CODEFORCES β
Sponsored by Telegram

himanshupareekiit01 | Logout

HOME TOP CONTESTS GYM PROBLEMSET GROUPS RATING API HELP HONORCUP CALENDAR

PIKMIKE BLOG TEAMS SUBMISSIONS GROUPS CONTESTS PROBLEMSETTING

PikMike's blog

Educational Codeforces Round 74 Editorial

By PikMike, history, 2 days ago, translation,

1238A - Prime Subtraction

Idea: **BledDest**

Tutorial

1238A - Prime Subtraction

Let's denote the difference between x and y as z ($z = x - y$). Then, if z has a prime divisor p, we can subtract p from x $\frac{z}{p}$ times.

The only positive integer that doesn't have any prime divisors is 1. So, the answer is NO if and only if $x - y = 1$.

Solution (BledDest)

```
t = int(input())

for i in range(t):
        x, y = map(int, input().split())
        if(x - y > 1):
                print('YES')
        else:
                print('NO')
```

1238B - Kill `Em All

Idea: **Ne0n25**

Tutorial

1238B - Kill `Em All

Notice the following fact: it's never optimal to fire a missile at such a position that there are monsters to the right of it. That suggests the next solution: sort the positions, leave only the unique ones and process to shoot at the rightmost alive monster until every monster is dead. Position of some monster after s shots are made is the original position minus $r \cdot s$, because the monster could only be pushed to the left.

Overall complexity: $O(n \log n)$.

Solution (Ne0n25)

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

using namespace std;

#define forn(i, n) for (int i = 0; i < int(n); i++)

const int N = 100 * 1000 + 13;
```

→ Pay attention

Before contest
Codeforces Round #592 (Div. 2)
36:59:22

→ himanshupareekiit01

Rating: 1515
Contribution: 0

- Settings
- Blog
- Teams
- Submissions
- Favourites
- Talks
- Contests

himanshupareekiit01

→ Top rated

#	User	Rating
1	tourist	3557
2	Um_nik	3494
3	Radewoosh	3344
4	wxhtxdy	3309
5	LHiC	3302
6	Benq	3286
7	mnbvmar	3274
8	Petr	3254
9	yutaka1999	3190
10	ksun48	3170

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Errichto	191
2	Radewoosh	180
3	Vovuh	166
4	PikMike	165
5	antontrygubO_o	164
6	rng_58	161
7	Um_nik	156
7	majk	156
9	300iq	155
10	farmersrice	151

View all →

→ Find user

Handle:

```
int n, r;
int a[N];

void solve() {
        scanf("%d%d", &n, &r);
        forn(i, n) scanf("%d", &a[i]);

        sort(a, a + n);
        n = unique(a, a + n) - a;

        int ans = 0;
        for (int i = n - 1; i >= 0; i--)
                ans += (a[i] - ans * r > 0);

        printf("%d\n", ans);
}

int main() {
        int q;
        scanf("%d", &q);
        forn(i, q) solve();
}
```

1238C - Standard Free2play

Idea: **adedalic**

Tutorial

1238C - Standard Free2play

You are given the input data in compressed format, let's decompress it in binary string, where the i-th character is `0` if the i-th platform is hidden and `1` otherwise. For, example, the third query is `101110011`.

Let's look how our string changes: if we had `...01...` then after pulling the lever it becomes `...10...` and if we had `...111...` then we'd get `...100...` (The underlined index is the platform we are currently on). So it looks like we are standing on `1` and move it to the left until it clashes with the next one. So we can determine that we should look only at subsegments on `1`-s.

Now we can note, that the "good" string should have all subsegments of ones with even length except two cases: the subsegment that starts from h should have odd length and subsegment, which ends in 1 can have any length.

Now we can say, that the answer is equal to number of subsegments which doesn't match the pattern of the "good string", since we can fix each subsegment with one crystal. And we can prove that it's optimal since the only way to fix two subsegments with one crystal is to merge them but it does not help.

Finally, we can understand that we have no need in decompressing the input and can determine subsegments of ones straightforwardly.

Solution (adedalic)

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

using namespace std;

#define fore(i, l, r) for(int i = int(l); i < int(r); i++)
#define sz(a) int((a).size())

#define x first
#define y second
```

