**This is a record for some good problems in contests that I have done on Codeforces**.

**Author: Edward Lu**

PROBLEM SET:

**Square root decomposition:**

455D

121E

Usually used when N <= 10^5 and with some query like “what is the number of elements in a[l..r] that equals k?”

**Hard DP:**

**Hard Data structure:**

**Hard Graph:**

CONTESTS:

**Educational round 39**

Problem E:

Let's call a positive integer *x* *beautiful* if its decimal representation without leading zeroes contains even number of digits, and there exists a permutation of this representation which is palindromic. For example, 4242 is a beautiful number, since it contains 4 digits, and there exists a palindromic permutation 2442.

Given a positive integer *s*, find the largest beautiful number which is less than *s*

The sum of lengths of *s* over all testcases doesn't exceed 2·105.

Answer:

If a number is beautiful, then the length of it must be even and each of it’s digit appears an even number of times.

In order to make such beautiful number as large as possible, we want to make it as close to s as possible. To construct this, we notice that, there must exist a prefix (could be empty) of s, such that this prefix is also a prefix in our answer. We can greedily enumerate this prefix by iterating from N to 1 (because we want this prefix as long as possible).

To check if a prefix is valid, say, s[1 … k -1], we need to make sure that when using some digit in [0 .. s[k] - 1], the digits that appear an odd number of times in s[1 .. k - 1] can be filled also an odd number of times in ans[k +1 .. N]. In order to make this as large as possible, the rest of the empty spots in ans[k +1 .. N] are assigned 9.

If we can not find such prefix. Output N – 2 “9”s.

Code:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
4. **const** **int** maxn = 2e5 + 10;
5. **char** a[maxn];
6. **int** cnt[maxn][10]; // up to length i, #(appear digigt[j]) % 2
7. **int** N;
9. **void** solve(){
10. **for**(**int** i = N; i >= 1; --i){ // greedy step (prefix as long as possible)
11. **for**(**int** j = a[i] - **'0'** - 1; j >= 0; --j){ // also greedy step (j as large as possible)
12. **if**(i == 1 && j == 0)**continue**;
13. **int** odd = 0; // number of digits appear odd number of times
14. **for**(**int** k = 0; k <= 9; ++k){
15. **if**(k == j)odd += 1 - cnt[i - 1][k];
16. **else** odd += cnt[i - 1][k];
17. }
18. **if**(odd <= N - i){ // check if we can place them
19. **for**(**int** k = 1; k < i; ++k){
20. printf(**"%c"** , a[k]);
21. }
22. printf(**"%c"** , **'0'** + j);
23. **for**(**int** k = i + 1; k <= N - odd; ++k){
24. printf(**"9"**);
25. }
26. **for**(**int** k = 9; k >= 0; --k){
27. **if**(cnt[i - 1][k] ^ (j == k)){
28. printf(**"%c"** , **'0'** + k);
29. }
30. }
31. printf(**"\n"**);
32. **return**;
33. }
34. }
35. }
36. **// if we can not find any**
37. **for**(**int** i = 1; i <= N - 2; ++i)printf(**"9"**);
38. printf(**"\n"**);
39. }
41. **int** main(){
42. **int** T;
43. scanf(**"%d"** , &T);
44. **while**(T--){
45. scanf(**"%s"** , a + 1);
46. N = strlen(a + 1);
47. **for**(**int** i = 1; i <= N; ++i){
48. **int** v = a[i] - **'0'**;
49. **for**(**int** j = 0; j <= 9; ++j){
50. **if**(j == v)cnt[i][j] = 1 - cnt[i - 1][j];
51. **else** cnt[i][j] = cnt[i - 1][j];
52. }
53. }
54. solve();
55. }
56. }

Problem F.

You are given a binary string *s* (each character of this string is either 0 or 1).

Let's denote the cost of string *t* as the number of occurences of *s* in *t*. For example, if *s* is 11 and *t* is 111011, then the cost of *t* is 3.

Let's also denote the Fibonacci strings sequence as follows:

* *F*(0) is 0;
* *F*(1) is 1;
* *F*(*i*) = *F*(*i* - 1) + *F*(*i* - 2) if *i* > 1, where  +  means the concatenation of two strings.

Your task is to calculate the sum of costs of all subsequences of the string *F*(*x*). Since answer may be large, calculate it modulo 109 + 7.

