Errichto's blog

Codeforces Round #356 — Editorial

By **Errichto**, 12 hours ago,

680A - Bear and Five Cards

Iterate over all pairs and triples of numbers, and for each of them check if all two/three numbers are equal. If yes then consider the sum of remaining numbers as the answer (the final answer will be the minimum of considered sums). Below you can see two ways to implement the solution.

code1

```
#include<bits/stdc++.h>
using namespace std;
int main() {
	int t[5];
	int s = 0;
	for(int i = 0; i < 5; ++i) {
		scanf("%d", &t[i]);
		s += t[i];
	}
	int best = s;

	// discard 2 cards
	for(int a = 0; a < 5; ++a)
		for(int b = a + 1; b < 5; ++b)
			if(t[a] == t[b])
				best = min(best, s - 2 * t[a]);

	// or discard 3 cards
	for(int a = 0; a < 5; ++a)
		for(int b = a + 1; b < 5; ++b)
			for(int c = b + 1; c < 5; ++c)
				if(t[a] == t[b] && t[a] == t[c])
					best = min(best, s - 3 * t[a]);

	printf("%dn", best);
	return 0;
}
```

code2

```
#include<bits/stdc++.h>
using namespace std;
int main() {
	int t[5];
	for(int i = 0; i < 5; ++i)
		scanf("%d", &t[i]);
	sort(t, t + 5);
	int best_remove = 0;
	for(int i = 0; i < 5; ++i) {
		if(i + 1 < 5 && t[i] == t[i+1])
			best_remove = max(best_remove, 2 * t[i]);
```

```
                if(i + 2 < 5 && t[i] == t[i+2])
                        best_remove = max(best_remove, 3 * t[i]);
        }
        printf("%dn", t[0]+t[1]+t[2]+t[3]+t[4]-best_remove);
        return 0;
}
```

680B - Bear and Finding Criminals

Limak can't catch a criminal only if there are two cities at the same distance and only one of them contains a criminal. You should iterate over the distance and for each distance d check if $a - d$ and $a + d$ are both in range $[1, n]$ and if only one of them has $t_i = 1$.

code1

```
#include<bits/stdc++.h>
using namespace std;
const int nax = 1005;
int t[nax];
int main() {
        int n, a;
        scanf("%d%d", &n, &a);
        for(int i = 1; i <= n; ++i)
                scanf("%d", &t[i]);
        int answer = 0;
        for(int i = 1; i <= n; ++i) if(t[i]) {
                // can we catch criminal in city i?
                int distance = i - a; // distance from a
                int j = a - distance; // the other city at the same distance
                if(j < 1 || j > n || t[i] == t[j])
                        ++answer;
        }
        printf("%dn", answer);
        return 0;
}
```

code2

```
#include<bits/stdc++.h>
using namespace std;
const int nax = 1005;
int t[nax];
bool impossible[nax];
int main() {
        int n, a;
        scanf("%d%d", &n, &a);
        for(int i = 1; i <= n; ++i)
                scanf("%d", &t[i]);
        for(int i = 1; i <= n; ++i)
                for(int j = i + 1; j <= n; ++j)
                        if(abs(i - a) == abs(j - a) && t[i] != t[j]) {
                                // i and j have the same distance to a
                                // also, there is a criminal in exactly one
of them
                                impossible[i] = impossible[j] = true;
                        }
        int answer = 0;
        for(int i = 1; i <= n; ++i)
                if(t[i] == 1 && !impossible[i])
                        ++answer;
        printf("%dn", answer);
        return 0;
}
```

Find

→ Recent actions

Errichto → Codeforces Round #356

dragoon → IPSC & Topcoder Open 2C Back to back?

hepaxorm → need some help!!!

wolf_sothe → Are you majoring a unique subject for coders?

Errichto → Codeforces Round #356 — Editorial

rahulm_1997 → confusion regarding online judge

chuka231 → CodeForces Starting Time is very hard for Asian Folks :(

MikeMirzayanov → Interactive Problems: Guide for Participants

Bredor → Unbelieveable story of success at ACM ICPC final

Um_nik → Codeforces Round #315 Editorial

rachitjain → Codeforces giving wrong judgements

praran26 → Strange behavior of Codeforces API

late_riser → Request for arranging contest on weekends

Nieelawn → Ideleness limit exceeded

vatsalsharma376 → Bit Manipulation Problems

avirammagen11 → Settlers of Catan nwerc 2009 help

ilex → Help with TLE on CF 355 problem D

Edvard → Editorial of Educational Codeforces Round 11

Wild_Hamster → Codeforces Round #355 (Div. 2) Editorial

ivplay → suffix array related problems

AndrewLazarev → Codeforces Beta Round #2 - Tutorial

Egor → CHelper manual

fcspartakm → Codeforces Round #288 (Div.2) Editorial

Al.Cash → Efficient and easy segment trees

Swistakk → Distributed Code Jam Practice Round is on!

Detailed →

679A - Bear and Prime 100

If a number is composite then it's either divisible by p^2 for some prime p, or divisible by two distinct primes p and q. To check the first condition, it's enough to check all possible p^2 (so, numbers $4, 9, 25, 49$). If at least one gives "yes" then the hidden number if composite.

If there are two distinct prime divisors p and q then both of them are at most 50 — otherwise the hidden number would be bigger than 100 (because for $p \geq 2$ and $q > 50$ we would get $p \cdot q > 100$). So, it's enough to check primes up to 50 (there are 15 of them), and check if at least two of them are divisors.

code1

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

bool isPrime(int a) {
        for(int i = 2; i * i <= a; ++i)
                if(a % i == 0)
                        return false;
        return true;
}

bool divisible(int a) {
        printf("%dn", a);
        fflush(stdout);
        char sl[10];
        scanf("%s", sl);
        return sl[0] == 'y' || sl[0] == 'Y';
}

int HIGH = 100;

int main() {
        int counter = 0; // we print "composite" if counter >= 2
        for(int a = 2; a <= HIGH/2 && counter < 2; ++a)
                if(isPrime(a))
                        if(divisible(a)) {
                                ++counter;
                                if(a * a <= HIGH && divisible(a * a))
                                        ++counter;
                        }
        puts(counter >= 2 ? "composite" : "prime");
        return 0;
}
```

code2. Python

```
from sys import stdout

PRIMES = [x for x in range(2,100) if 0 not in [x%i for i in range(2,x)]]