Find

→ Recent actions

Subash23 → Bitset in C++

gaurav172 → Idleness Time exceeded.

lakshmi123 → Basketball exercise(editorial)

MikeMirzayanov → Codeforces Round #277.5 (Div. 2) Editorial [A-D for now]

noobita → Invitation to Exun 2019 Programming Prelims [Div1+Div2] [CodeChef]

PikMike → Educational Codeforces Round 74 Editorial

bvd → How to find the expected shortest distance between two points in a subset of a set of points on a line?

MikeMirzayanov → About the Failed Round 591/Technocup 2020 — Elimination Round 1

hmehta → Topcoder SRM 768 — Sponsored By Google — TCO20 Stage 1

McDic → Codeforces Round #589 (Div. 2) Editorial

c3885247 → Permutation Algorithm

notking → Searching for problems

motatoes → London UK Meetup (competitive programming London)

huzaifa242 → HackerEarth Data Structures and Algorithms Challenge October 2019

chokudai → AtCoder Beginner Contest 142 Announcement

300iq → Codeforces Round #562 — Editorial

pllk → CSES Problem Set update June 2019: New problems and hacking

Enchom → Hungarian algorithm

dragonslayerintraining → Codeforces Round #584 (Dasha Code Championship Elimination Round) (div. 1 + div. 2) Editorial

removed1 → Codeforces Beta Round #1 - Tutorial

Roms → Codeforces Round 591 (and Technocup 2020 — Elimination Round 1) Editorial

sgtlaugh → 2018-2019 ACM-ICPC, Asia Dhaka Regional Contest Online Mirror

Aritra741 → [HELP]Number of lattice points (integer points) on a line segment with fractional endpoints

ghoshsai5000 → DMOPC '19 October Contest Announcement

ouuan → Codeforces Round #564 Editorial

Detailed →

```
const int INF = int(1e9);

int h, n;
vector<int> p;

inline bool read() {
        if(!(cin >> h >> n))
                return false;
        p.resize(n);
        fore(i, 0, n)
                cin >> p[i];
        return true;
}

inline void solve() {
        int ans = 0;

        int lf = 0;
        fore(i, 0, n) {
                if (i > 0 && p[i - 1] > p[i] + 1) {
                        if (lf > 0)
                                ans += (i - lf) & 1;
                        else
                                ans += 1 - ((i - lf) & 1);
                        lf = i;
                }
        }
        if (p[n - 1] > 1) {
                if (lf != 0)
                        ans += (n - lf) & 1;
                else
                        ans += 1 - ((n - lf) & 1);
        }

        cout << ans << endl;
}

int main() {
#ifdef _DEBUG
        freopen("input.txt", "r", stdin);
#endif
        ios_base::sync_with_stdio(false);
        cin.tie(0), cout.tie(0);
        cout << fixed << setprecision(15);

        int tc; cin >> tc;
        while(tc--) {
                read();
                solve();
        }
        return 0;
}
```

1238D - AB-string

Idea: **Roms**

Tutorial

1238D - AB-string

Instead of counting the number of good substrings, let's count the number of bad

substrings $cntBad$, then number of good substrings is equal to $\frac{n(n+1)}{2} - cntBad$.

Let's call a character t_i in string $t_1t_2 \ldots t_k$ is bad if there is no such palindrome $t_lt_{l+1} \ldots t_r$ that $l \le i \le r$. Any character in substring $t_2t_3 \ldots t_{k-1}$ is good. It can be proven as follows. If $t_i = t_{i+1}$ or $t_i = t_{i-1}$ then t_i belong to a palindrome of length 2. If $t_i \neq t_{i+1}$ and $t_i \neq t_{i-1}$ then t_i belong to a palindrome $t_{i-1} \ldots t_{i+1}$.

So only characters t_1 and t_k can be bad. But at the same time character t_1 is bad if there is no character t_i such that $i > 1$ and $t_i = t_1$. It is true because substring $t_1t_2 \ldots t_i$ is palindrome (index i is minimum index such that $t_i = t_1$).

