > 0, means whether last bit is 1 or not, ie odd or not. If odd, then if will be executed.
         // if we come to odd bit, we will add our current a to answer.
            if ((b & 1) > 0) {
                res = (res + a);
                if (res >= mod)
                    res -= mod;
            }
            // double our current a.
            a = (a + a);
            if (a >= mod)
                a -= mod;
            //  divide b by 2.
            b >>= 1;
        }
        return res;
        // take example execution of algorithm.
        // a = 2, b = 5, mod = 3.
        // b = 101 in binary.
        // res = 0 and a = 2 initially
        // first digit of b is 1, (From right to left), So we will execute the if loop, so res = 0 + 2 = 2.
        // execute a = a + a, so a becomes 4.
        // second digit of b is 0, (From right to left), So we will not execute the if loop.
        // execute a = a + a, so a becomes 8.
        // third digit of b is 1, (From right to left), So we will execute the if loop, so res = 2 + 8 = 10.
        // execute a = a + a, so a becomes 16.

         // Now all you have to take care of modulos in your calculation. so final answer will be 10 % 3 = 1.
```

```
    }
}
```

Another Solution

Use dp :)

Implementation

```
/*
Algo :: DP
Acc
Time :: 1.3 sec
*/
#include<stdio.h>
#include<assert.h>
typedef long long int ll;
ll q,r;
void powmod(ll R,ll a,ll b)
{
   ll q1=0,r1=0;
   while(a)
      if(a&1){
         a--;
         r1=r1+R;
         if(r1>=b){
            r1=r1-b;
            q1=q1+1;
         }
      }
      else{
         a>>=1;
         R=R+R;
         if(R >= b){
            R=R-b;
            q1=q1+a;
         }
      }
   q=q1;
   r=r1;
}
int main()
{
   int test,curr,prev;
   ll state[2][2],mod,a,b;
   ll z=1e9;
   z=(ll)z*(ll)z;
   scanf("%d",&test);
   assert(test<=100);
   while(scanf("%lld %lld",&a,&b)!=EOF){
      test--;
      curr=1;
      //scanf("%lld %lld",&a,&b);
      assert(a<=100000 && b<=z);
      mod=b%2;
      if((a==1 && b==1) || (a==0 && b==1) ){
```

```
        printf("Odd\n");
        continue;
      }
      state[0][0]=0;
      state[0][1]=1;  //remainder is 1
      state[1][0]=0;
      state[1][1]=0;
      for(ll i=2;i<=a;i++){
        prev=curr^1;
        powmod(state[prev][1],i,b);
        state[curr][0]=(state[prev][0]*i+q)&1;
        state[curr][1]=r;
        curr=curr^1;
      }
      prev=curr^1;
      if(state[prev][0]==0) printf("Even\n");
      else printf("Odd\n");
    }
    assert(test==0);
}
```

B. Add and Compute Path

Difficulty

Medium-Hard, requires Heavy Light Decomposition , Mathematics.

Problem

Given a Tree , you need to mantain the tree under two operations. a) Add s to subtree rooted at v b) Compute the sum of the values stored in the node from X to Y .

Solution

For each update : U v s , add (s, depth(v) * s) in the node v , let us call the first term as sum and the second term as dsum (as it is been multiplied by depth of that node)
Note: depth of a node is the number of edges from root to that node. depth array can be precomputed using a DFS.
Let Suppose we want to find sum of the values of the nodes from a given node(say X) to the root of the tree:
We will find the sum of both pair from X to the root (You can find this by using Heavy Light Decomposition), let suppose we got the sum and we stored it in variable A, then sum from X to root is given by Sum = ((depth[X]+1)*(A.sum) - A.dsum)

Now Let's get back to answering our query in a path from X to Y. Let anc be the ancestor of X and Y.
Vertices which are the descendent of either X or Y will not contribute in the answer . Vertices between X and Y and the ancestor of anc will contribute in the answer.

Let us first deal with vertices from X to anc :
A = HL_Query(X,anc); //This function return the sum of both pairs from X to anc.
S1 = (depth[X]+1)*(A.sum) - A.dsum) //S1 contains the required sum from X to anc.

Let us now calculate for vertices from Y to anc:
B= HL_Query(Y,anc); //This function return the sum of both pairs from Y to anc.
S2 = (depth[Y]+1)*(B.sum) - B.dsum) //S2 contains the required sum from Y to anc.

Let us now consider for vertices which are the ancestor of the anc:
C= HL_Query(anc,root); //This function return the sum of both pairs from anc to root.
S3 = (total_nodes_in_path * C.sum) = ((depth[X] + depth[Y] -2*depth[anc]+1)*C.sum)

Now you may think that the answer is S1 + S2 + S3 but wait , you have calculated the effect of adding to ancestor multiple times.But how much is the question ?
S4 = (nodes) * anc.sum + anc.sum //This is left as an exercise for the reader to evaluate how this is the extra value which is added
Now the Answer to the Query is S1 + S2 + S3 - S4
Try Out Yourself to figure it out how this works.

Problem C: Super Line Segments

Difficulty

Medium Hard, requires understanding of triangulation.

Problem

You are given n points in 2 D plane. Let us represent the n points by $p_1, p_2, ...p_n$. No two points have same coordinates and no three points are colinear.

Consider the set of lines **L** = L : L is a line segment made by connecting (p_i, p_j) for all $1 \leq$ i,j $\leq$ n and i $\neq$ j

Two line segments L_1 and L_2 are said to be related if they intersect at a point which is not an end point of either of segments.

A set of lines L is said to be a *super* set if it does not contain any two line segments which are related.

Now you wonder what could be the maximum possible size of a "super set" which belongs to **L**.

Solution

Let us reformulate the problem given n points in plane, You have to find number of non-intersecting lines in the plane using these points. First note that it is always beneficial to use triangulation instead of any other polygon. This can be proved using very simple arguements that you can take the polygon which is not triangle by again triangulating the polygon.

Now problem is reformulated in the problem to find out number of edges in the triangulation. First you will be amazed to know that answer to this is fixed on the number of total points and number of total points on hull only. Answer to given problem is $3 * (n - 1)$ - number of points on hull. For $n = 1$, answer is zero but if you use this formulla it comes -1, So this is a tricky case for you to handle.

Number of triangles in any triangulation are 2 * n - 2 - number of points on hull.

Read following link for complete proof:

- delaunay
- triangulation
- triangulations-of-surfaces-minimum-number-of-triangles

Note that the answer for problem can change if points become collinear. Though the formulla is also valid for collinear cases, but then you need to be careful making convex hull which would have made more harder than necessary.

Let us take examples to first understand why the answer is correct. For n = 2, answer is $3 * (2 - 1) - 2 = 1$. For n = 3, $3 * (3 - 1) - 3 = 3$

Complexity of the solution is $\mathbf{O}(nlogn)$ which will easily pass within the time.

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

typedef long long T;
const T EPS = 1e-6;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const { return make_pair(y,x) < make_pair(rhs.y,rhs.x); }
  bool operator==(const PT &rhs) const { return make_pair(y,x) == make_pair(rhs.y,rhs.x); }
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }

void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0) up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0) dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);
}

int main() {
   int t;
   cin >> t;
   int n;

   while(scanf("%d",&n)!=EOF){
      vector <PT> a;

      for (int i = 0; i < n; i++) {
         int x = 0, y = 0;
         scanf ("%d %d", &x, &y);
         a.push_back (PT (x, y));
      }

      if (n == 1) {
         cout << 0 << endl;
         continue;
      }

        // find hull of the points.
      ConvexHull (a);

      int hull = a.size();
      int ans = 3 * n - 3 - hull;
```

```
      cout << ans << endl;
   }

   return 0;
}
```