(1 ≤ *n* ≤ 100, 0 ≤ *x* ≤ 100)

Answer:

This problem remind me of a problem in HackerRank.( <https://www.hackerrank.com/contests/ioi-2014-practice-contest-1/challenges/skwishinese-ioi14>)

F[i] is the concatenation of F[i - 1] and F[i - 2].

So we can calculate the length of F[i] easily. Hard part is that the definition of cost.

It is not easy to get all subsequences of F[x] and it is impossible to calculate the cost

directly from the subsequences. So maybe we can do this bit by bit. (subproblems)

Maybe we should try DP.

It is hard to calculate the cost of s[1..N] in F[x], but it is easy to calculate the cost of s[i .. i] in F[x]! And maybe we can try to get cost for s[l .. r] from s[x .. y] where x >= l && y <= r?

With this in mind, we can come up with a DP state, dp[i][l][r] = the number of times that s[l .. r] appears in F[i].

Let’s think about the transition.

Assuming that we have solved dp[0..i-1][0..N][0..N]

Firstly , consider s[l..r] as a whole in F[i-1]

If r == N, then no matter what subsequences we pick on F[i-2], it doesn’t affect the matching of s[l..r], so we have dp[i-1][l][r] \* 2^(len[i-2])ways

If r < N, then we need to be careful on F[i-2], because we might over count it. So we better choose empty subsequence in F[i-2], so we have dp[i-1][l][r] ways.

Similar for s[l..r] in F[i-2].

Now, we consider the case that F[i-1] has s[l..k] and F[i-2] has s[k+1..r].

We just multiply them. dp[i-1][l][k]\*dp[i-2][k+1][r] for k = l..r-1

Code:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **int** mod = 1e9 + 7;
6. **char** s[105];
7. **int** N , X;
8. **int** dp[105][105][105];
9. **int** len[105];
10. **int** power2[105];
12. ll add(ll a , ll b){
13. **return** (a + b) % mod;
14. }
16. ll mul(ll a , ll b){
17. **return** a \* b % mod;
18. }
20. ll qpow(ll x , ll n){
21. **if**(n <= 0)**return** 1;
22. ll ret = qpow(x , n / 2);
23. ret = mul(ret , ret);
24. **if**(n & 1)ret = mul(ret , x);
25. **return** ret;
26. }
28. **int** main(){
29. scanf(**"%d %d"** , &N , &X);
30. scanf(**"%s"** , s + 1);
31. len[0] = len[1] = 1;
32. **for**(**int** i = 2; i <= 100; ++i){
33. len[i] = (len[i - 1] + len[i - 2]) % (mod - 1);
34. }
35. **for**(**int** i = 0; i <= 100; ++i){
36. power2[i] = qpow(2ll , len[i]);
37. }
38. **for**(**int** i = 1; i <= N; ++i){
39. dp[0][i][i] = s[i] == **'0'**;
40. dp[1][i][i] = s[i] == **'1'**;
41. }
42. **for**(**int** i = 2; i <= X; ++i){
43. **for**(**int** len = 1; len <= N; ++len){
44. **for**(**int** l = 1; l + len - 1 <= N; ++l){
45. **int** r = l + len - 1;
46. **if**(r == N)dp[i][l][r] = add(dp[i][l][r] , mul(dp[i - 1][l][r] , power2[i - 2]));
47. **else** dp[i][l][r] = add(dp[i][l][r] , dp[i - 1][l][r]);
48. **if**(l == 1)dp[i][l][r] = add(dp[i][l][r] , mul(dp[i - 2][l][r] , power2[i - 1]));
49. **else** dp[i][l][r] = add(dp[i][l][r] , dp[i - 2][l][r]);
50. **for**(**int** k = l; k < r; ++k){
51. dp[i][l][r] = add(dp[i][l][r] , mul(dp[i - 1][l][k] , dp[i - 2][k + 1][r]));
52. }
53. }
54. }
55. }
56. printf(**"%d\n"** , (dp[X][1][N] + mod) % mod);
57. }

Problem G.

We call an array *almost increasing* if we can erase not more than one element from it so that the array becomes strictly increasing (that is, every element is striclty greater than every element before it).

You are given an array *a* consisting of *n* elements. You are allowed to replace any element with any integer number (and you may do so any number of times you need). What is the minimum number of replacements you have to perform in order to make the array *almost increasing*?