So, there are only 4 patterns of bad strings:

1. $ABB \ldots BB$;
2. $BAA \ldots AA$;
3. $AA \ldots AAB$;
4. $BB \ldots BBA$;

All that remains is to count the number of substrings of this kind.

Solution (Roms)

```
n = int(input())
s = input()
res = n * (n - 1) // 2

for x in range(2):
        cur = 1
        for i in range(1, n):
                if s[i] == s[i - 1]:
                        cur += 1
                else:
                        res -= cur - x
                        cur = 1
        s = s[::-1]

print(res)
```

1238E - Keyboard Purchase

Idea: **Roms**

Tutorial

1238E - Keyboard Purchase

Let's solve this problem by subset dynamic programming.

Let's denote $cnt_{x,y}$ as the number of adjacent characters (s_i and s_{i+1}) in s such that $s_i = x, s_{i+1} = y$ or $s_i = y, s_{i+1} = x$.

Let's dp_{msk} be some intermediate result (further it will be explained what kind of intermediate result) if we already added letters corresponding to subset msk to the keyboard (and we don't care about the order of these letters).

Now let's consider how to recalculate values of this dynamic programming using some dp_{msk}. Let's iterate over a new letter x on keyboard (and we know the position of this letter on the keyboard: it's equal to the number of elements in msk). After adding this new letter, we want to calculate what it added to the $dp_{msk \cup x}$. Let consider some letter $y \neq x$ and calculate how much time will be spent on moving $x \to y$ and $y \to x$. There are two cases. If letter y is already on current keyboard, then we should add to answer $cnt_{x,y}(pos_x - pos_y)$, and $cnt_{x,y}(pos_y - pos_x)$ otherwise (where pox_a is the position of character a on the keyboard). But we don't know the position of the letter y. Let's fix it as follows. We will add the contribution of some letter when it will be added to the keyboard. So, when we added letter x, we should add the value

$\sum_{y \in msk} (cnt_{x,y} pos_x) - \sum_{y \notin msk} (cnt_{x,y} pos_x)$.

So, the total complexity is $O(a^2 2^a + n)$.

Solution (Roms)

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

using namespace std;

void upd(int &a, int b){
        a = min(a, b);
}

const int N = 20;
const int M = (1 << N) + 55;
const int INF = int(1e9) + 100;

int a, n;
string s;
int cnt[N][N];
int d[M][N];
int dp[M];
int cntBit[M];
int minBit[M];

int main() {
        cin >> n >> a >> s;

        int B = (1 << a) - 1;
        for(int i = 1; i < s.size(); ++i){
                ++cnt[s[i] - 'a'][s[i - 1] - 'a'];
                ++cnt[s[i - 1] - 'a'][s[i] - 'a'];
        }
        for(int i = 0; i < M; ++i)
                dp[i] = INF;
        dp[0] = 0;
        for(int msk = 1; msk < M; ++msk){
                cntBit[msk] = 1 + cntBit[msk & (msk - 1)];
                for(int i = 0; i < N; ++i) if((msk >> i) & 1){
                        minBit[msk] = i;
                        break;
                }
        }
        for(int msk = 1; msk < M; ++msk)
                for(int i = 0; i < a; ++i){
                        int b = minBit[msk];
                        d[msk][i] = d[msk ^ (1 << b)][i] +
cnt[i][b];
                }

        for(int msk = 0; msk < (1 << a); ++msk){
                for(int i = 0; i < a; ++i){
                        if((msk >> i) & 1) continue;
                        //i -> x
                        int pos = cntBit[msk];
                        int nmsk = msk | (1 << i);
                        upd(dp[nmsk], dp[msk] + pos * (d[msk][i] - d[B ^
nmsk][i]));
                }
        }
```