E. Easy Airthmetic Geometric Progression

Difficulty

Hard, requires DFS,Mathematics and DP.

Problem

Given a Tree where each node stores a value(initially they are all 0).
Each Tree has some value(denoted by R).
You are then given a continuous sequence of Updates and then a continuous sequence of Q Queries.
Update (X, A, B) : add series $(1 * X * (r^1), 2 * a * d * (r^2), 3 * a * d * (r^3), 4 * a * d * (r^4), 5 * a * d(r^5)..)$ from node A to B in the tree
Report (A,B) : Sum of the values stored in the node modulo 100711433(prime)

Solution

Observation : 100711433 is a prime
Let us first solve the problem for 1D array where we are given a query to add AGP (S,D,A,B) with start term as S and common difference as D from position A to position B.Idea is to consider Infinite AGp's.

One AGp with start term as S and common difference as D from A to B can be broken into 2 infinte AGp's as follows:
a) One AGp begining from A with start term as S and common difference as D
b) Another AGp begining from B+1 with start term as $-(S + (B - A + 1) * D) * R^{B-A+2}$ and common difference as $-D * R^{B-A+2}$

Approach for solving in 1D array
Store 2 elements for each entry in the array (sum and increment).
For each Update Query :
a) Add S*R to Array[A].sum and add D*R to Array[A].increment
b) Add $-(S + (B - A + 1) * D) * R^{B-A+3}$ to Array[B+1].sum and add $-D * R^{B-A+3}$ to Array[B+1].increment

After all the updates are performed, run a Dp from 1 to N (size of the array)

for i=2 to N :
Array[i].sum + = (Array[i-1].sum + Array[i-1].start)*R
Array[i].start + = Array[i-1].start*R

Now Array[i].sum will denote the value added in the Array at position i.
Try working out , i am sure you will be able to get this.

Once you have understood the 1D solution , now you can solve this problem easily.Here S= X , and D= X.
Let anc be the ancestor of A and B in the tree. For path from A to anc , solve it like 1D array which

we just did .For path from anc to B, use the concept of inverse of R(as given modulo is prime) and you can reduce it to 1D array easily.I would not disclose much now, and i hope the reader can complete it by himself.

F. Fibonacci Returns

Difficulty

Medium, requires Pisano Periods, CRT and adhoc mathematics.

Problem

Find $Fibo(nCr)$ mod m , where $m <= 1e7, n <= 1e5$ and $r <= 1e5$

Solution

First you need to find the pisano period or some multiple of pisano period . You may refer to this link which explains how to find pisano period.
Once you get the period , the next challenge is to compute nCr mod period.
m was choosen in such a way that period(say p) is always less than 1e9.

finding nCr modulo p:
Factorise $p = p_1^{e_1} * p_2^{e_2} * ..... * p_k^{e_k}$ where p_i is a prime number.
Note that $K < 10$ (as $2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 * 23 * 29 = 6469693230 > 1e9$)
Then you need to apply CRT(Chinese Remainder Theorom) on every prime factorisation to calculate nCr mod p.
nCr mod $p = CRT(nCr \text{ mod } p_1^{e_1}, nCr \text{ mod } p_2^{e_2}, ..., nCr \text{ mod } p_k^{e_k})$

Calculating nCr mod $p_i^{e_i}$
For every p_i, you need to compute factorials eliminating powers of p_i that divide the number, you can calculate the exponent of p_i separately.

G. Copy and Paste

Difficulty

Medium-HARD, requires segment trees.

Problem

Given an infinite array whose period is S(<= 1000) (say the array be A[]), and a zero initialised array B[] of size $N <= 1e5$.
You have to process 2 queries:
U x y M :: for (int $i = x; i <= y; i++$) $B[i] = A[M + i - x]$ //Assuming 1 indexed A,B
Q x y :: Find $MaxB[i] + B[i + 1] + ... + B[j]; xijy$.

Solution

First of all , you need to precompute an array of 2 dimensional which will store the best sum , best prefix sum , best suffix sum , sum for all indexes from a to b for A[].
Now suppose , if you are asked to find the best sum for an array which is repeated x times.If you have all 4 data's (i.e. best sum , best prefix sum , best suffix sum , sum) for the array , you can calculated the value of array repeated x times in log x time. I hope you can figure it out.
Now there will be log n intervals in the segment tree query , for each interval , you need to paste a particular interval from A[] , you can do this in log (size_of_interval / S) by following the above hint.

H. Catering Services

Difficulty

Hard, requires Max-Flow , Mathematics.

Problem

Given N apartments in the city, i-th apartment housing X[i] offices and and each office is charged P[i] rupees every day for catering at their lunch time, i.e., for time S[i] to E[i]. Each employee in the startup can cater to atmost one office at a time and one employee is enough to complete the catering needs of an office. An employee earns C rupees a day. Also, there are M one way roads in the city, each motorable by bike. The average speed of bike in Kanpur is 1km per V minutes.
Alice, a very close friend and business partner of Bob, advises him against catering to a specific apartment J.
Bob decided to cater to as many offices of Jth apartment, till his profit does not decrease.
Find the maximum profit bob can have?