(2 ≤ *n* ≤ 200000) (1 ≤ *ai* ≤ 109)

Answer:

Ignore the fact that we can erase one element, the solution is just the LIS (longest increasing sequence). Here comes a **trick**!! Transform **{a[i]} to {a[i] - i}**, so that instead of finding LIS,

We can find LNS (longest non-increasing sequence). Now, we can enumerate on the element

That we can erase, for a[1] … a[N]. Now, our goal is to find the one LNS from a[1..k-1] and

one from a[k + 1..N], such that when you merging them they are LNS and as long as possible. This can be done by range tree that maintains the maximal in ranges.

Notice that, when merging 2 sequences, we want the longest ending at a[k - 1] and longest starting with elements at least a[k - 1] – 1 (after erasing, for i > k, a[i] -> a[i] - 1)

Code:

1. **#include** <bits/stdc++.h>
2. **using** **namespace** std;
4. **const** **int** maxn = 2e5 + 10;
5. **int** a[maxn];
6. **int** N , tot;
7. **unordered\_map**<**int** , **int**> mp , imp;
8. **int** pre[maxn] , suf[maxn];
10. **struct** maxtree{
11. **int** tree[maxn << 3];
13. **void** init(){
14. memset(tree , 0 , **sizeof**(tree));
15. }
17. **int** query(**int** l , **int** r , **int** x , **int** y , **int** id){
18. **if**(l <= x && y <= r){
19. **return** tree[id];
20. }
21. **int** mid = (x + y) >> 1;
22. **if**(mid < l)
23. **return** query(l , r , mid + 1 , y , id << 1 | 1);
24. **if**(mid >= r)
25. **return** query(l , r , x , mid , id << 1);
26. **return** max(
27. query(l , r , x , mid , id << 1),
28. query(l , r , mid + 1 , y , id << 1 | 1)
29. );
30. }
32. **void** update(**int** l , **int** r , **int** pos , **int** val , **int** id){
33. **if**(l > r || pos < l || pos > r)**return**;
34. **if**(l == r && pos == l){
35. tree[id] = max(tree[id] , val);
36. }
37. **else**{
38. **int** mid = (l + r) >> 1;
39. **if**(mid >= pos)
40. update(l , mid , pos , val , id << 1);
41. **else**
42. update(mid + 1 , r , pos , val , id << 1 | 1);
43. tree[id] = max(tree[id << 1] , tree[id << 1 | 1]);
44. }
45. }
47. } T;

50. **int** main(){
51. scanf(**"%d"** , &N);
52. tot = 1;
53. **vector<int>** tmp;
54. **for**(**int** i = 1; i <= N; ++i){
55. scanf(**"%d"** , &a[i]);
56. a[i] -= i;
57. tmp.push\_back(a[i]);
58. }
59. sort(tmp.begin() , tmp.end());
60. **for**(**int** i : tmp){
61. **if**(!mp[i]){
62. mp[i] = tot;
63. imp[tot] = i;
64. ++tot;
65. }
66. }
67. **for**(**int** i = 1; i <= N; ++i){
68. a[i] = mp[a[i]];
69. pre[i] = T.query(0 , a[i] , 0 , tot , 1) + 1;
70. T.update(0 , tot , a[i] , pre[i] , 1);
71. }
72. T.init();
73. **int** ans = N - 1 - pre[N - 1]; **// delete the last element**
74. **for**(**int** i = N - 1; i >= 1; --i){
75. suf[i + 1] = T.query(a[i + 1] , tot , 0 , tot , 1) + 1;
76. T.update(0 , tot , a[i + 1] , suf[i + 1] , 1);
77. **if**(mp[imp[a[i - 1]] - 1]){
78. ans = min(ans , N - 1 - pre[i - 1] - T.query(a[i - 1] - 1 , tot , 0 , tot , 1));
79. }
80. **else**{
81. ans = min(ans , N - 1 - pre[i - 1] - T.query(a[i - 1] , tot , 0 , tot , 1));
82. }
83. }
84. printf(**"%d\n"** , ans);
85. }

**Educational round 40**

This is a very good round of problems!

C.

This is a simulation problem, for all candidates y, simulate the whole process again.

E.

Goal is to make while the total sum of x[i] is maximal.

Suppose we have

For each t[i] = T, we can simply pick x[i] = a[i].

Then we can divide the remining into “positive” and “negative” groups and our goal is make the absolute value of them to be the same while maximising total sum of x[i]s.

Notice that, we want to make the maximal possible changes on (t[i] - T) while sacrificing as

Little of x[i] as possible, so here comes greedy.