NORMAL = PRIMES[:15]
SQUARES = [x*x for x in PRIMES[:4]]

for ele in SQUARES:
    print(ele)
    stdout.flush()
    x = input()
    if x == "yes":
        print("composite")
        exit(0)

yes = 0
for ele in NORMAL:
```

```
    print(ele)
    stdout.flush()
    x = input()
    if x == "yes":
        yes += 1

print("prime" if yes < 2 else "composite")
```

679B - Bear and Tower of Cubes

Let's find the maximum a that $a^3 \leq m$. Then, it's optimal to choose X that the first block will have side a or a - 1. Let's see why.

- If the first block has side a then we are left with $m_2 = m$ - $first\_block = m$ - a^3.
- If the first block has side a - 1 then the initial X must be at most a^3 - 1 (because otherwise we would take a block with side a), so we are left with $m_2 = a^3$ - 1 - $first\_block = a^3$ - 1 - $(a$ - $1)^3$
- If the first blocks has side a - 2 then the initial X must be at most $(a$ - $1)^3$ - 1, so we are left with $m_2 = (a$ - $1)^3$ - 1 - $first\_block = (a$ - $1)^3$ - 1 - $(a$ - $2)^3$.

We want to first maximize the number of blocks we can get with new limit m_2. Secondarily, we want to have the biggest initial X. You can analyze the described above cases and see that the first block with side $(a$ - $2)^3$ must be a worse choice than $(a$ - $1)^3$. It's because we start with smaller X and we are left with smaller m_2. The situation for even smaller side of the first block would be even worse.

Now, you can notice that the answer will be small. From m of magnitude a^3 after one block we get m_2 of magnitude a^2. So, from m we go to $m^{2/3}$, which means that the answer is $O(loglog(m))$. The exact maximum answer turns out to be 18.

The intended solution is to use the recursion and brutally check both cases: taking a^3 and taking $(a$ - $1)^3$ where a is maximum that $a^3 \leq m$. It's so fast that you can even find a in $O(m^{1/3})$, increasing a by one.

code1

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

pair<ll,ll> best;

ll my_pow(ll x) { return x * x * x; }

void rec(ll m, ll steps, ll subtracted) {
        if(m == 0) {
                best = max(best, make_pair(steps, subtracted));
                return;
        }
        ll x = 1;
        while(my_pow(x+1) <= m) ++x;
        rec(m - my_pow(x), steps+1, subtracted + my_pow(x));
        if(x - 1 >= 0)
                rec(my_pow(x)-1-my_pow(x-1), steps+1, subtracted + my_pow(x-
1));
}

int main() {
        ll m;
        scanf("%lld", &m);
        rec(m, 0, 0);
        printf("%lld %lldn", best.first, best.second);
        return 0;
}
```

code2

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

ll my_pow(ll x) { return x * x * x; }

ll steps(ll m) { // works in O(m^(1/3))
        if(m <= 7) return m;
        ll x = 1;
        while(my_pow(x+1) <= m) ++x;
        return 1 + steps(max(m - my_pow(x), my_pow(x)-1-my_pow(x-1)));
}

int main() {
        ll m;
        scanf("%lld", &m);
        ll subtracted = 0, steps_so_far = 0;
        while(m) {
                ++steps_so_far;
                ll x = 1;
                while(my_pow(x+1) <= m) ++x;
                if(steps(m) == 1 + steps(m - my_pow(x))) {
                        m -= my_pow(x);
                        subtracted += my_pow(x);
                }
                else {
                        m = my_pow(x) - 1 - my_pow(x-1);
                        subtracted += my_pow(x-1);
                }
        }
        printf("%lld %lldn", steps_so_far, subtracted);
        return 0;
}
```

code3

```
#include "bits/stdc++.h"

using namespace std;

#define ll long long

vector <ll> X;
map <ll, pair <int, ll> > M;

pair <int, ll> go(ll x)
{
  if(x <= 1)
    return {x,x};
  if(M.find(x) != M.end())
    return M[x];
  int i = upper_bound(X.begin(), X.end(), x) - X.begin() - 1;
  int p1, p2;
  long long q1, q2;
  tie(p1, q1) = go(x - X[i]);
  tie(p2, q2) = go(X[i] - 1);
  return M[x] =  max(make_pair(p1+1, q1+X[i]), {p2, q2});
}

int main()
{
  for(int i=0; i<=1e5+7; i++)
    X.push_back(1LL*i*i*i);
```

```
  ll x;
  cin >> x;
  int a;
  long long b;
  tie(a,b) = go(x);
  cout << a << " " << b << "n";
}
```

679C - Bear and Square Grid

Let's first find CC's (connected components) in the given grid, using DFS's.

We will consider every possible placement of a $k \times k$ square. When the placement is fixed then the answer is equal to the sum of k^2 the the sum of sizes of CC's touching borders of the square (touching from outside), but for those CC's we should only count their cells that are outside of the square — not to count something twice. We will move a square, and at the same time for each CC we will keep the number of its cells outside the square.

We will used a sliding-window technique. Let's fix row of the grid — the upper row of the square. Then, we will first place the square on the left, and then we will slowly move a square to the right. As we move a square, we should iterate over cells that stop or start to belong to the square. For each such empty cell we should add or subtract 1 from the size of its CC (ids and sizes of CC's were found at the beginning).

And for each placement we consider, we should iterate over outside borders of the square ($4k$ cells — left, up, right and down side) and sum up sizes of CC's touching our square. Be careful to not count some CC twice — you can e.g. keep an array of booleans and mark visited CC's. After checking all $4k$ cells you should clear an array, but you can't do it in O(number_of_all_components) because it would be too slow. You can e.g. also add visited CC's to some vector, and later in the boolean array clear only CC's from the vector (and then clear vector).

The complexity is $O(n^2 \cdot k)$.

code1

```
#include<bits/stdc++.h>
using namespace std;
const int nax = 505;
int n;
char grid[nax][nax]; // input
int cc[nax][nax]; // id of CC in which this cell is
int cc_size[nax*nax]; // size of CC
int when_added[nax*nax];

const int dx[4] = {-1, 1, 0, 0};
const int dy[4] = {0, 0, -1, 1};
const char EMPTY = '.';

bool inside(int x, int y) {
        return 0 <= min(x, y) && max(x, y) < n;
}

void dfs(int x, int y, int which_cc) {
        cc[x][y] = which_cc;
        ++cc_size[which_cc];
        for(int i = 0; i < 4; ++i) { // iterate of 4 adjacent cells
                int x2 = x + dx[i];
                int y2 = y + dy[i];
                if(inside(x2, y2) && grid[x2][y2] == EMPTY && cc[x2][y2] == 0)
                        dfs(x2, y2, which_cc);
        }
}

void add(int x, int y, int & answer, int current_time) {
```

```
        if(inside(x, y) && grid[x][y] == EMPTY) {
                int id = cc[x][y];
                if(when_added[id] != current_time) {
                        when_added[id] = current_time;
                        answer += cc_size[id];
                }
        }
}

int main() {
        int k;
        scanf("%d%d", &n, &k);
        for(int i = 0; i < n; ++i)
                scanf("%s", grid[i]);