Solution

First let us solve the problem which is formed after we remove alice constraints. That is we want to minimise the number of employees required to cater all apartments.
This reduced problem is similar to "Southwestern Europe Regional Contest 2013" which went unsolved in the contest. You are advised to first solve the problem B here http://swerc.dsic.upv.es/ProblemSet2013/statements.pdf . If you are unable to solve this , then please go through this editorial http://swerc.dsic.upv.es/ProblemSet2013/solutions.pdf .
Once you have solved the problem eliminating alice condition then you can solve the problem including alice condition by simple maths.
Let the minimum employee required excluding the apartment J be A1
Let the minimum employee required including all apartment be A2
Employee Required will be A1 if $P[J] < C$
Let a be the offices which will be served for apartment J .
Hence it follows that $A2 = A1 + X[J] - a$
Let devu be the profit in hiering A2 employee
and Let rachit be the profit in hiering A1 employee
$devu = rachit + X[J] * P[J] - C * (A2 - A1)$ // profit - loss
substituting we get $devu = rachit + X[J] * P[J] - C * (X[J] - a)$
$devu - rachit - X[J] * P[J] + C * X[J] = a * C$
$a = (devu - rachit - X[J] * P[J] + C * X[J])/C$
Answer will be $(rachit + a * P[J])$
If $P[J] >= C$ then it is obvious to serve all the offices of an apartment.

I.Joker and the Arkham Asylum

Difficulty

HARD, requires Segment Trees and Mathematics.

Problem

The Given Problem boils down to :
Given an empty array(named array[]) of size N.
There are 2 types of Queries
U X Y D : For each i from X to Y
$array[i] = array[i] + (i - X) * D + (i - X)^2 * D^2 + (i - X)^3 * D^3$
Q X Y : print Sum From $(array[X] + ... + array[Y])$ mod $(1e9 + 7)$

Solution

Let t = (i-X) be a variable
Expand $(x + t)^2 * d^2, (x + t)^3 * d^3, (x + t)^2 * 3xd^3$ and get this equation:
$td + t^2d^2 + t^3d^3 =$
$(-x^3 * d^3 + x^2d^2 - xd)$
$+ d(1 - 2xd + 3x^2d^2) * (x + td)$
$+ (d^2 - 3xd^3) * (x + td)^2$
$+ (d^3) * (x + td)^3$
NOTE :
$td + t^2d^2 + t^3d^3 = Z[0] + Z[1] * (x + td) + Z[2] * (x + td)^2 + Z[3] * (x + td)^3$
where
$Z[0] = (-x^3 * d^3 + x^2d^2 - xd)$
$Z[1] = d(1 - 2xd + 3x^2d^2)$
$Z[2] = (d^2 - 3xd^3)$
$Z[3] = (d^3)$
Now solve this simple equation using segment trees.

Problem J: Bomberman and the attack of Emperor Terrorin

Difficulty

Medium, Matching in a graph.

Problem

This problem requires knowledge of matching. It is a set of edges without common vertices in a graph. You can read more about matching here.

Solution

Let us solve this problem by some questions and their answers. Here, for (a,b) we consider a to be row number and b to be column number.
The question of whether it is possible or not to eliminate all enemies is simple to answer. Just check that each enemy can be eliminated or not. Now, we may assume that it is possible to eliminate all enemies.
What will happen when I message (a,b) to Bomberman?
Bomberman would plant 1 bomb at position (a,b) destroying enemy a on left column, b on top row, a on right column and b on bottom row. Then, bomberman would plant 2nd bomb at (b,a) destroying enemy b on left column, a on top row, b on right column and a on bottom row.
This means, that whenever (a,b) is sent in message, enemies at position a and b are destroyed in left column, right column, top row and bottom row. Note that, this also holds when b=a, i.e (a,a) is messaged.
That suggests, that we should just keep track of enemies at left column. All enemies are destroyed if and only if all enemies of left column are destroyed.
What will happen if we make a graph of N vertices, with an edge between u,v if bomb can be planted at maze[u][v] (with self loop if bomb can be planted at maze[u][u])?
Then all the messages will correspond to edges of this graph.
We would want to minimize the number of edges that dominate all vertices. Let P be a matching in the graph of size S. This matching would dominate 2*S vertices and other N-(2*S) edges would be required to dominate rest of the vertices.
In this case, the number of messages required to eliminate enemies would be S for vertices dominated by P and (N-2*S) for remaining. So, total messages required are N-S, which is minimized when S is maximized.
So, we need to find the size S of maximum matching and (N-S)*M would give the required answer.

```
1- Check for Impossible case, if true Report IMPOSSIBLE.
2- Else Find maximum matching on the reduced graph. Let it be S.
3- Report (N-S)*M.
```

Problem K: Three substrings

Difficulty

Medium, requires knowledge of dp and small optimization.

Problem

Given a string S of length N, consisting of lowercase English alphabets from $'a'$ to $'z'$, you have to select three non overlapping substrings of S, such that number of $a's$ in the string made by concatenating the three substrings are greater than or equal to M. Output answer modulo $(10^9 + 7)$.

Solution

Let us think how we can make three substrings in a string, We move from left to right, we maintain the position where we are ($index$ in the string s, amount of counts of A upto now in the substring ($cntA$), we also have to maintain number of segments we have upto now, we also need to know whether the last segment we made is still running $isSegmentRunning$.

For this problem you have to use 4 dimensional dynamic programming. Let us look its state, dp ($index$, $cntA$, $numberSegments$, $isSegmentRunning$).

To look into transition between the states, please have a look at the code, Please look at the second solution, it is slow solution, but you can easily understand the transition. But the solution uses memory of $\mathbf{O}(N * M * 4 * 2)$, As it is a huge memory, we have to reduce memory too. As our solution for $index$ depends only on $index - 1$, we can store the answer layer for $index - 1$, So we only need two layers making the memory space $\mathbf{O}(2 * M * 4 * 2)$.

Complexity of the solution is $\mathbf{O}(N * M * 4 * 2)$ which will easily pass within the time.