Code:

1. **#pragma** GCC optimize(3)
2. **#pragma** GCC optimize(2)
3. **#include** <bits/stdc++.h>
4. **#define** DEBUG(x) std::cerr << **#x << '=' << x << std::endl**
5. **using** **namespace** std;
6. **typedef** **long** **long** ll;
7. **typedef** pair<**int** , **int**> ii;
9. **const** **int** maxn = 2e5 + 10;
10. **int** N;
11. **double** T;
12. **double** a[maxn] , t[maxn];
14. **double** solve(){
15. **double** **Sp** = 0 , **Sn** = 0 , ret = 0;
16. **vector<int>** id;
17. **for**(**int** i = 1; i <= N; ++i){
18. id.push\_back(i);
19. **if**(t[i] > T){
20. **Sp** += a[i] \* (t[i] - T);
21. }
22. **if**(t[i] < T){
23. **Sn** += a[i] \* (T - t[i]);
24. }
25. }
26. **double** goal = min(**Sp** , **Sn**);
27. **Sp** = **Sn** = goal;
28. sort(id.begin() , id.end() , [&](**int** x , **int** y){
29. **return** abs(t[x] - T) < abs(t[y] - T);
30. });
31. **for**(**int** i : id){ **// for positive group**
32. **if**(t[i] > T){
33. **double** take = min(a[i] , **Sp** / (t[i] - T));
34. **Sp** -= take \* (t[i] - T);
35. ret += take;
36. }
37. }
38. **for**(**int** i : id){ **// for negative group**
39. **if**(t[i] < T){
40. **double** take = min(a[i] , **Sn** / (T - t[i])); **// the maximal allowed x[i] for (T - t[i])**
41. **Sn** -= take \* (T - t[i]);
42. ret += take;
43. }
44. }
45. **for**(**int** i : id){
46. **if**(t[i] == T)ret += a[i];
47. }
48. **return** ret;
49. }
51. **int** main(){
52. scanf(**"%d %lf"** , &N , &T);
53. **for**(**int** i = 1; i <= N; ++i){
54. scanf(**"%lf"** , &a[i]);
55. }
56. **for**(**int** i = 1; i <= N; ++i){
57. scanf(**"%lf"** , &t[i]);
58. }
59. printf(**"%lf\n"** , solve());
60. }

F.

This problem is a matrix + sweeping task.

We can use DP when m is small, however m is large, but inspired by DP, one can find the transition matrix easily. The remaining task is to deal with the obstacles, we need to do sweeping on these obstacles by treating them as events. We also need to do modification over matrix when any 1 of the 3 rows is changed.

Code & details in comment:

G.

First thing is to notice that the solution is monotone. So we can do binary search on the solution. When verify some ans = x, we can use the “sliding window” technique.

When a tower’s defense power is less than x, we can add x – p[i] archers at tower i + r.

(Greedy observation).

Code:

1. **#pragma** GCC optimize(3)
2. **#pragma** GCC optimize(2)
3. **#include** <bits/stdc++.h>
4. **#define** DEBUG(x) std::cerr << **#x << '=' << x << std::endl**
5. **using** **namespace** std;
6. **typedef** **long** **long** ll;
7. **typedef** pair<**int** , **int**> ii;
9. **const** **int** maxn = 5e5 + 10;
10. ll N , K , R;
11. ll a[maxn] , delta[maxn];
12. ll sum;
14. **bool** check(ll x){
15. ll lim = K , power = 0;
16. memset(delta , 0 , **sizeof**(delta));
17. **for**(**int** i = 1; i <= N && i <= R + 1; ++i)power += a[i];
18. **for**(**int** i = 1; i <= N; ++i){
19. **if**(power < x){
20. **int** pos = min(N , i + R);
21. delta[pos] += x - power;
22. lim -= x - power;
23. power = x;
24. }
25. **if**(lim < 0)**return** 0;
26. **// need to get new stuffs**
27. **if**(i + R + 1 <= N){
28. power += a[i + R + 1] + delta[i + R + 1];
29. }
30. **// need to pop out**
31. **if**(i - R >= 1){
32. power -= a[i - R] + delta[i - R];
33. }
34. }
35. **return** 1;
36. }
38. **int** main(){
39. scanf(**"%lld %lld %lld"** , &N , &R , &K);
40. sum = K;
41. **for**(**int** i = 1; i <= N; ++i){
42. scanf(**"%lld"** , &a[i]);
43. sum += a[i];
44. }
45. ll l = 0 , r = sum , ans = 0;
46. **while**(l <= r){
47. ll mid = (l + r) >> 1;
48. **if**(check(mid)){
49. ans = mid;
50. l = mid + 1;
51. }
52. **else**{
53. r = mid - 1;
54. }
55. }
56. printf(**"%lld\n"** , ans);
57. }

H.