        // run DFS many times to find CC's (connected components)
        int how_many_cc = 0;
        for(int x = 0; x < n; ++x)
                for(int y = 0; y < n; ++y)
                        if(grid[x][y] == EMPTY && cc[x][y] == 0)
                                dfs(x, y, ++how_many_cc);

        int cur_time = 1;
        int best_answer = 0;

        for(int y_low = 0; y_low + k <= n; ++y_low) {
                // first we put a square with corner in (0, y_low)
                for(int x = 0; x < k; ++x)
                        for(int y = y_low; y < y_low + k; ++y)
                                --cc_size[cc[x][y]]; // subtract cells inside
a square

                for(int x_low = 0; x_low  + k <= n; ++x_low) {
                        int answer = k * k; // all cells inside a square
                        // consider one row: below, above, left, right
                        for(int x = x_low; x < x_low + k; ++x) {
                                add(x, y_low - 1, answer, cur_time);
                                add(x, y_low + k, answer, cur_time);
                        }
                        for(int y = y_low; y < y_low + k; ++y) {
                                add(x_low - 1, y, answer, cur_time);
                                add(x_low + k, y, answer, cur_time);
                        }
                        ++cur_time;
                        best_answer = max(best_answer, answer);

                        if(x_low + k != n) {
                                // move a square to increase x_low by 1
                                for(int y = y_low; y < y_low + k; ++y) {
                                        ++cc_size[cc[x_low][y]]; // remove
cells with x = x_low

                                        --cc_size[cc[x_low+k][y]]; // insert
cells with x = x_low+k
                                }
                        }
                }

                for(int x = n - k; x < n; ++x)
                        for(int y = y_low; y < y_low + k; ++y)
                                ++cc_size[cc[x][y]]; // we don't need cells
inside to be subtracted
        }
        printf("%dn", best_answer);
        return 0;
```

```
}
```

code2

```
#include<bits/stdc++.h>
using namespace std;
#define FOR(i,a,b) for(int i = (a); i <= (b); ++i)
#define FORD(i,a,b) for(int i = (a); i >= (b); --i)
#define RI(i,n) FOR(i,1,(n))
#define REP(i,n) FOR(i,0,(n)-1)
#define mini(a,b) a=min(a,b)
#define maxi(a,b) a=max(a,b)
#define mp make_pair
#define pb push_back
#define st first
#define nd second
#define sz(w) (int) w.size()
typedef vector<int> vi;
typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
const int inf = 1e9 + 5;
const int nax = 500 + 5;
const int nax2 = nax * nax;

int n,k;

int ile_na_zew, akt_stan = 1;

char t[nax][nax];
int ojc[nax][nax];
int ilosc[nax2], lewo[nax2], prawo[nax2], gora[nax2], dol[nax2];
int suma[nax][nax], kol[nax], stan[nax2];
bool bylo[nax][nax];

vector<pair<pii, int> > wiersz[nax];

int ruchi[] = {-1, 0, 1, 0};
int ruchj[] = {0, -1, 0, 1};

void dfs(int i, int j, int wsk) {
  ilosc[wsk]++;
  ojc[i][j] = wsk;
  bylo[i][j] = true;
  lewo[wsk] =  min(lewo[wsk], j);
  prawo[wsk] = max(prawo[wsk], j);
  gora[wsk] =  min(gora[wsk], i);
  dol[wsk] =   max(dol[wsk], i);

  REP(g,4) {
    int ni = i + ruchi[g];
    int nj = j + ruchj[g];

    if (t[ni][nj] == '.' && bylo[ni][nj] == false)
      dfs(ni,nj,wsk);
  }
}

void check(int o) {
  if (stan[o] != akt_stan)
    ile_na_zew += ilosc[o];
  stan[o] = akt_stan;
}
```

```
int main() {
  scanf("%d%d",&n,&k);
  FOR(i,1,n) scanf(" %s",t[i]+1);

  int wsk = 1;
  FOR(i,1,n) FOR(j,1,n) if (t[i][j] == '.' && !bylo[i][j]) {
    lewo[wsk] = prawo[wsk] = j;
    gora[wsk] = dol[wsk] = i;
    dfs(i,j,wsk);
    //printf("%dn",ilosc[wsk]);
    if (prawo[wsk] - lewo[wsk] + 1 <= k && dol[wsk] - gora[wsk] + 1 <= k) {
      //puts("DD");
      int x = max(1, dol[wsk] - k + 1);
      wiersz[x].pb(mp(mp(lewo[wsk], prawo[wsk]), ilosc[wsk]));
      wiersz[gora[wsk]+1].pb(mp(mp(lewo[wsk], prawo[wsk]), -ilosc[wsk]));
    }
    ++wsk;
  }

  FOR(i,1,n) FOR(j,1,n) suma[i][j] = suma[i-1][j] + suma[i][j-1] - suma[i-1]
[j-1] + (t[i][j] == '.');
  //FOR(i,1,n) FOR(j,1,n) printf("(%d,%d): %dn",i,j,suma[i][j]);

  int res = 0;
  FOR(i,1,n-k+1) {
    for (auto p: wiersz[i]) {
      int l = max(1, p.st.nd - k + 1);
      //printf("%d %d %dn",l,p.st.st,p.nd);
      kol[l] += p.nd;
      kol[p.st.st + 1] -= p.nd;
    }

    int suma_s = 0;
    FOR(j,1,n-k+1) {
      suma_s += kol[j];
      int ile_w_s = suma[i+k-1][j+k-1] - suma[i-1][j+k-1] - suma[i+k-1][j-1] +
suma[i-1][j-1];
      ile_na_zew = 0;
      REP(g,k) {
        check(ojc[i-1][j+g]);
        check(ojc[i+k][j+g]);
        check(ojc[i+g][j-1]);
        check(ojc[i+g][j+k]);
      }

      int x = ile_na_zew + (k * k  - ile_w_s) + suma_s;
      //printf("(%d,%d) : %d %d %d %dn",i,j,ile_na_zew, ile_w_s, suma_s, x);

      res = max(res, x);
      ++akt_stan;
    }
  }
  printf("%dn",res);
        return 0;
}
```

code3, Java

```
import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.List;
```

```
import java.util.StringTokenizer;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.io.InputStream;

/**
 * Built using CHelper plug-in
 * Actual solution is at the top
 *
 * @author AlexFetisov
 */
public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        TaskE_356 solver = new TaskE_356();
        solver.solve(1, in, out);
        out.close();
    }

    static class TaskE_356 {
        int n;
        int k;
        boolean[][] f;
        List<Integer> componentCellCount;
        int[][] color;
        int x = 0;
        int y = 0;
        int totalEmpty = 0;
        int sumInComponents = 0;
        int[] amInComponents;
        int[] flag;
        int currentFlagColor = 0;

        public void solve(int testNumber, InputReader in, PrintWriter out) {
            n = in.nextInt();
            k = in.nextInt();
            f = new boolean[n][n];
            color = new int[n][n];
            ArrayUtils.fill(color, -1);
            for (int i = 0; i < n; ++i) {
                char[] c = in.nextString().toCharArray();
                for (int j = 0; j < n; ++j) {
                    f[i][j] = (c[j] == '.');
                }
            }