Implementation

```
/* @author: Praveen Dhinwa */
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {
    private static final long INF = (long) 1e9;
    private static final long mod = (long) 1e9 + 7;
    private static long[][][][] memo;
    private static int n;
    private static char s[];
    private static int requiredA;

    public static void main (String[] args) throws FileNotFoundException {
        Scanner sc = new Scanner (System.in);
        int testCase = sc.nextInt();
        while (testCase -- > 0) {
            String str = sc.next();
            requiredA = sc.nextInt();

            n = str.length();
```

```
        s = new char[n + 1];
        for (int i = 1; i <= n; i++) {
            s[i] = str.charAt (i - 1);
        }

        long ans2 = doItCorrectly(str, requiredA);
        System.out.println (ans2);
    }
}

private static long doItCorrectly(String str, int required) {
    int n = str.length();
    long dp[][][][] = new long[2][n + 3][4][2];

    for (int cntA = required; cntA <= n; cntA++) {
        dp[(n + 1) & 1][cntA][3][0] = 1;
    }

    for (int i = n; i >= 1; i--) {
        for (int cntA = n; cntA >= 0; cntA --) {
            for (int seg = 3; seg >= 0; seg --) {
                for (int isRunning = 0; isRunning < 2; isRunning ++) {
                    dp[(i & 1)][cntA][seg][isRunning] = 0;
                }
            }
        }

        for (int cntA = n; cntA >= 0; cntA --) {
            for (int seg = 3; seg >= 0; seg --) {
                for (int isRunning = 0; isRunning < 2; isRunning ++) {
                    long res = 0;
                    if (isRunning > 0) {
                        if (s[i] == 'a') {
                            // end the segment
                            res += dp[(i + 1) & 1][cntA + 1][seg][0];
                            if (res >= mod)
                                res -= mod;

                            // continue;
                            res += dp[(i + 1) & 1][cntA + 1][seg][1];
                            if (res >= mod)
                                res -= mod;
                        } else {
                            // end the segment
                            res += dp[(i + 1) & 1][cntA][seg][0];
                            if (res >= mod)
                                res -= mod;
                            // continue;
                            res += dp[(i + 1) & 1][cntA][seg][1];
                            if (res >= mod)
                                res -= mod;
                        }
                    } else {
                        // make a segment of just size equal to 1.
                        if (s[i] == 'a') {
                            if (seg + 1 <= 3) {
                                // begin and end the segment
```

```
                        res += dp[(i + 1) & 1][cntA + 1][seg + 1][0];
                        if (res >= mod)
                            res -= mod;
                    }
                } else {
                    if (seg + 1 <= 3) {
                        // begin and end the segment
                        res += dp[(i + 1) & 1][cntA][seg + 1][0];
                        if (res >= mod)
                            res -= mod;
                    }
                }

                // do the remaining thing.
                if (s[i] == 'a') {
                    if (seg + 1 <= 3) {
                        // begin the segment
                        res += dp[(i + 1) & 1][cntA + 1][seg + 1][1];
                        if (res >= mod)
                            res -= mod;
                    }
                    // continue;
                    res += dp[(i + 1) & 1][cntA][seg][0];
                    if (res >= mod)
                        res -= mod;
                } else {
                    if (seg + 1 <= 3) {
                        // begin the segment
                        res += dp[(i + 1) & 1][cntA][seg + 1][1];
                        if (res >= mod)
                            res -= mod;
                    }
                    // continue;
                    res += dp[(i + 1) & 1][cntA][seg][0];
                    if (res >= mod)
                        res -= mod;
                }
            }

            dp[i & 1][cntA][seg][isRunning] = res;
        }
    }
}

long ans = dp[1][0][0][0];
return ans;
}
}
```

Another code

```
// NOTE THAT we have provided this method to understand the idea better, You will get memory limit exceeded if you
// surprisely it passes in c++ which was not expected solution.
    private static long dp(int index, int cntA, int cntSegements, int isSegmentRunning) {
        if (memo[index][cntA][cntSegements][isSegmentRunning] != -1) {
            return memo[index][cntA][cntSegements][isSegmentRunning];
```

```
}

long res = 0;
if (index == n + 1) {
    //System.out.println (cntSegements + " " + cntA + " " + requiredA + " " + isSegmentRunning);
    if (cntSegements == 3 && cntA >= requiredA && isSegmentRunning == 0) {
        res = 1;
    } else {
        res = 0;
    }
} else {
    if (isSegmentRunning > 0) {
        if (s[index] == 'A') {
            // end the segment
            res += dp (index + 1, cntA + 1, cntSegements, 0);

            // continue;
            res += dp (index + 1, cntA + 1, cntSegements, 1);
        } else {
            // end the segment
            res += dp (index + 1, cntA, cntSegements, 0);

            // continue;
            res += dp (index + 1, cntA, cntSegements, 1);
        }
    } else {
        // make a segment of just size equal to 1.
        if (s[index] == 'A') {
            if (cntSegements + 1 <= 3) {
                // begin and end the segment
                res += dp (index + 1, cntA + 1, cntSegements + 1, 0);
            }
        } else {
            if (cntSegements + 1 <= 3) {
                // begin and end the segment
                res += dp (index + 1, cntA, cntSegements + 1, 0);
            }
        }

        // do the remaining thing.
        if (s[index] == 'A') {
            if (cntSegements + 1 <= 3) {
                // begin the segment
                res += dp (index + 1, cntA + 1, cntSegements + 1, 1);
            }
            // continue;
            res += dp (index + 1, cntA, cntSegements, 0);
        } else {
            if (cntSegements + 1 <= 3) {
                // begin the segment
                res += dp (index + 1, cntA, cntSegements + 1, 1);
            }
            // continue;
            res += dp (index + 1, cntA, cntSegements, 0);
        }
    }
}
```

```
        res %= mod;

        memo[index][cntA][cntSegements][isSegmentRunning] = res;
        return res;
    }
```

Problem L: Sweets Problem

Difficulty

Hard, requires understanding of dp, FFT or understanding of generating functions.

Problem

Given an array a of size n. $a_1, , , , a_n$. Find out the number of ways in which he can select s sweets. Output modulo $(10^9 + 7)$.

Solution

You have to find coefficients of x^s in $(1+x+x^2+...+x^{a_1})*(1+x+x^2+...+x^{a_2})*...(1+x+x^2+...+x^{a_n})$.

We know n is very large, $(1 \leq n \leq 10^6)$, but a_i is not very large $(0 \leq q_i \leq 10^3)$. We need to somehow make use of the small constraints of a_i. We will have many of the terms in $(1 + x + x^2 + ... + x^{a_1}) * (1 + x + x^2 + ... + x^{a_2}) * ...(1 + x + x^2 + ... + x^{a_n})$ being same and mulitplied many times, So clubing the same together and rearraning the expression, we get $(1 + x)^{c_1} * (1 + x + x^2)^{c_2} * ...(1 + x + x^2 + ... + x^{a_{1000}})^{c_{1000}}$.

Now using the geometic progression sum, we can re-write the expression as $(x^2 - 1)^{c_1} * (x^3 - 1)^{c_2} * .. * (x^{1001} - 1)^{c_{1000}} * (x - 1)^{-n}$

Now we need to find out coefficient of x^k $(0 \leq k \leq 2000)$. Let us split the expression into two parts. First part being $(x^2 - 1)^{c_1} * (x^3 - 1)^{c_2} * .. * (x^{1001} - 1)^{c_{1000}}$ and second part being $(x - 1)^{-n}$. We need to find coefficient of x^k $(0 \leq k \leq 2000)$ in the both the expression and later we can just muliply both the parts directly to get actual coefficients required in $\mathbf{O}(2000^2)$.