This is a combinatoric type of DP problem.

The idea is not hard.

Let dp1[i][j] = “number of nodes that I can reach that are exactly j steps from a node in level i, and these nodes are in the subtree rooted at i”. The transition: dp1[i][j] = dp1[i+1][j] \* a[i].

dp1 will handle all the pairs in subtrees.

For the pairs that are “across” subtrees, let dp2[i][j] = “number of nodes that I can reach that are exactly j steps from a node in level i, given that these nodes are not in the subtree rooted at i”. The transition: dp2[i][j] = dp2[i-1][j-1] + (a[i-1] - 1) \* dp1[i-1][j-2]

If we maintain these 2 arrays, we will get MLE, so we can use one array dp[i][j] only, calculate dp1 first, then dp2, using the following trick in the code:

**for(int i = 1; i <= N; ++i){**

**for(int j = 2 \* N - 2; j >= 1; --j){**

**dp[i][j] = dp[i - 1][j - 1];**

**if(i > 1 && j - 2 >= 0)dp[i][j] = add(dp[i][j] , mul(a[i - 1] - 1 , dp[i][j - 2]));**

**ans[j] = add(ans[j] , mul(dp[i][j] , level[i]));**

**}**

**}**

The order of enumeration forces that dp[i-1][1..2N-2] are all done. When calculating dp2[i][j], we are making sure that dp[i][j-2] is still equal to dp1[i][j-2].

**Educational round 80**

A.

Can we choose x such that x + ceil(d / (x + 1)) <= N ?

N , d <= 10^9

Enumerate x from 1 to

Intuitively, x + ceil(d / (x + 1)) should be bounded by some function of sqrt(d)

x + ceil(d / (x + 1)) <= (x + 1) + d / (x + 1) <= (1 / 2) \* sqrt(d) (AM-GM)

B.

Find #(pairs of a, b such that 1 <= a <= A , 1 <= b <= B , a \* b + a + b = concat(a , b))

Where concat(a , b) = ab

A , B <= 10^9

Observe that if a \* b + a + b = concat(a , b) = a \* 10^|b| + b

a \* b + a = a \* 10^|b|

b + 1 = |b|

b = |b| - 1

so the goal is to find number of b, such that b = |b| - 1

at the end, multiply by A (as this has nothing to do with a)

C.

Find #(a , b) such that

\*length of both a and b is m

\*each element is from in [1,n]

\*a[i] <= b[i], for 1 <= i <= m

\*a is in non-decreasing order

\*b is in non-ascending order

Observe that if we merge a and reverse(b), we get a non-decreasing sequence.

As a[1]<=a[2]<=..<=a[M]<=b[M]<=b[M-1]<=..<=b[1]

Let dp[i][j] = number of non-decreasing sequence of length i, where the maximal element is j.

dp[0][0]=1

dp[i][j]=dp[i-1][1]+..dp[i-1][j]

Ans = dp[2M][1]+..+dp[2M][N]

D.

Given N (<= 3e5) arrays, each has M (<=8) elements.

Choose array i and j, (i can be j) and b[k] = max(a[i][k] , a[j][k])

Goal is to choose i , j such that min{b[k]} is maximal possible.

We notice that if we can obtain a configuration with some choice of i and j, such that

Min{b[k]} >= x, then we can have min{b[k]} >= x – 1. Thus we notice that the answer

Is monotone. This reminds me of binary search. We can binary search the following query:

Is there exist a choice of some i,j such that min{b[k]} >= x?

We can check our query by the follows: let a[i][j] >= x be 1, let a[i][j] < x be 0, Similar to one question in COMP4128 lecture.

Then, we can view each array as a bit mask. To make min{max{a[i][k] , a[j][k]}} = 1,

We need (mask[i] | mask[j]) to be (1 << M) – 1. We can store a hashmap for each mask[i].

E.