```
        cout << dp[B] << endl;
        return 0;
}
```

1238F - The Maximum Subtree

Idea: **Roms**

Tutorial

1238F - The Maximum Subtree

At first let's understand which trees are good. For this, let's consider some vertex v (we denote its segment as $[l_v, r_v]$) which is not a leaf. Also let's consider some adjacent vertex u (we denote its segment as $[l_u, r_u]$) which also is not leaf. It is claimed that segment $[l_v, r_v]$ can't be inside segment $[l_u, r_u]$ (it's means $l_u \le l_v \le r_v \le r_u$) and vice versa. It's true because if segment $[l_v, r_v]$ is inside the segment $[l_u, r_u]$ then some vertex t adjacent with v also will be adjacent with u. So any non-leaf vertex can be adjacent with at most 2 non-leaf vertexes. Therefore good tree is a path with a leafs adjacent to this path.

So all the have to do it's find the such subtree of maximum size. We can do it by subtree dynamic programming.

At first, let chose the root of the tree — some not leaf vertex.

Let $dp_{v,0}$ be the answer for the subtree with root in v and dp_{v, 1} be the answer for the subtree with root in v if we already took v and its parent to the answer.

It can be calculated as follows:

- $dp_{v,0} = \max\limits_{to} dp_{to,0}$;
- $dp_{v,0} = \max(dp_{v,0}, deg_v + 1 + firstMax + secondMax)$, there $firstMax$ is a first maximum of all $dp_{to,1}$, and $secondMax$ is a second maximum, and deg_v is a degree of vertex v;
- $dp_{v,1} = deg_v - 1 + \max\limits_{to} dp_{to,1}$.

Solution (Roms)

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

using namespace std;

const int N = int(3e5) + 99;

int n;
vector <int> g[N];
int d[N];
int dp[N][2];

void dfs(int v, int p){
        vector <int> d1;
        dp[v][1] = d[v] - 1;
        for(auto to : g[v]){
                if(to == p) continue;
                dfs(to, v);
                dp[v][0] = max(dp[v][0], dp[to][0]);
                if(g[to].size() > 1){
                        d1.push_back(dp[to][1]);
                        dp[v][1] = max(dp[v][1], d[v] + dp[to][1] - 1);
                }
        }
```

```
        sort(d1.rbegin(), d1.rend());
        int x = d[v] + 1;
        for(int i = 0; i < 2; ++i)
                if(i < d1.size())
                        x += d1[i];
    dp[v][0] = max(dp[v][0], x);
}

int main() {
    int q;
    scanf("%d", &q);
    for(int qc = 0; qc < q; ++qc){
        scanf("%d", &n);
        for(int i = 0; i < n; ++i){
            g[i].clear();
            d[i] = 0;
            dp[i][0] = dp[i][1] = 0;
        }
        for(int i = 0; i < n - 1; ++i){
                int u, v;
                scanf("%d %d", &u, &v);
                --u, --v;
                g[u].push_back(v), g[v].push_back(u);
        }

        if(n <= 2){
            printf("%d\n", n);
            continue;
        }

        for(int v = 0; v < n; ++v){
                //d[v] = 1;
                //for(auto to : g[v])
                //      d[v] += g[to].size() == 1;
                d[v] = g[v].size();
        }

        int r = -1;
        for(int v = 0; v < n; ++v)
            if(g[v].size() != 1)
                r = v;

        dfs(r, r);
        printf("%d\n", dp[r][0]);
    }
        return 0;
}
```

1238G - Adilbek and the Watering System

Idea: **Ne0n25**

Tutorial

1238G - Adilbek and the Watering System

Despite the fact that statement sounds like some dp or flow, the actual solution is pretty greedy.

Let's iterate over all minutes Adilbek has to water at and maintain the cheapest C liters he can obtain to this minute. Let this be some structure which stores data in form (price for 1 liter, total volume Adilbek can buy for this price). Pairs will be sorted by the price of a liter. The most convenient structure for that might be a C++ map, for example.

When moving to the next minute, pop the cheapest liter out of this structure and add it to the answer.

If that minute some friend comes, then push his water to the structure: if the total updated volume in the structure is greater than C, then pop the most expensive left-overs out of it so that the structure holds no more than C liters total. That prevents out solution to fill the watering system over its capacity.

The main idea for why this greedy strategy works is that it's never optimal to take not the cheapest liter because a liter of that price or cheaper will still be available in the future minutes.