Now let us focus on finding coefficients of x^k in the first part only. We can do this in $\mathbf{O}(n^2 logn)$ time using dp. Note that we can find coefficient of x^k in $(x^i - 1)^{c_i}$ easily. All the coefficients in $(x^i - 1)_i^c$ will be of only of powers of x^i, coefficients of all other powers of x will be zero.

Now we can do a dp for the above purpose. Let dp[i][s] denotes coefficient of s in first i terms of the given expression $(x^2 - 1)_1^c * (x^3 - 1)_2^c * .. * (x^{1001} - 1)^{c_1 000}$.

Now let us say we are currently at i^{th} position in the expression (ie at $(x^i - 1)^{c_i}$), we will try all multiplies of $i + 1$ as possible powers of x and will do a transition to the previous state in the dp. See the exact code for details about the transition function.

Note that we are making $\frac{1000}{i}$ transition per step due to which our complexity will be like $\mathbf{O}(1000 * 1000 * (1000/1 + 1000/2 + ..1000/1000))$ $\mathbf{O}(1000 * 1000 * (H(1000))$ where H is haromic function. As H_n is bounded by $logn$. We can say overall complexity is $\mathbf{O}(1000 * 1000 * log(1000))$.

Now let us focus to second part of the expression. How to find coefficient of x^k in $(x - 1)^{-n}$?. Please follow the following link for entire explanation Negative Exponents in Binomial Theorem.

Please see the commented code for full understanding of the dp states.

Complexity of the entire solution is $\mathbf{O}(1000^2 log 1000)$ which will easily pass within the time.

Implementation

```
#include <bits/stdc++.h>

using namespace std;

typedef long long LL;
const int N = (int) 1e6 + 4005;
const LL mod = (LL) 1e9 + 7;

LL c[N], fact[N], inv[N], dp[2005][2005], ans[4005];

// calculates (-1)^n
int sign (int n) {
   if (n & 1) return -1;
   else return 1;
}

// calculates a^b % mod
LL modpow (LL a, LL b, LL mod) {
   LL res = 1;
   while (b) {
      if (b & 1) res = (res * a) % mod;
      a = (a * a) % mod;
      b >>= 1;
   }
   return res;
}

// calculates nCr.
LL C (LL n, LL r) {
   if (r < 0 || r > n) return 0;
   LL ans = (fact[n] * inv[n - r]) % mod;
   return (ans * inv[r]) % mod;
}


int main () {
   fact[0] = 1;
   // calculate factorial and inverse modulo p for each factorial.
   for (LL i = 0; i < N; i++) fact[i + 1] = (LL) (fact[i] * (i + 1)) % mod;
   for (int i = 0; i < N; i++) inv[i] = modpow (fact[i], mod - 2, mod);

   // process the input and calculate array c_i as shown in the editorial.
   int n;
   scanf ("%d", &n);
   for (int i = 0; i < n; i++) {
      int foo;
      scanf ("%d", &foo);
      c[foo + 1] ++;
   }

   // Now we come for the dp step.
   // dp[i][s] = as given in the editorial, coefficient of x^s in first i steps.
   dp[0][0] = 1;
   for (int i = 1; i <= 1001; i++) {
      for (int s = 0; s <= 2000; s++) {
         int k = i;
```

```
        LL sum = 0;
        // try all coefficients of x^(k * l), this is crucial step in complexity reduction.
        for (int l = 0; l <= s; l++) {
          if (k * l > s) break;
          // coefficient of x^(k * l) in (x^(i + 1) - 1) ^ c_i
          LL prod = C (c[i], l) * sign (c[i] - l);
          sum = (sum + dp[i - 1][s - k * l] * prod) % mod;
          if (sum < 0) sum += mod;
        }
        dp[i][s] = sum;
      }
    }

    // Now do the multiplication part as shown in the editorial.
    for (int i = 0; i <= 2000; i++) {
      for (int j = 0; j <= 2000; j++) {
        // coefficients of x^j in (x - 1)^(-n) is (-1)^n C (n + j - 1, j), See the given link for explantion.
        ans[i + j] = (ans[i + j] + dp[1001][i] * sign (n) * C (n + j - 1, j)) % mod;
        if (ans[i + j] < 0) ans[i + j] += mod;
      }
    }

    // finally take the query input as given in the problem.
    int Q;
    scanf ("%d", &Q);
    while (Q--) {
      int s;
      scanf ("%d", &s);
      printf ("%lld\n", ans[s]);
    }

    return 0;
}
```

IOPC 2014

Number Transform

Input: n, a, b, c, d, k, q
To convert n to p such that $p \mod k \equiv q$
Allowed Transformations

- $n \mapsto n + a$
- $n \mapsto n - b$
- $n \mapsto n * c$
- $n \mapsto n \mod d$

Explanation

This problem can be solved using *BFS* starting from n and then visiting $n + a$, $n - b$, $n * c$, $n \mod d$. Until we reach a number p such that $p \mod k \equiv q$.

The crux of the problem is that as $n + a$, $n - b$, $n * c$, $n \mod d$ get bigger instead of storing them modulo k we have store them modulo $d \cdot k$ or we have to visit $((n + a) \mod d \cdot k)$, $((n - b) \mod d \cdot k)$, $((n * c) \mod d \cdot k)$, $((n \mod d) \mod d \cdot k)$.

This is because $(((n \mod d) \mod k) + a) \mod k \neq ((n \mod d) + a) \mod k$. The proof is left as exercise.

Problem N: Swap and Explore

Difficulty

Cakewalk, requires simple looping and knowledge of maintaining prefix sum of the array. It can also be solved using two pointer or sliding window method.

Problem

Given a string s, Find out min number of swaps needed such as to have a substring of $\geq m$ consecutive A's in the string. You can swap any two positions in the array, It is also guaranteed that number of A's in the string are $\geq m$.

Solution

First of all, the guarantee that number of $A's$ in the string are $\geq m$, ensures that it is always possible to get the required substring. Next observation is that getting from m consecutive $A's$ to $m+1$ will always require additional swaps. Hence we will only try to solve the problem for exactly m consecutive $A's$.

Now consider the first substring of string s of size m ie, $s[0]$ to $s[m-1]$. If the string consists of all $A's$, then we dont need to do anything and we have found the answer without requiring to swap. Otherwise, let us say the substring has k A's, then we require $m-k$ $A's$ still to make the substring with all $A's$ only. Now as we are ensured that number of $A's$ in the string are $\geq m$, so we can change this substring into a substring with all $A's$ with $m-k$ swaps. In a single swap, you can take any of $A's$ outside the current substring and swap that with a non A character in the our current substring.