            // Caclulate all CC
            int res = 0;
            componentCellCount = new ArrayList<Integer>();
            int currentComponentId = 0;
            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    if (color[i][j] == -1) {
                        int count = dfs(i, j, currentComponentId++);
                        componentCellCount.add(count);
                        res = Math.max(res, count);
                    }
                }
            }
```

```
            flag = new int[currentComponentId];
            amInComponents = new int[currentComponentId];
            // Preprocess first square
            for (int i = 0; i < k; ++i) {
                for (int j = 0; j < k; ++j) {
                    addCell(i, j);
                }
            }

            res = Math.max(res, checkResult());

            for (int i = 0; i <= n - k; ++i) {
                int delta = i % 2 == 0 ? 1 : -1;
                for (int j = 0; j < n - k; ++j) {
                    moveSquare(0, x, y, x, y + delta);
                    y += delta;
                    res = Math.max(res, checkResult());
                }
                if (i != n - k) {
                    moveSquare(1, x, y, x + 1, y);
                    ++x;
                    res = Math.max(res, checkResult());
                }
            }
            out.println(res);
        }

        void moveSquare(int type, int cx, int cy, int nx, int ny) {
            if (type == 0) {
                if (ny < cy) {
                    for (int i = 0; i < k; ++i) {
                        removeCell(cx + i, cy + k - 1);
                        addCell(cx + i, ny);
                    }
                } else {
                    for (int i = 0; i < k; ++i) {
                        removeCell(cx + i, cy);
                        addCell(cx + i, ny + k - 1);
                    }
                }
            } else {
                for (int i = 0; i < k; ++i) {
                    removeCell(cx, cy + i);
                    addCell(nx + k - 1, cy + i);
                }
            }
        }

        void removeCell(int cx, int cy) {
            if (f[cx][cy]) {
                --totalEmpty;
                amInComponents[color[cx][cy]]--;
                if (amInComponents[color[cx][cy]] == 0) {
                    sumInComponents -= componentCellCount.get(color[cx][cy]);
                }
            }
        }

        void addCell(int cx, int cy) {
            if (f[cx][cy]) {
                ++totalEmpty;
                amInComponents[color[cx][cy]]++;
                if (amInComponents[color[cx][cy]] == 1) {
```

```
                sumInComponents += componentCellCount.get(color[cx][cy]);
            }
        }
    }

    int dfs(int cx, int cy, int cId) {
        if (cx < 0 || cx >= n || cy < 0 || cy >= n) return 0;
        if (color[cx][cy] != -1) return 0;
        if (!f[cx][cy]) return 0;
        color[cx][cy] = cId;
        int am = 1;
        am += dfs(cx - 1, cy, cId);
        am += dfs(cx + 1, cy, cId);
        am += dfs(cx, cy - 1, cId);
        am += dfs(cx, cy + 1, cId);
        return am;
    }

    int checkResult() {
        ++currentFlagColor;
        int res = k * k - totalEmpty + sumInComponents;
        for (int i = 0; i < k; ++i) {
            res += checkCell(x - 1, y + i);
            res += checkCell(x + k, y + i);
            res += checkCell(x + i, y - 1);
            res += checkCell(x + i, y + k);
        }
        return res;
    }

    int checkCell(int cx, int cy) {
        if (cx < 0 || cx >= n || cy < 0 || cy >= n) return 0;
        if (!f[cx][cy]) return 0;
        if (amInComponents[color[cx][cy]] > 0) return 0;
        if (flag[color[cx][cy]] == currentFlagColor) return 0;
        flag[color[cx][cy]] = currentFlagColor;
        return componentCellCount.get(color[cx][cy]);
    }

}

static class ArrayUtils {
    public static void fill(int[][] f, int value) {
        for (int i = 0; i < f.length; ++i) {
            Arrays.fill(f[i], value);
        }
    }

}

static class InputReader {
    private BufferedReader reader;
    private StringTokenizer stt;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream));
    }

    public String nextLine() {
        try {
            return reader.readLine();
        } catch (IOException e) {
            return null;
        }
```

```
        }

        public String nextString() {
            while (stt == null || !stt.hasMoreTokens()) {
                stt = new StringTokenizer(nextLine());
            }
            return stt.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(nextString());
        }

    }
}
```

679D - Bear and Chase

Check my code below, because it has a lot of comments.

First, in $O(n^3)$ or faster find all distances between pairs of cities.

Iterate over all $g1$ — the first city in which you use the BCD. Then, for iterate over all $d1$ — the distance you get. Now, for all cities calculate the probability that Limak will be there in the second day (details in my code below). Also, in a vector `interesting` let's store all cities that are at distance $d1$ from city $g1$.

Then, iterate over all $g2$ — the second city in which you use the BCD. For cities from `interesting`, we want to iterate over them and for each distinct distance from $g2$ to choose the biggest probability (because we will make the best guess there is).

Magic: the described approach has four loops (one in the other) but it's $O(n^3)$.

Proof is very nice and I encourage you to try to get it yourself.

Proof here

After fixing g1 divide cities by their distance from g1. Then, when we get distance d1 in the first day, then in the second day all possible cities are at distance d1-1, d1 and d1+1. So, we will consider each city at most three times.

code1

```
#include<bits/stdc++.h>
using namespace std;
#define FOR(i,a,b) for(int i = (a); i <= (b); ++i)
typedef double T;
const int nax = 1005;
int dist[nax][nax];
vector<int> w[nax];
T p_later[nax]; // p[v] - probability for city v in the second day
T p_dist_max[nax];
bool vis[nax];

void max_self(T & a, T b) {
        a = max(a, b);
}

T consider_tomorrow(int n, int g1, int dist1) {
        T best_tomorrow = 0;
        // we need complexity O(n * x)
        // where x denotes the number of v that |dist1-dist[g1][v]| <= 1
        for(int i = 1; i <= n; ++i) {
                p_later[i] = 0;
                vis[i] = false;
```

```
        }
        vector<int> interesting;
        for(int v = 1; v <= n; ++v) if(dist[g1][v] == dist1)
                for(int b : w[v]) {
                        // Limak started in v with prob. 1/n
                        // he then moved to b with prob. 1/degree[v]
                        p_later[b] += (T) 1 / n / w[v].size();
                        if(!vis[b]) {
                                vis[b] = true;
                                interesting.push_back(b);
                        }
                }