Note that between each pairs of adjacent coming friends basically nothing happens. Thus you can find the time between them and pop that number of cheapest liters right away instead of iterating minute by minute.

Overall complexity: $O(n \log n)$ per query.

Solution (Ne0n25)

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

using namespace std;

#define x first
#define y second
#define mp make_pair
#define forn(i, n) for (int i = 0; i < int(n); ++i)

typedef long long li;
typedef pair<int, int> pt;

const int N = 500 * 1000 + 13;

int n, m, c, c0;
pair<int, pt> a[N];

li solve() {
	scanf("%d%d%d%d", &n, &m, &c, &c0);
	forn(i, n) scanf("%d%d%d", &a[i].x, &a[i].y.x, &a[i].y.y);
	a[n++] = mp(m, mp(0, 0));
	sort(a, a + n);

	int sum = c0;
	map<int, int> q;
	q[0] = c0;

	li ans = 0;
	forn(i, n) {
		int x = a[i].x;
		int cnt = a[i].y.x;
		int cost = a[i].y.y;

		int dist = x - (i ? a[i - 1].x : 0);
		while (!q.empty() && dist > 0) {
			int can = min(q.begin()->y, dist);
			ans += q.begin()->x * 1ll * can;
			sum -= can;
			dist -= can;
			q.begin()->y -= can;
			if (q.begin()->y == 0) q.erase(q.begin());
		}

		if (dist > 0)
			return -1;
```

```
                int add = min(c - sum, cnt);
                sum += add;

                while (add < cnt && !q.empty() && q.rbegin()->x > cost)
{
                        if (cnt - add >= q.rbegin()->y) {
                                add += q.rbegin()->y;
                                q.erase(--q.end());
                        } else {
                                q.rbegin()->y -= cnt - add;
                                add = cnt;
                        }
                }