You need to do exactly $m-k$ such operations.

Implementation

Recurrence relation for maintaining prefix sum of the string is as follows.

if (s[i] == 'A') then prefixSum[i] = prefixSum[i - 1] + 1.
otherwise prefixSum[i] = prefixSum[i - 1].

Here prefixSum[i] denotes number of $A's$ in the prefix of length i of the string s ie. $s[0, i]$.

So using above mentioned recurrence, we will loop over all the substrings of size m and for finding out number of $A's$ in that substring we can use prefixSum information to find the maximum count of number of $A's$ in any substring of size m, making the overall complexity of the algorithm $\mathbf{O}(n)$.

Please see commented code snippet for more details..

```
a[0] = 0; // here a is our prefixSum array/
for (int i = 0; i < n; i++) a[i + 1] = a[i] + (s[i] == 'A');
// the above loop maintains prefix sum.
int mx = 0; // we are finding out the max number of A's in any substring of size m
for (int i = 0; i + m <= n; i++)
  if (a[i + m] - a[i] > mx) mx = a[i + m] - a[i]; // a[i + m] - a[i] gives number of A's in the substring s[0] t
printf("%d\n", m - mx); // answer will m - mx, as given in explanation
```

Another Solution

You can use a sliding window algorithm to solve the problem too. Maintain a window of length m corresponding to substrings of size m in the substring, Each time when you advance from i to $i+1$, update the count of $A's$ of the previous window to get counts of $A's$ in the current window in $\mathbf{O}(1)$ time. Overall complexity of this algorithm is also $\mathbf{O}(n)$.

Problem O: Infinite GP in Infinite GP

Difficulty

Cakewalk, adhoc Mathematics.

Problem

The problem requires the knowledge of Geometric Progressions. You can read more about it here.

Solution

Assume the required GP is: $\frac{1}{2^a}$, $\frac{1}{2^{a+b}}$, $\frac{1}{2^{a+2b}}$... Its sum can be written as: $\frac{2^{b-a}}{2^b-1}$. You just have check whether the given fraction $\frac{p}{q}$ is of this form or not.
First, reduce $\frac{p}{q}$ to its irreducible form. Since the values of p, q are relatively small, you can save the values of all permissible powers of 2.
For this, first, remove all powers of 2 from the fraction. The remaining fraction should be of the form $\frac{1}{2^b-1}$. Find b and substitute it back to get a. If you get valid a, b answer is Yes, else No.

```
from fractions import gcd

def rip(n):
   while(n%2==0):
      n=n/2
   return n
def ispow2(n):
   while(n%2==0):
      n=n/2
   return n==1

def which(n):
   ans=0
   while(n%2==0):
      n=n/2
      ans+=1
   return ans

test=int(raw_input())
for t in range(0,test):
   inp=raw_input().split(" ")
   p=int(inp[0])
   q=int(inp[1])
   g=gcd(p,q)
   p=p/g
   q=q/g

   cn=rip(p)
   cd=rip(q)

   if(cn!=1):
      print "NO"
      continue
```

```
if(ispow2(cd+1)==0):
   print "NO"
   continue

temp=(q/cd)*(cd+1)
if(temp%p==0):
   temp/=p
   if(ispow2(temp)):
      print "YES"
   else:
      print "NO"

else:
   print "NO"
```

P. Power Tower

Difficulty

Medium, requires CRT and some Number Theory Properties.

Problem

Given a tree in which each node has a value.
You need to do 2 operations :
a) Update the value of a node(which is bounded by 5)
b) Query for the cost of the path from A to B.
Cost is defined as $val(A)^{val(parent(A))^{\cdot^{val(B)}}} \mod M$

Solution

The Basic Idea Exploited here is Chinese Remainder Theorom.
let $m = p_1^{e_1} * p_2^{e_2} * p_3^{e_3} * .... * p_k^{e_k}$
p_i is a prime number.
let expr = A[1] $\wedge A[2] \wedge .... \wedge A[n] \mod M$
so, calculating $expr \bmod m$ is equivalent in calculating $CRT(expr \bmod p_1^{e_1}, expr \bmod p_2^{e_2}, ..., expr \bmod p_k^{e_k})$

Now lets see the case when $e_i = 1$, then calculating $expr \bmod p_i$ is equivalent to $A[1]^{(A[2] \wedge .... \wedge A[n] \bmod (p_i - 1))} \bmod p_i$.Euler Totient Period for p_i is $p_i - 1$.
Now for the case of $p_i^{e_i}$ where $e_i ! = 1$, euler totient period is given by $p_i^{e_i} - p_i^{e_i - 1}$
So calculate it recursively but where to stop ?
Stop as soon as mod become 1.
Overall idea is to calculate it recursively .
What is the maximum step we need to go up when the period will boil down to 1 ?
It is bounded by 18 steps (as $2^{18} > 100000$, proof is left as an exercise. Though it is easy to proof that the maximum step is bounded by 36 steps, Time Limit was kept very relaxed and if one tried for 36 steps would surely get it accepted.
Note:
a) Be carefull with Euler Totient period concept which is valid only if A[1] is coprime to p_i. It is left as an exercise for the reader to figure it out how to procede then.
b) Number of Updates had no issue on time complexity as updates can be done in O(1) time , it was used to trap user thinking .

Problem Q: Counting Trees

Difficulty

Medium Hard, requires knowledge of flow/ greedy and understanding the idea of lexicographically smallest matrix.

Problem

Consider a matrix A of size $N*M$, entries of A can only be 0 or 1. You are given two arrays $rowSum[N]$ and $columnSum[M]$ where $rowSum[i]$ represents sum of i^{th} row of matrix A and $coloumnSum[i]$ represents sum of i^{th} coloumn of matrix A. Now you wonder that can you find at least one matrix A, satisfying these constraints, If you can find such matrix A, then output the lexicographically smallest such matrix. Otherwise output -1.

Solution

To check whether a given solution is feasible or not?

pick row 1 lets say it remand is d_1 lets say demand of first column is c_1, second is c_2, ... where $c1 \geq c2 \geq ..c_m$ then assign 1 to first d_1 columns in row 1 for row 2, reduce $c_1, c_2, ...c_{d_1}$ by 1. Reshuffle to maintain non-increasing order and continue in this following order.

Complexity of the solution is $\mathbf{O}(n^4)$ which will easily pass within the time. Time limit was strict for this problem. If you used even slightly slow solution, then it will be time limit exceeded.