        // interesting.size() <= x, where x is defined above (needed for
complexity)
        for(int g2 = 1; g2 <= n; ++g2) {
                T local_sum = 0; // over situations with fixed g1, dist1, g2

                for(int b : interesting)
                        max_self(p_dist_max[dist[g2][b]], p_later[b]);
                for(int b : interesting) {
                        local_sum += p_dist_max[dist[g2][b]];
                        p_dist_max[dist[g2][b]] = 0; // so it won't be
calculated twice
                }
                max_self(best_tomorrow, local_sum);
        }
        return best_tomorrow;
}

int main() {
        int n, m;
        scanf("%d%d", &n, &m);
        FOR(i,1,n) FOR(j, 1, n) if(i != j)
                dist[i][j] = n + 1; // infinity
        FOR(i,1,m) {
                int a, b;
                scanf("%d%d", &a, &b);
                w[a].push_back(b);
                w[b].push_back(a);
                dist[a][b] = dist[b][a] = 1;
        }
        // Floyd-Warshall
        FOR(b,1,n)FOR(a,1,n)FOR(c,1,n)
                dist[a][c] = min(dist[a][c], dist[a][b] + dist[b][c]);

        // g1 is the first guess
        T answer = 0;
        FOR(g1, 1, n) {
                T sum_over_dist1 = 0;
                FOR(dist1, 0, n) {
                        int cnt_cities = 0;
                        FOR(i, 1, n) if(dist[g1][i] == dist1)
                                ++cnt_cities;
                        if(cnt_cities == 0) continue; // there are no cities
within distance dist1

                        // 1) consider guessing immediately
                        T immediately = (T) 1 / n; // how much it counts
towards the answer
                        // 2) consider waiting for tomorrow
                        T second_day = consider_tomorrow(n, g1, dist1);
                        sum_over_dist1 += max(immediately, second_day);
                }
```

```
                max_self(answer, sum_over_dist1);
        }
        printf("%.12lfn", (double) answer);
        return 0;
}
```

679E - Bear and Bad Powers of 42

The only special thing in numbers $1, 42, ...$ was that there are only few such numbers (in the possible to achieve range, so up to about 10^{14}).

Let's first solve the problem without queries "in the interval change all numbers to x". Then, we can make a tree with operations (possible with lazy propagation):

- add on the interval
- find minimum in the whole tree

In a tree for each index $i \in [1, n]$ let's keep the distance to the next power of 42. After each "add on the interval" we should find the minimum and check if it's positive. If not then we should change value of the closest power of 42 for this index, and change the value in the tree. Then, we should again find the minimum in the tree, and so on. The amortized complexity is $O((n+q) * log(n) * log_{42}(values))$. It can be proved that numbers won't exceed $(n+q) * 1e9 * log$.

Now let's think about the remaining operation of changing all interval to some value. We can set only one number (the last one) to the given value, and set other values to INF. We want to guarantee that if $t[i] \neq t[i+1]$ then the i-th value is correctly represented in the tree. Otherwise, it can be INF instead (or sometimes it may be correctly represented, it doesn't bother me). When we have the old query of type "add something to interval $[a, b]$" then if index a - 1 or index b contains INF in the tree then we should first retrieve the true value there. You can see that each operation changes $O(1)$ values from INF to something finite. So, the amortized complexity is still $O((n+q) * log(n) * log_{42}(values))$.

One thing regarding implementation. In my solution there is "$set<int> interesting$" containing indices with INF value. I think it's easier to implemement the solution with this set.

code1

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int nax = 1e6 + 5;
const int pot = 256 * 1024;
const ll INF = 3e18L;
ll cur_power[nax]; // the least power of 42 larger than the current value
// true_value + remaining = cur_power,  where remaining=tr[pot+i]

struct Node {
        ll local;
        ll lazy;
        ll smallest() { return local + lazy; } // smallest value in this
subtree
} tr[2*pot];

void propagate(int i) {
        assert(i < pot);
        tr[2*i].lazy += tr[i].lazy;
        tr[2*i+1].lazy += tr[i].lazy;
        tr[i].local += tr[i].lazy;
        tr[i].lazy = 0;
}
void act(int i) {
        assert(i < pot);
        assert(tr[i].lazy == 0);
        tr[i].local = min(tr[2*i].smallest(), tr[2*i+1].smallest());
}
```

```
void change(int i, int low, int high, int q_low, int q_high, ll val) {
        if(q_low <= low && high <= q_high) {
                tr[i].lazy += val;
                return;
        }
        propagate(i);
        int mid1 = (low + high) / 2;
        if(q_low <= mid1)
                change(2*i, low, mid1, q_low, q_high, val);
        if(q_high >= mid1+1)
                change(2*i+1, mid1+1, high, q_low, q_high, val);
        act(i);
}
void change(int q_low, int q_high, ll val) {
        change(1, 0, pot - 1, q_low, q_high, val);
}

int where_smallest(int i = 1) {
        if(i >= pot) return i - pot;
        propagate(i);
        int ret;
        if(tr[2*i].smallest() < tr[2*i+1].smallest())
                ret = where_smallest(2*i);
        else
                ret = where_smallest(2*i+1);
        act(i);
        return ret;
}

ll move_power(int i, ll rem) {
        assert(rem <= 0);
        while(rem < 0) {
                ll cur_value = cur_power[i] - rem;
                cur_power[i] *= 42;
                rem = cur_power[i] - cur_value;
        }
        return rem;
}

set<int> interesting; // the set of indices i that t[i] != t[i+1] (and maybe
few others)
// all other indices have value INF in the tree

ll get_value(int i) {
        assert(interesting.count(i));
        vector<int> w;
        for(int x = (pot + i) / 2; x >= 1; x /= 2)
                w.push_back(x);
        reverse(w.begin(), w.end());
        for(int x : w) propagate(x); // top-down
        return tr[pot+i].smallest();
}

const int SET_TYPE = 10042;
const int INC_TYPE = 10043;
void set_or_inc(int i, ll val, const int type) {
        vector<int> w;
        for(int x = (pot + i) / 2; x >= 1; x /= 2)
                w.push_back(x);
        reverse(w.begin(), w.end());
        for(int x : w) propagate(x); // top-down
        if(type == SET_TYPE) {
                tr[pot+i].lazy = 0;
```

```
                tr[pot+i].local = val;
        }
        else if(type == INC_TYPE) {
                tr[pot+i].local += val;
        }
        else assert(false);
        reverse(w.begin(), w.end());
        for(int x : w) act(x); // bottom-up
}

void re_insert(int i) {
        int j = *interesting.lower_bound(i);
        if(i == j) return;
        cur_power[i] = cur_power[j];
        set_or_inc(i, get_value(j), SET_TYPE);
        interesting.insert(i);
}

void init_value(int i, int val) {
        cur_power[i] = 1;
        ll how_much_remains = 1 - val; // how much remains to cur_power[i]
        ll rem = move_power(i, how_much_remains);
        set_or_inc(i, rem, SET_TYPE);
}

int main() {
        int n, q;
        scanf("%d%d", &n, &q);
        for(int i = 0; i < 2 * pot; ++i)
                tr[i].local = INF;
        for(int i = 1; i <= n; ++i) {
                interesting.insert(i); // we don't mind that maybe t[i] =
t[i+1]
                int val;
                scanf("%d", &val);
                init_value(i, val);
        }
        // queries
        while(q--) {
                int type;
                scanf("%d", &type);
                if(type == 1) { // print value
                        int i;
                        scanf("%d", &i);
                        i = *interesting.lower_bound(i);
                        printf("%lldn", cur_power[i] - get_value(i));
                }
                else if(type == 2) { // set interval to x
                        int low, high, val;
                        scanf("%d%d%d", &low, &high, &val);
                        // t[low] = t[low+1] = ... = t[high-1] = INF
                        // t[high] = val
                        if(low - 1 >= 1)
                                re_insert(low - 1);
                        interesting.insert(high);
                        init_value(high, val);
                        while(true) {
                                auto it = interesting.lower_bound(low);
                                assert(it != interesting.end());
                                int i = *it;
                                assert(i <= high);
                                if(i == high) break; // we only want [low,
high-1]
                                interesting.erase(it);
```

```
                                set_or_inc(i, INF, SET_TYPE);
                        }
                }
                else { // increase by x
                        assert(type == 3);
                        int low, high, val;
                        scanf("%d%d%d", &low, &high, &val);
                        if(low - 1 >= 1)
                                re_insert(low - 1);
                        re_insert(high);
                        bool ok = false;
                        while(!ok) {
                                ok = true;
                                change(low, high, -val);
                                while(true) {
                                        int i = where_smallest();
                                        ll rem = tr[pot+i].smallest();
                                        assert(1 <= i && i <= n);
                                        if(rem > 0) break;
                                        if(rem == 0) {
                                                ok = false;
                                                break;
                                        }
                                        ll new_rem = move_power(i, rem);
                                        set_or_inc(i, new_rem, SET_TYPE);
                                }
                        }
                }
        }
        return 0;
}
```