                q[cost] += add;
        }

        return ans;
}

int main() {
        int q;
        scanf("%d", &q);
        forn(i, q) printf("%lld\n", solve());
}
```

Tutorial of Educational Codeforces Round 74 (Rated for Div. 2)

Tutorial of Educational Codeforces Round 74 (Rated for Div. 2)

+40 PikMike 2 days ago 50

Comments (50)

Write comment?

sm1ley

2 days ago, # | 0

typo in D. (n-1)*n/2

→ Reply

rananjay23

2 days ago, # ^ | ← Rev. 2 +9

No that's not typo .Total count of possible substrings is n*(n+1)/2 (including substring of length one)

→ Reply

Yan_Olerinskiy

2 days ago, # ^ | ← Rev. 2 -28

→ Reply

The comment is hidden because of too negative feedback, click here to view it

rananjay23

2 days ago, # ^ | 0

See , substrings of length 1 are bad , so they will be automatically counted in cntBad .So n(n+1)/2 is count of all substrings and cntBad = number of bad substrings .Hence n(n+1)/2 — cntBad is number of good substrings .For example consider "AAA" , here n(n+1)/2 = 6 and number of bad substrings i.e cntBad = 3 (Three substrings "A" of length 1) , thus total number of good substrings is 6 — 3 = 3 i.e the number we get after removing bad substrings from all possible strings .

→ Reply


Yan_Olerinskiy

2 days ago, # ^ | ☆ -8

We ignore them in the beginning, check the author's solution.

→ Reply


MubtasimShahriar

39 hours ago, # ^ | ☆ 0

If we take total substring as n*(n+1)/2 then we have to initialize cntBad = n at first.

And if we take total substring as n*(n-1)/2 then we have to initialize cntBad = 0. Authors solution took cntBad = 0.

So both are right.

→ Reply

tryptophan

2 days ago, # | ☆ ← Rev. 3 -25

Problem A Challenge: Assume, that you can substract a prime from x at most 3 times.

Solution

→ Reply

MrEK

2 days ago, # | ☆ 0

I couldn't understand the explanation of problem C. Can someone describe it?

→ Reply


MubtasimShahriar

40 hours ago, # ^ | ☆ +13

For explanation I'm using a test case:

9 8 5 4 3 1

Do you know what does this mean? Let me explain. It means that the height of the cliff is 9. So initially at 9, there is a platform where the player is standing. At height 8 there is a platform too. But at height 7 and 6 there is no platform by now. And then at height 5 and 4 and 3 there we have platforms at a stretch. Height 2 have no platform. Height 1 has platform.

At first you are at the highest point means at 9. From here we'll will see three kinds of moves. Let's see those:

Case 1: Do both of the next two height have platforms? If yes, then we can and should move to the second using 0 crystal.

Case 2: Do the next height have a platform? if yes, then it is optimal to move to that next height using 1 crystal.

Case 3: Now it means that there is at least one gap between now and the next platform. It doesn't matter if there is one or multiple gap. You can move to the very last gap using no crystal that means 0 crystal.

Now solve our test case: **9 8 5 4 3 1**.

We are now at 9. Here we see **case 2** because only the next height 8 has a platform. So we move to 8 using **1** crystal.

So now we are at 8. Here we see **case 3** because we have gap at heights 7 and 6. So we move to 6 using **0** crystal.

So now we are at 6. Here we see **case 1** because both of the next two heights 5 and 4 contains platform. so we move to height 4 using **0** crystal.

So now we are at 4. Here we see **case 2**. So we move to height 3 using **1** crystal.

So now we are at 3. Here we see **case 3**. So we move to height 2 using **0** crystal.

Now notice one thing. At Ground that means at height 0 we can assume that there is always a platform but it doesn't move at all. Because it's ground :D

So now we are at 2. Here we see **case 1**. So we move to ground using **0** crystal. So in total 2 crystal is used.

We need a crystal only when we meet **case 2**.

I hope you understand.

→ Reply

MrEK

40 hours ago, # ^ | ☆ ← Rev. 2 0

Thank you very much. I understanded it now.

→ Reply

Noob-ita-pro

37 hours ago, # ^ | ☆ 0

Question is just observations:

Observation 1 -> if u have 0 below u, you are good

Observation 2 -> if u have 1 below u and 0 below that 1 u are good

Observation 3 -> if u are on starting block then length of contigous 1s should be odd. eg 1 1 1 1 1

u will land on last 1, if it was 1 1 1 1 0 1 then u will fall.

Observation 4 -> if u are not on starting block u want even length contigous 1s. Why ? coz u

will land on the 0 just above u. for eg. 1 1 1 0 0 0 0 1 1 1 1 0 0 here u will land on 7th 0 and from there on u will land on last 1. 1 1 1 0 0 0 0 1 1 1 0 0 0 this is invalid bocz here u

will fall after getting on 2nd last last 1.

Observation 5 -> if the last stone is on 1st position then u don't need to worry about anything

coz last cell is ground anyway so 1 1 1 0 1 1 1 1 0 and 1 0 0 1 1 1 0 both are right in 1st case

u will land on last 1 and go to ground and in 2nd case u will land on 2nd last 1 but u won't

fall coz last place is ground anyway.

This problem should have contrsuctive tag also :thinking:

→ Reply

MrEK

31 hour(s) ago, # ^ | ☆ 0

Thank you very much

→ Reply

rananjay23

2 days ago, # | ☆ -6

In problem D is tagged as DP .Can some one tell the method to solve it using DP ? We can calculate bad strings using 4 different for loops (corresponding to type of bad string) , any other method to do that with less typing ?

→ Reply

ryan10bansal

2 days ago, # | ☆ ← Rev. 3 0

For problem D, how can there be only 4 patterns? For example the string $ABBAB$ is also a bad string but its pattern is not the same as the 4 given patterns.

→ Reply

tryptophan

2 days ago, # ^ | ☆ ← Rev. 2 +5