Implementation

```
/* @author: Praveen Dhinwa */
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
#define rep(i,n) for(int (i)=0;(i)<(int)(n);(i)++)
#define rep1(i,a,b) for(int (i)=(a);(i)<=(int)(b);(i)++)
#define fore(c,itr) for(__typeof((c).begin()) itr=(c).begin();itr!=(c).end();itr++)

const int N=55;
int rs[N],cs[N],a[N][N],myr[N],myc[N],n,m;
vector<vector<int> >Adj(55);

vector<int >b;
int possible(){
    // use greedy algorithm as explained in the editorial.
    rep(i,n) myr[i]=rs[i]; rep(i,m) myc[i]=cs[i];
    rep(i,n) {
        b.clear();
      int cnt=0;
      // use counting sort to make solution O(n^4).
      int mx=0; rep(j,m) if(a[i][j]==-1) mx=max(mx,myc[j]);
      rep(j,mx+1) Adj[j].clear();
        rep(j,m) if(a[i][j]==1) myc[j]--,cnt++; else if(a[i][j]==-1) Adj[myc[j]].push_back(j);
      for(int val=mx;val>=0;val--) rep(j,Adj[val].size()) b.push_back(Adj[val][j]);
        myr[i]-=cnt;
        if(b.size()<myr[i]) {return false;}
      rep(j,myr[i]) if(myc[b[j]]==0) return false; else myc[b[j]]--;
```

```
        myr[i]=0;
    }
    rep(i,m) if(myc[i]) return false;
   rep(i,n) if(myr[i]) return false;
    return true;
}

int main() {
    int T,tc=1; cin>> T; assert(T>=1&&T<=100); while(T--) {
        cout<<"TestCase #:"<<tc++<<"\n";
        cin>>n>>m; assert(n>=1&&n<=50); assert(m>=1&&m<=50);
        rep(i,n) {cin>>rs[i]; assert(rs[i]>=0&&rs[i]<=m);}
        rep(i,m) {cin>>cs[i]; assert(cs[i]>=0&&cs[i]<=n);}
        int valid=true,valid1=true; memset(a,-1,sizeof(a));
        // if even sumRow != sumCol, then answer is -1.
        if(accumulate(rs,rs+n,0)!=accumulate(cs,cs+m,0)) valid1=false;
        // For building lexicographically smallest solution, first fill array upto some position in
        // lexicographical order and then check whether it is still possible to extend the solution.
        rep(i,n) { if(!valid) break; rep(j,m) {
            int ok=false; rep(k,2) {a[i][j]=k; if(possible()) {ok=true; break;}}
            if(!ok) {valid=false; break;}
        }}
        if(!valid1) assert(valid==false);
        if(valid) rep(i,n) {int cnt=0; rep(j,m) if(a[i][j]) cnt++; assert (rs[i]==cnt);}
        if(valid) rep(i,m) {int cnt=0; rep(j,n) if(a[j][i]) cnt++; assert (cs[i]==cnt);}
        if(!valid) cout<<-1<<endl;
        else rep(i,n) {rep(j,m) {cout<<a[i][j];} cout<<"\n";}
    }
    return 0;
}
```

Another Implementation

```
/* author: illusion_iopc */
#include <bits/stdc++.h>
using namespace std;

#define LL long long int
#define PII pair<int,int>
#define PB push_back
#define MP make_pair
#define INF 1000000000
#define debug(args...) do {cerr << #args << ": "; dbg,args; cerr << endl;} while(0)

int cap[201][201];
vector<vector<int> > v;
int vis[201],id = 0;

// check the feasibility using flow.
int getflow(int cur,int dest){
    int i;
    if(vis[cur] == id)
        return 0;
    vis[cur] = id;
    for(i=0;i<(int)v[cur].size();i++){
        if(cap[cur][v[cur][i]] > 0){
```

```
            if(v[cur][i] == dest){
                cap[cur][v[cur][i]]--;
                cap[v[cur][i]][cur]++;
                return 1;
            }
            if(getflow(v[cur][i],dest)){
                cap[cur][v[cur][i]]--;
                cap[v[cur][i]][cur]++;
                return 1;
            }
        }
    }
    return 0;
}

int main(){
    int n,m,i,j,fl = 0;
    int t;
    int cas = 0;
    cin >> t;
    while(t--){
        memset(cap,0,sizeof(cap));
        cin >> n >> m;
        v.clear();
        v.resize(201);
        int sm = 0,sm2 = 0;

        for(i=0;i<n;i++){
            cin >> cap[199][i];
            sm += cap[199][i];
            v[199].PB(i);
            v[i].PB(199);
        }
        for(i=0;i<m;i++){
            cin >> cap[n + i][200];
            sm2 += cap[n+i][200];
            v[200].PB(n+i);
            v[n+i].PB(200);
        }
        for(i=0;i<n;i++){
            for(j=0;j<m;j++){
                v[i].PB(j+n);
                v[j+n].PB(i);
                cap[i][j+n] = 1;
            }
        }
        printf("TestCase #:%d\n", ++cas);
        int ans = 0;
        id ++;
        while(getflow(199,200)){
            id ++;
            ans ++;
        }
        // if sumRow != sumCol then answer is -1 as explained in editorial.
        if(ans != sm || sm != sm2){
            cout<<"-1"<<endl;
            continue;
        }
```

```
        for(i=0;i<n;i++){
            for(j=0;j<m;j++){
                // check the feasibility using flow.
                if(cap[n+j][i] == 1){
                    cap[n+j][i] = 0;
                    cap[i][199] --;
                    cap[199][i] ++;
                    cap[n+j][200]++;
                    cap[200][n+j] --;
                    id ++;
                    if(getflow(199,200) != 1){
                        cap[n+j][i] = 1;
                        cap[i][199] ++;
                        cap[199][i] --;
                        cap[n+j][200]--;
                        cap[200][n+j] ++;
                    }
                }
                else
                    cap[i][n+j] =0;
            }
        }
        for(i=0;i<n;i++){
            for(j=0;j<m;j++){
                if(cap[n+j][i] == 1)
                    cout<<"1";
                else
                    cout<<"0";
            }
            cout<<endl;
        }
    }
    return 0;
}
```

Problem R: LCS Returns

Difficulty

Easy, requires knowledge of greedy algorithms and simple understanding of LCS (Longest common sub-sequence)

Problem

You are given a string s of size n consisting only of lowercase English letters ('a' to 'z'). You have to find minimum length of string t also consisting of lowercase English letters('a' to 'z') such that length of LCS $(s, t) \geq L$. The string t should not have more than K consecutive equal characters.

Solution

Make an array a denoting count of consecutive equal characters. eg. aaabbaab then array a should be $\{3, 2, 2, 1\}$. Now you can take all the number $\leq K$, without spending anything for it.