code2

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

int n, q;

vector <long long> poty;

int n1;

long long inf=(long long)1000000000*1000000000;

long long narz[1000007];
int czybom[1000007];
long long bom[1000007];

long long maxd[1000007];
long long mind[1000007];
long long zos[1000007];

int typ, p1, p2, p3;

inline int potenga(int v)
{
    for (int i=1; 1; i<<=1)
    {
        if (i>=v)
        {
            return i;
        }
    }
```

```
}

inline long long wie(long long v)
{
    return poty[lower_bound(poty.begin(), poty.end(), v)-poty.begin()];
}

inline long long brak(long long v)
{
    return wie(v)-v;
}

inline void pusz(int v)
{
    if (v>=n1)
    {
        czybom[v]=1;
        bom[v]+=narz[v];
        narz[v]=0;
        maxd[v]=bom[v];
        mind[v]=bom[v];
        zos[v]=brak(bom[v]);
        return;
    }
    int cel1, cel2;
    cel1=(v<<1);
    cel2=(cel1^1);
    if (czybom[v])
    {
        czybom[cel1]=1;
        bom[cel1]=bom[v];
        narz[cel1]=0;

        czybom[cel2]=1;
        bom[cel2]=bom[v];
        narz[cel2]=0;

        czybom[v]=0;
        bom[v]=0;
    }
    narz[cel1]+=narz[v];
    narz[cel2]+=narz[v];
    narz[v]=0;

    mind[v]=inf;
    maxd[v]=-inf;
    zos[v]=inf;

    for (int h=0; h<2; h++)
    {

        if (czybom[cel1])
        {
            bom[cel1]+=narz[cel1];
            narz[cel1]=0;
            maxd[v]=max(maxd[v], bom[cel1]);
            mind[v]=min(mind[v], bom[cel1]);
            zos[v]=min(zos[v], brak(bom[cel1]));
        }
        else
        {
            maxd[v]=max(maxd[v], maxd[cel1]+narz[cel1]);
            mind[v]=min(mind[v], mind[cel1]+narz[cel1]);
            zos[v]=min(zos[v], zos[cel1]-narz[cel1]);
```

```
        }

        swap(cel1, cel2);
    }
}

void dod(int v, int a, int b, int graa, int grab, long long w)
{
    if (a>=graa && b<=grab)
    {
        narz[v]+=w;
        return;
    }
    if (a>grab || b<graa)
    {
        return;
    }
    pusz(v);
    dod((v<<1), a, (a+b)>>1, graa, grab, w);
    dod((v<<1)^1, (a+b+2)>>1, b, graa, grab, w);
    pusz(v);
}

void zmi(int v, int a, int b, int graa, int grab, long long w)
{
    if (a>=graa && b<=grab)
    {
        narz[v]=0;
        czybom[v]=1;
        bom[v]=w;
        return;
    }
    if (a>grab || b<graa)
    {
        return;
    }
    pusz(v);
    zmi((v<<1), a, (a+b)>>1, graa, grab, w);
    zmi((v<<1)^1, (a+b+2)>>1, b, graa, grab, w);
    pusz(v);
}

void popr(int v)
{
    pusz(v);
    if (zos[v]>=0)
    {
        return;
    }
    if (mind[v]==maxd[v])
    {
        czybom[v]=1;
        bom[v]=mind[v];
        narz[v]=0;
        pusz(v);
        return;
    }
    popr((v<<1));
    popr((v<<1)^1);
    pusz(v);
}

long long wyn;
```

```
void czyt(int v, int a, int b, int cel)
{
    if (a>cel || b<cel)
    return;
    pusz(v);
    if (a==b)
    {
        wyn=bom[v];
        return;
    }
    czyt((v<<1), a, (a+b)>>1, cel);
    czyt((v<<1)^1, (a+b+2)>>1, b, cel);
}