Messenger simulator

Question is long, so here is the link to it: <https://codeforces.com/contest/1288/problem/E>

Approach:

Min position is easy, it is either 1 or it’s initial position, (we will not move element to the back)

The problem is the max position. Notice, that the position of an element changes only when we move it. And it’s position is exactly the number of elements in front of it.

So we can build a range tree/fenwick tree, on range[1 .. M+N], keep track with the number of elements in some range, as well as the latest position of each element in the tree!

So initially, pos[i] = M + i

Code:

1. **#include** **<iostream>**
2. **#include** **<algorithm>**
3. **using** **namespace** std;
5. **const** **int** maxn = 3e5 + 10;
7. **int** sum[maxn << 3];
8. **int** N , M;
9. **int** maxans[maxn] , minans[maxn] , pos[maxn];
11. **void** update(**int** pos , **int** val){
12. **while**(pos <= N + M){
13. sum[pos] += val;
14. pos += pos & (-pos);
15. }
16. }
18. **int** query(**int** pos){
19. **int** ret = 0;
20. **while**(pos){
21. ret += sum[pos];
22. pos -= pos & (-pos);
23. }
24. **return** ret;
25. }
27. **int** main(){
28. ios::sync\_with\_stdio(**false**);
29. cin.tie(**nullptr**);
30. cin >> N >> M;
31. **for**(**int** i = 1; i <= N; ++i){
32. update(i + M , 1);
33. minans[i] = maxans[i] = query(i + M);
34. pos[i] = i + M;
35. }
36. **for**(**int** i = M; i >= 1; --i){
37. **int** x; cin >> x;
38. minans[x] = 1;
39. maxans[x] = max(maxans[x] , query(pos[x]));
40. update(pos[x] , -1);
41. pos[x] = i;
42. update(pos[x] , 1);
43. }
44. **for**(**int** i = 1; i <= N; ++i){
45. cout << minans[i] << **" "** << max(maxans[i] , query(pos[i])) << endl;
46. }
47. }

F. Unfortunately, F is out of my league right now.

**Educational round 81**

**Educational round 82**

**Contest 622 Div2 (this round is difficult for me at that time)**

A. <https://codeforces.com/contest/1313/problem/A>

Greedy, try each a[0...2] first, then try pairs, try triple.

B. <https://codeforces.com/contest/1313/problem/B>

N is large, so there must be a math formula for this.

For minimal place, we want the number of ppl whose paired score is higher than mine as large as possible. So, greedily speaking, we want their score to be at least x + y + 1.

1 , 2 , …. , x + y

1 , 2 , …. , x + y

For these 2 rows, we can pair them up, so that each of the sum is at least x + y + 1.

If x + y <= N, then we can make the minimal place to be 1. (for individual score > x + y, they can be paired with any other number, and for score <= x + y since we need to pair up x and y, but all other pairs have sum at least x + y + 1)

Otherwise, the number of numbers that exceeds N which is x + y – N + 1.

So, in this case, our minimal place is min (N , x + y – N + 1)

For maximal place, we want the number of ppl whose paired score is not higher than mine as large as possible. So, we can apply greedy here once again, we want their score to be at most x + y.

1 , 2 , … x + y - 1

1 , 2 , … x + y - 1

We can pair them up such that each of their sum is exactly x + y.

So, we can make sure that there are x + y – 1 candidates having score x + y.

Therefore, our maximal place is min (N , x + y - 1)

C.<https://codeforces.com/contest/1313/problem/C2>

This problem can be solved by a divide-conquer way.

Notice that we cannot have j < i < k st a[j] > a[i] < a[k].

So for A[l…r] , when we pick a place to be the peak, say a[x], then

Either all elements in A[l…x - 1] are at most a[x] and we need to solve for A[x + 1 … r],

Or all elements in A[x + 1… r] are at most a[x] and we need to solve for A[l … x - 1].

Notice that we can only decrease our A[x], so it is obvious that for each [l … r] , the peak we pick is going to be the minimal value in this range. This can be done using range tree.

Code: <https://codeforces.com/contest/1313/submission/71735850>

D.<https://codeforces.com/contest/1313/problem/D>

This is a hard DP + sweeping problem.

To summarise, we have N intervals [l[i] , r[i]]. Each of them can cover a range of children,

And no more than k (<= 8) intervals can overlap. For a range, if the number of intervals covering it is odd then it’s contribution is it’s length, else it contributes 0. Now, our goal is to use some subset of the intervals to maximize the total contributions.