Now consider the modified array after deleting all the numbers $\leq K$, You can take length $K * size(a)$ without spending anything. Now whatever operation I will do on the remaining $a_i s$, I will have to pay cost for that.

Now all the positions in the array which are $\geq$ K, you take a single K from of them and it will cost you one operation. Sort the array again, apply the above step again.

Finally you will have your sequence only containing elements with $\leq K$. Now for all those I have to pay a single cost, So I will try them in decreasing sorted order.

Complexity analysis: $\mathbf{O}(nlogn)$.

A trial run of algorithm:

Given a string 2z's, 3a's, 4b's, 5 c's, 6 d's, 7e's. K = 3. Now you to find minimum length of string t such that $LCS(s, t) \geq 12$.

First I will take 2zs and 3as without paying anything. len = 5. Then I will take $3as3bs3cs3ds$ without paying anything. len = 15.

Array a is now 1, 2, 3, 4. Now take 3 from a_4, You will be remaining with array len = 18. 1, 2, 3, 1.

Array a is now 1, 2, 3, 1. Now take 3 from a_3, You will be remaining with array len = 21. 1, 2, 0, 1 Now all the elements $\leq K$. Sort them pick the top two to make the length = 23.

Implementation

```
// @author: Praveen Dhinwa
#include <bits/stdc++.h>

using namespace std;

typedef long long LL;

int ceil (int n, int k) {
   return (n + k - 1) / k;
}
```

```
int main() {
   int T;
   cin >> T;
   while (T--) {
      string s;
      cin >> s;
      int k, L;
      cin >> k >> L;

      vector <int> a;
      int len = 0; // denotes LCS that can be made using extras more digits.
      int cnt = 1;
      int extras = 0; // denotes how many extra character we need to take.

      for (int i = 1; i < s.size(); i++) {
         if (s[i] == s[i - 1]) cnt ++;
         else {
               // if number of characters are <= k, we can take in the first trial without paying anything.
            if (cnt <= k) len += cnt;
            else {
               a.push_back (cnt - k);
               len += k;
            }

            cnt = 1;
         }
      }

      if (cnt > 0) {
         if (cnt <= k) len += cnt;
         else {
            a.push_back (cnt - k);
            len += k;
         }
      }

      if (len >= L) { // we already have LCS(s,t) >= L, we dont need any extra characters.
         extras = 0;
      } else {
         vector <int> modValues;
         int done = false;

         for (int i = 0; i < a.size(); i++) {
            if (len >= L) {
               done = true;
               break;
            }

               // check if we can make the lcs(s,t) >= L in using this a_is only.
            int canAdd = (a[i] / k) * k;
            int needed = L - len;

            if (needed <= canAdd) {
               len = L;
               extras += ceil (needed, k);

               done = true;
               break;
```

```
            } else {
                // otherwise we have can take can canAdd / k, remaining will go into modulo values, Which we will tak
                // sorted in decreasing order.
               extras += canAdd / k;
               modValues.push_back (a[i] % k);

               len += canAdd;
            }
         }

         if (len >= L) {
            done = true;
         }

         if (!done) {
            // now try the remainder thing :)
            // sort the modulo values in decreasing order and take one by one.
            sort (modValues.begin(), modValues.end());
            reverse (modValues.begin(), modValues.end());

            for (int i = 0; i < modValues.size(); i++) {
               if (modValues[i] == 0) continue;

               if (len >= L) break;

               len += modValues[i];
               extras ++;
            }
         }
      }

      assert (len >= L);

      int ans = L + extras;

      cout << ans << endl;
   }

   return 0;
}
```

Problem S: Saying Hello

Difficulty

Medium-Hard, Graph Theory / Numerical Methods.

Problem

The problem is to find the expected time to reach node t from 1 and come back.

Solution

There is a numerical method solution to this problem using Gauss-Jordan elimination technique, but it is quite non-trivial and requires hardcore mathematics.
The elegant solution is using Depth First Search (DFS). When you take the detour on any route, the probability of taking that path reduces by a fraction at every node. As hinted in the explanation, the increment in the length of path at every node is linear, but the decrement in its probanility is exponential, resulting in AGP. You can solve this question by simply travelling on paths, and aborting a path if its factor in the expected time (i.e probability x time taken) reduces below some epsilon.
Though, you are advised to solve this question using both the methods, especially using numerical technique, as it will boost your confidence!

Problem T

Reduction of the problem

Given the constraint : Questions of same section cannot be attempted consecutively.
Consider the reduction of the problem to an analogous problem of graph theory. Let the questions represent the nodes of the graph. An edge exists between two nodes u and v if question v can be attempted after question u, or vice-versa. Because of the above constraint, there cannot be an edge between two questions of the same section. Hence the k sections in the paper represent the k partitions of the graph, such that there is no edge between two nodes of the same section. Also, its given that "You can assume that the number of students is enough to encompass all possible combinations of question attempts, as allowed by the above constraint; in short, for every two question I and J, you are assured that all possible ways in which one can move from question I to question J is attempted by atleast one student". This means that all vertices have an edge to all other. Hence the graph is a **complete k-partitite graph**.

Let the labels on the buzzers pressed on going from one question to another be represented by colours on the edges. Hence the job of the teacher is to ensure that between every two nodes there exists a path, such that all edges on that path have distinct colours. The minimum number of colours required to ensure that this property is satisfied by the graph is commonly known as the **rainbow connection (rc) number** of the graph.

Therefore the problem is to find the rainbow connection number of a complete k-partitite graph.

Remark : Convince yourself that even though the diameter of the graph is 2, the rc number of the graph may be greater than 2.

Solution

Let $G = K_{n1,n2...nk}$ be the complete k-partite graph, where $n_1 \leq n_2 .... \leq n_k$ and $s = \sum_{i=1}^{k-1} n_i$ and $t = n_k$.

The problem can be reduced to the following cases :
1. $k = 2, n_1 = 1$
In this case, it can be trivially seen that the rc-number of G is n_2.

2. $k = 2, n_1 \geq 2$
In this case, the rc-number will be
$rc(K_{n_1,n_2}) = min\{\lceil \sqrt[s]{t} \rceil, 4\}$

3. $k \geq 3$
In this case, the rc-number will be

$$rc(G) = \begin{cases} 1, & \text{if } n_k = 1 \\ 2, & \text{if } n_k \geq 2 \text{and } s > t \\ min\{\lceil \sqrt[s]{t} \rceil, 3\}, & \text{if } s \leq t \end{cases} \quad (1)$$

The proof of the cases 2 and 3 can be found at the following link :
`http://dml.cz/bitstream/handle/10338.dmlcz/133947/MathBohem_133-2008-1_8.pdf`