int main()
{
    scanf("%d%d", &n, &q);
    n1=potenga(n+2);
    poty.push_back(1);
    for (int i=1; i<=11; i++)
    poty.push_back(poty.back()*42);
    zmi(1, 1, n1, 1, n1, 0);
    for (int i=1; i<=n; i++)
    {
        scanf("%d", &p1);
        zmi(1, 1, n1, i, i, p1);
    }
    while(q--)
    {
        scanf("%d", &typ);
        if (typ==1)
        {
            scanf("%d", &p1);
            czyt(1, 1, n1, p1);
            printf("%lldn", wyn);
        }
        if (typ==2)
        {
            scanf("%d%d%d", &p1, &p2, &p3);
            zmi(1, 1, n1, p1, p2, p3);
        }
        if (typ==3)
        {
            scanf("%d%d%d", &p1, &p2, &p3);
            dod(1, 1, n1, p1, p2, p3);
            while(1)
            {
                popr(1);
                if (zos[1]==0)
                {
                    dod(1, 1, n1, p1, p2, p3);
                }
                else
                {
                    break;
                }
            }
        }
    }
    return 0;
}
```

Tutorial of Codeforces Round #356 (Div. 1)

Tutorial of Codeforces Round #356 (Div. 2)

bear, limak, editorial, round 356

+57 Errichto 12 hours ago 51

Comments (51)

Write comment?

12 hours ago, # | ← Rev. 2 +20

maximaxi

All of the editorials have been timed just as the contest ends, but still I have to say thanks a lot for satiating our curiosity for solutions as soon as the contest finished. Thanks for a great round

P.S. Also thanks for many solutions, as sometimes it is hard to understand if there is only one per problem

→ Reply

12 hours ago, # | 0

arknave

That was a very fast editorial! Thank you for a fun round!

→ Reply

12 hours ago, # | ← Rev. 6 +10

maximaxi

Not a big issue, but for 679B "Cubes" editorial, the second bullet point should be,

- *If the first block has side a - 1 [...]*

no?

→ Reply

Errichto

11 hours ago, # ^ | 0

Thank you, corrected.

→ Reply

11 hours ago, # | ← Rev. 2 +6

bardek

An interesting fact: there is a formula to calculate a(n) — the smallest volume of tower consisting of n blocks:

a(n) = a(n-1) + ceil(\sqrt{a(n-1)/3+1/4}-1/2)^3 for n >= 2.

(https://oeis.org/A055402)

→ Reply

Errichto

11 hours ago, # ^ | 0

Yeah, OEIS is powerful. Though, I didn't see how to use this formula to solve a problem.

→ Reply

hellman1908

11 hours ago, # ^ | 0

I don't understand the meaning of the sequence. Could you elaborate?

→ Reply

11 hours ago, # ^ | +5

It answers a bit different question: What is the smallest volume of a tower, that — built with the greedy algorithm from the problem- consists of exactly n blocks?

Yes — as **Errichto** pointed — it's not very useful in the problem, but I wrote about is just as an interesting fact

bardek

because it was a bit surprising for me. Actually it could help only with estimating maximal number of blocks.

→ Reply


hagemaru

11 hours ago, # | ☆ +12

Alternate idea for Bear and Tower of Cubes.
First construct the minimum possible X such that you use maximum number of blocks. This can be done greedily. Store all the sides of cubes. Now we need to convert this ans to maximum. For that, loop over the sides of cubes in decreasing order. Keep increasing the side of cube till it is valid.
Code
Unfortunately during the contest I had a bug in code which I found after the contest. :(

→ Reply


retrograd

11 hours ago, # ^ | ☆ ← Rev. 3 0

Here it seems like you are trying to do a greedy version of a knapsack-type problem, and it doesn't feel like it should give the right answer every time. Please update us with the status of the resubmission (whether it gets AC or not). In my opinion it should not.

Later edit: It seems like it got accepted (I should see the links before commenting, shouldn't I?). But why is that?

→ Reply


hagemaru

11 hours ago, # ^ | ☆ 0

AC on resubmit. Code

→ Reply


hellman1908

11 hours ago, # | ☆ 0

Got TL in C because of map<int,int> instead of simple array :/ with array it's ~600ms.

→ Reply


luka25

11 hours ago, # | ☆ 0

Could Anybody explain me, how to get "12 941" in Div1 B when input is 994? I see that 729+216+27+8+8+1+1+1+1+1+1=941 and it's just 11...

→ Reply


Errichto

11 hours ago, # ^ | ☆ 0

$729 + 216 + 27 + 8 + 8 + 1 + 1 + 1 + 1 + 1 + 1 = 994$, not 941 as you wrote.

→ Reply


luka25

11 hours ago, # ^ | ☆ 0

Ah, you're right, silly me.

→ Reply


shark_s

11 hours ago, # ^ | ☆ 0

@Errichato our first task in Div1B is find max no of blocks and then find maximum X. (1<=X<=M) so for m = 994, ans will be 14 798 how it is 12 941 ?

→ Reply

11 hours ago, # ^ | ☆ 0

I guess for $X = 798$ Limak won't use 14 blocks.

→ Reply

Errichto

→ Reply

shark_s

10 hours ago, # ^ | ☆ 0

okk get it my mistake.

→ Reply

Sawako

11 hours ago, # | ☆ ← Rev. 2 0

Edit: removed, issue resolved

→ Reply

ivankrut856

11 hours ago, # | ☆ -8

Will russian version of editorial available?

→ Reply

Errichto

11 hours ago, # ^ | ☆ 0

There will be only English version, sorry for the inconvenience.

→ Reply

derAdler

11 hours ago, # | ☆ 0

in problem d of div2 are we fixing a square of side k and for all k*k members we are subtracting 1 if it's part of a component which lies outside this k*k zone and then doing it for all possible rows from 0 to n-k (and k columns in that zone). Could someone plz explain the editorial (to a fool like me) in very SIMPLE terms. Thanks in advance

→ Reply

Errichto

11 hours ago, # ^ | ☆ 0

The slower approach is the following. For each possible placement subtract 1 from the size of a connected component which contains this cell. Then, iterate over connected components and for each of them if it touches the $k \times k$ square then add its size to some current sum (possible answer). Also, you must add k^2 (size of the $k \times k$ square) to the "current sum".

To make this solution faster, you should use the technique called sliding-window (described in the editorial). Also, you can google more tutorials about this technique.

→ Reply

fahim.cross

11 hours ago, # | ☆ 0

can someone please explain the problem Bear and Tower of Cubes?

→ Reply

YangZhao512

10 hours ago, # ^ | ☆ +1

Let me try to explain to you. All you need to realise is that, the answer to a given number 'm', will not be worse than a given number of 'm-1'. Why? Notice that, we **don't** have to construct a tower having **exactly** 'm' volume. The problem only requires that we build a tower with a volume less or equal to 'm'. Thus, if the answer to build a tower with **exact** volume of 'm-1' is better than building a tower with **exact** volume of 'm', we can just build the smaller tower instead.

After realising this point, you can compare those cases mentioned in the tutorial. In the first case, we have initial volume of 'm', then after selecting size 'a' block, the left volume is m — a^3. In case 2, the left volume is a^3 — (a-1)^3. We just simply choose the case which has larger remaining volume. I think code2 in the editorial has a better reflection of this idea.