```
ABBA.
```

```
..BAB
```

=> string is good

→ Reply

ryan10bansal

2 days ago, # ^ | ☆ ← Rev. 2 0

Oh thanks. I didn't read the question carefully, my bad.

→ Reply

Bekh

2 days ago, # | ☆ 0

Constraints are really deceiving in E. Could somebody explain an O($a * 2^a$) solution?

→ Reply

Nson

47 hours ago, # ^ | ☆ 0

The a^2 factor inside the dp is for computing the sums of $cnt_{x,y}$ for $x \in mask$ and $y \notin mask$. You can improve this with sos dp, for each mask you compute how much should be added to.

→ Reply

Bekh

47 hours ago, # ^ | ☆ 0

The standard dp sos i'm thinking about is $a2^a$ for each character, which makes it $a^2 2^a$ in total. Could you explain further the recurrence of your dp?

→ Reply

Nson

47 hours ago, # ^ | ☆ 0

I solved in $O(2^a * a^2)$ in the contest. I just coded the same idea in $O(2^a * a)$ with sos dp.

→ Reply

mohamedeltair

43 hours ago, # ^ | ☆ +7

If you calculate the cost in DP by using the newly added letter's position:

Let $co[i][j]$ be the number of times letters i and j appeared next to each other. You can build an array arr where $arr[i][mask] = \sum_{k \in mask} co[i][k]$ in $O(m * 2^m)$, by using this trick: $arr[i][mask] = arr[i][mask \oplus 2^j] + co[i][j]$ for $mask > 0$, where j is any set bit's position in $mask$ (can be the lowest significant bit for example).

→ Reply

Bekh

31 hour(s) ago, # ^ | ☆ +5

Got it. Thanks!

→ Reply

32 hours ago, # ^ | ☆ +5

Bekh

31 hour(s) ago, # ^ | ☆ 0

Thank you.

→ Reply

ArshiaDadras

15 hours ago, # ^ | ☆ 0

You're welcome! :)

But in my code I just used another for in calculating my dp every time I fix the *c*, so my code order is *O(m^2.2^m)*, too! but you can easily change it.

→ Reply

LunaticPriest

46 hours ago, # | ☆ 0

In Problem D, I'm having trouble counting the number of such substrings. I read a lot of different codes but I don't seem to get it. Can somebody please explain their own method to me? Thanks a lot in advance.

→ Reply

satya1998

40 hours ago, # ^ | ☆ ← Rev. 2 0

let our string is AABBAAABAA here we have to find answer for AABB and BBAAA and AAAB and BAA. total of there will be answer for our string . And answer for string AAABB will be 3+2-1. We are subtracting 1 because we are adding the conjution (AB) twice . so answer for string AABBAAABAA will be (2+2-1)+(2+3-1)+(3+1-1)+(1+2-1)=12.

→ Reply

mohamedeltair

44 hours ago, # | ☆ +11

A proof for why the greedy algorithm works in C:

First note that for any $v > 1$, you have to stand on at least v or $v - 1$, because if you didn't stand on both this means that you descended from $a > v$ to $b < v - 1$, which means you are dead.

You are initially standing at h, assume you will walk without any crystals until you first reach that $x > 2$ where $x - 1$ is moved out and $x - 2$ is hidden. Now you must use at least 1 crystal to continue. You can use it either to:

1. move out the platform at $x - 2$, then descend to it directly.
2. hide the one at $x - 1$ and move to it, where you will still have the opportunity to continue to $x - 2$ without extra crystals, so this option is better.

The other scenario you can follow starting from h, is that which would end on

$x - 1$ (without going to x), and this scenario will cost at least 1 crystal. So we deduce that the best option is option 2 in the previous scenario. Starting from $x - 1$, you can continue to use the same followed logic starting from h.
→ Reply


221405

41 hour(s) ago, # | ☆ 0

in problem E, when iterating to new character x then how we calculate dp[mask|(1<<x)] ?
→ Reply


amolpratap007

38 hours ago, # | ☆ 0

I have tried an O(m^2*2^m) solution for problem E but it is giving TLE.
→ Reply


Ventus

38 hours ago, # | ☆ 0