When there are ranges, we might think about sweeping, we can denote each interval as 2 events. (standard technique)

For a range of children, we care about which intervals are overlapping with it, since k is small, we can use bitmask to encode it. So we have our DP state, DP[i][mask] = maximal

Score we can get up to the i-th range of children (ranges are created by events), such that

Current range overlaps a “set” of intervals (which is encoded in mask).

Each new added interval will have a “id” representing the bit in mask that represents it.

When in interval is added, we can choose to either use it or do not use it, this can be interpreted as DP[i][state | (1 << id)] = max{DP[i][state | (1 << id)] , DP[i - 1][state] + dis}

And DP[i][state] = max{DP[i][state] , DP[i - 1][state] + dis}.

When an interval is leaving, we can enumerate the states, if the state include this bit, then we are removing this bit now from the state, otherwise, does not change.

DP[i][state ^ (1 << id)] = max{DP[i][state ^ (1 << id)] , DP[i - 1][state] + dis}

DP[i][state] = max{DP[i][state] , DP[i - 1][state] + dis}

Total complexity O(N \* 2 ^ k)

Code : <https://codeforces.com/contest/1313/submission/75076119>

**Contest 626 Div2**

**Educational round 83**

A.

Link: <https://codeforces.com/contest/1312/problem/A>

Solution:

Check if n % m = 0 -> yes, else no

(think about this in terms of internal angle)

B.

Link: <https://codeforces.com/contest/1312/problem/B>

Solution:

If i<j then j – a[j] != i – a[i]

This implies a[j] – j != a[i] – i.

If we sort the array in descending order, then we get a[i] >= a[i+1],

a[i]-i>a[i+1]-i-1, the property obviously holds.

C.

Link: <https://codeforces.com/contest/1312/problem/C>

Can these n numbers be written in base-K, where each power of K appears at most once?

The check is easy.

D.

Link: <https://codeforces.com/contest/1312/problem/D>

Since exactly one pair of equal, we need N-1 distinct numbers from M numbers.

The maximal number cannot be duplicated, so we have N-2 choices for the duplicated one.

Apart from the maximal one and the duplicated one, each of the N-3 numbers can go either the

Left or the right of the maximal element (and their order is fixed once decided)

So the answer is C(M , N - 1) \* (N - 2) \* 2 ^ (N - 3)

E.

Link: <https://codeforces.com/contest/1312/problem/E>

If we can shrink a block of numbers, then they will end up as “one” element.

So, our goal is to find a way so that the total number of “one” is minimised.

Notice that if we can check if a[i..j] can be shrinked into “one” element, then

There exists a[i..k] and a[k+1..j] such that, a[i..k] and a[k+1..j] can be shrinked into “one” element with same value. Once we are done for all ranges, we can have another dp, with

Dp[i] = “minimal number of “one”s for the first i elements”

Code:

1. **#pragma** GCC optimize(3)
2. **#pragma** GCC optimize(2)
3. **#include** <bits/stdc++.h>
4. **#define** DEBUG(x) std::cerr << **#x << '=' << x << std::endl**
5. **using** **namespace** std;
6. **typedef** **long** **long** ll;
7. **typedef** pair<**int** , **int**> ii;
8. **int** a[505];
9. **int** N;
10. **const** **int** inf = 1e9 + 10;
12. **int** dp[505][505];
13. **int** ans[505];
15. **int** main(){
16. cin >> N;
17. **for**(**int** i = 1; i <= N; ++i)cin >> a[i];
18. **for**(**int** i = 1; i <= N; ++i){
19. dp[i][i] = a[i];
20. }
21. **for**(**int** len = 2; len <= N; ++len){
22. **for**(**int** i = 1; i + len - 1 <= N; ++i){
23. **int** j = i + len - 1;
24. dp[i][j] = -1;
25. **for**(**int** k = i; k < j; ++k){
26. **if**(dp[i][k] != -1 && dp[i][k] == dp[k + 1][j]){
27. dp[i][j] = dp[i][k] + 1;
28. **break**;
29. }
30. }
31. }
32. }
33. **for**(**int** i = 1; i <= N; ++i){
34. ans[i] = inf;
35. **for**(**int** j = 1; j <= i; ++j){
36. **if**(dp[j][i] != -1){
37. ans[i] = min(ans[i] , ans[j - 1] + 1);
38. }
39. }
40. }
41. cout << ans[N] << endl;
42. }

F.

Link: <https://codeforces.com/contest/1312/problem/F>

TODO