I cannot understand the solution of problem E. Specially, "If letter y is already on current keyboard, then we should add to answer cntx,y(posx−posy), and cntx,y(posy−posx) otherwise" sentence. Why should we consider the letter y which is not included in the keyboard? And on last formula,∑y∈msk(cntx,yposx)−∑y∉msk(cntx,yposx), why there is no posy? Where do they go?
→ Reply


Ventus

37 hours ago, # ^ | ☆ 0

Ah Sorry, I understand.
→ Reply


akiradeveloper

37 hours ago, # ^ | ☆ 0

How do you erase cntx,y*posy? Can you explain? I think it is only possible when the two summations have the same range.
→ Reply


Leks360

33 hours ago, # ^ | ☆ 0

care to explain please
→ Reply


ichiro

35 hours ago, # | ☆ +24

Can any one explain how to solve D if there are 26 character?
→ Reply


vrishinvv

34 hours ago, # | ☆ +1

Can someone explain how to solve D with 'binary search'?
→ Reply


white__walker

34 hours ago, # | ☆ +5

How to solve D using DP?
→ Reply


roll_no_1

32 hours ago, # | ☆ +1

Just a side fact: The resultant tree that gets formed in the problem F is also called Caterpillar Tree.
→ Reply

pk842

31 hour(s) ago, # | ☆ ← Rev. 2 0

can someone please explain the structure of tree formed in problem F? I didn't get it from editorial completely.

→ Reply

Manas_For_WF

19 hours ago, # ^ | ☆ 0

It's a caterpillar: https://en.wikipedia.org/wiki/Caterpillar_tree

→ Reply

pk842

6 hours ago, # ^ | ☆ 0

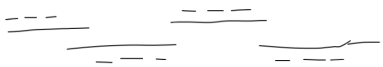
Thank you for your reply! But I am asking can you please explain how we land on this structure?

→ Reply

Manas_For_WF

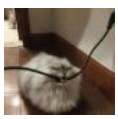
3 hours ago, # ^ | ☆ ← Rev. 2 0

One major point here is that an integer can't be contained in three or more segments, since that would correspond to a cycle in the described graph, but we're given a tree, so that's ruled out. With this knowledge, let's construct some intervals so that we can maximise the number of intervals. We'll try to have two segments containing an integer as long as it is possible, since we're trying to maximise the size of the required subgraph.



So we have a continuous sequence of intersecting intervals, and then for points that are not yet covered by two intervals, we're trying to find as many non-intersecting intervals as possible to cover those points (you may even think of them as degenerate intervals, where l = r, if you want, since we're free to size our intervals as we please). The only thing that limits us is the given tree, where the smaller intervals can only be as many as the count of neighbours except the next interval in the path. I hope you see why this is optimal, and why this would correspond to the caterpillar graph.

→ Reply



actual_truthseeker

6 hours ago, # ^ | ☆ 0

it's a DICK

→ Reply

harsh1599

17 hours ago, # | ☆ 0

Why is there a "Binary Search" Tag on Problem D? How to solve it using Binary Search?

→ Reply

lonely_moon

17 hours ago, # | ☆ +3

How to understand the meaning of the dp array in E problem?

→ Reply

Tommyr7

14 hours ago, # | ☆ ← Rev. 2 0

Problem fixed.

Sorry for inconvenience.

→ Reply

ichiro

14 hours ago, # | +10

Read the statement carefully.When you will explosion on x=3 you will get 6 7 7 7

Then every monster is either killed by explosion or pushed away.

→ Reply

wol

14 hours ago, # | +10

if you'll fire at c=3 then positions will become 6,7,7,7

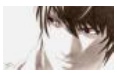
→ Reply

actual_truthseeker

6 hours ago, # | -15

no one cares do you realize that
stop trying to be relevant

→ Reply



_greymatter

5 hours ago, # | ← Rev. 6 +1

PikMike, in problem E, could you explain how you arrived at the final formula

$$\sum_{y \in msk} (cnt_{x,y} pos_x) - \sum_{y \notin msk} (cnt_{x,y} pos_x)$$

→ Reply

Codeforces (c) Copyright 2010-2019 Mike Mirzayanov
The only programming contests Web 2.0 platform
Server time: Oct/12/2019 01:25:40 UTC+5.5 (e2).
Desktop version, switch to mobile version.
Privacy Policy

Supported by

ITMO UNIVERSITY