→ Reply

ffao

10 hours ago, # | ☆ +1

When I read D I knew that it would have a beautiful solution: glad to see the magic did not disappoint me :D

→ Reply

tenshi_kanade

10 hours ago, # | ☆ ← Rev. 2 +13

I had a different approach for problem B.

- First I greedily found the maximum number of blocks by adding each time the maximum possible element. An element can be added if $S + x^3 < (x+1)^3$, where S is the sum of all elements so far.
- Once I know the maximum number of blocks, I start from the back, and then again, greedily try to increase elements while the arrangement remains correct (the same principle applies to determine correctness). This last step is $O(|v|^2)$, where v is the vector with the sizes of the added blocks.

→ Reply

CodeForceps

10 hours ago, # ^ | ☆ ← Rev. 2 0

If the order is `O(v²)` shouldn't it be TLE? I mean, |v| is 10^5, right?

→ Reply

tenshi_kanade

9 hours ago, # ^ | ☆ ← Rev. 2 0

No, as the editorial says, $|v|$ is very small. I didn't do the exact calculatios during contest, but I assumed it would be pretty small. The editorial claims that $|v|$ is at most 18 in the worst case, so $|v|^2$ is blazingly fast.

→ Reply

CodeForceps

8 hours ago, # ^ | ☆ +5

Oh, I missed the part of "the *added* blocks". Sorry!

→ Reply

ErdemKirez

9 hours ago, # ^ | ☆ 0

Is there any proof for second part? Our problem is knapsack and you do it greedily?

→ Reply

RomanM

10 hours ago, # | ☆ 0

Thanks for the interesting contest!

→ Reply

derAdler

10 hours ago, # | ☆ 0

can someone explain bear and square grid using sliding window technique.PLzz!! I specifically did not understand this paragraph "We will used a sliding-window technique. Let's fix row of the grid — the upper row of the square. Then, we will first place the square on the left, and then we will slowly move a square to the right. As we move a square, we should iterate over cells that stop or start to belong to the square. For each such empty cell we should add or subtract 1 from the size of its CC (ids and sizes of CC's were found at the beginning)." in the editorial and even more specifically this part "As we move a square, we should iterate over cells that stop or start to belong to the square.". Thanks in advance(I m new to programming so plz go easy on me).

→ Reply

9 hours ago, # ^ | ☆ ← Rev. 3 0

Let's fix the upper left corner at (i_1, j_1). Then $i_2 = i_1 + K - 1$ and $j_2 = j_1 + K - 1$. The included cells in this case are in the range $(i_1 - 1, j_1 - 1)$ to $(i_2 + 1, j_2 + 1)$, except cells $(i_1 - 1, j_1 - 1)$

$(i_1 - 1, j_1 - 1)$ to $(i_2 + 1, j_2 + 1)$, except cells $(i_1 - 1, j_1 - 1)$, $(i_1 - 1, j_2 + 1)$, $(i_2 + 1, j_1 - 1)$ and $(i_2 + 1, j_2 + 1)$.

To do this efficiently, let's fix $i1$ and initialize $j1 = 1$. We can then calculate it by visiting all mentioned cells, then when we move j_1 to the right, we only need to remove cells in the range $(i_1, j_1 - 2)$ to $(i_2, j_1 - 2)$ and add cells in the range $(i_1, j_2 + 1)$ to $(i_2, j_2 + 1)$. Additionally, we need to remove cells $(i_1 - 1, j_1 - 1)$ and $(i_2 + 1, j_1 - 1)$ and add cells $(i_1 - 1, j_2)$ and $(i_2 + 1, j_2)$. Of course, we don't consider cells that are out of the board. Once we've reached the state with $j_1 = N - K + 1$, we need to reset all the counters, and we can do it the same way we initialized them for $j_1 = 1$.

So we perform K^2 operations at the start of every row, and then as we move $j1$ to the right, we perform around $2K$ operations each time, so the algorithm's complexity is in the order of $O(N * K^2 + N^2 * K) = O(N^2 * K)$.


tenshi_kanade

→ Reply


derAdler

9 hours ago, # ^ | ☆ ← Rev. 2 +5

thanku so much i understand now

→ Reply


shas19

10 hours ago, # | ☆ 0

In div2 E ,How to count for current window how many of the nodes of the component are outside and how many are inside?

→ Reply


derAdler

10 hours ago, # ^ | ☆ 0

I have the same doubt but could you plz explain me how to select the window

→ Reply


shas19

10 hours ago, # ^ | ☆ 0

According to editorial,we have to select all possible windows having size k*k there can be at max n*n such windows which will completely fit inside the grid

→ Reply


derAdler

9 hours ago, # ^ | ☆ 0

are we supposed to iterate for all k*k units of the window??

→ Reply


Errichto

10 hours ago, # ^ | ☆ 0

It's described in the editorial, and you can check my code for details. We need an array to remember for each CC how many cells are outside the square at this moment (for this square).

→ Reply


shas19

9 hours ago, # ^ | ☆ 0

Got it!! Thanks.. Awesome problem btw..

→ Reply


Qubit01

7 hours ago, # | ☆ +2

I'm not really convinced why in D Div2 you should always select side a and a-1 but not a-2. The reason is that having a smaller quantity doesn't necessarily yield less towers.

Can you tell me where's my mistake?

→ Reply

6 hours ago, # | ☆ +1

haleyk100198

Thanks for the quick editorial, I'd imagine many div2 participants would stop working after finishing Problem C, the gap between C and D is pretty huge... Perhaps the interactive question tutorial helped out most of us more than expected.

→ Reply

5 hours ago, # | ☆ ← Rev. 2 0

MayankPratap

For 3rd problem my first submission got WA on pretest 5 but I can't understand why ? Can anybody help me?? Here is the submission.
http://codeforces.com/contest/680/submission/18318572

→ Reply

3 hours ago, # ^ | ☆ 0

w1n5t0n

In the last part, when you check if the number is divisible by (whoDivided[0])^2, or the square of prime, you did not cin>>string to get the answer. The string that you initiated has random behavior. so in most cases, that string would not be yes.

→ Reply

3 hours ago, # ^ | ☆ ← Rev. 2 0

Dushyant

After correcting that, it gives WA on testcase 6.

Submission

→ Reply

84 minutes ago, # ^ | ☆ 0

w1n5t0n

After checking if the number is divisible by prime^2, you should print composite if yes, and prime if no. In your code, after checking if prime^2 is in range, you assume that the answer would be composite if prime^2>100, which is wrong. If the number cant be divided by prime^2, then it would be a prime number.

→ Reply

4 hours ago, # | ☆ ← Rev. 3 0

In problem Div1D (Bear and Chase) on the code you say:

marX

```
// 1) consider guessing immediately
T immediately = (T) 1 / n; // how much it counts towards the answer
```

Why you said the probability of predicting immediately is 1 / n, instead of 1 / cnt_cities, if you already now that bear will be in a city with distance d1 from g1?

→ Reply

3 hours ago, # | ☆ 0

docodon

in tower of cubes why a and a-1 are the possible options why not a-2 or less ? Although it seems to be working for smaller values but I am not able to figure it out why it will work for each case for sure ! Can anyone help ?

→ Reply

54 minutes ago, # ^ | ☆ 0

YangZhao512

Look at my comment for explanation.

→ Reply

2 hours ago, # | ☆ 0

SuryanshT

My solution is shorter than **Errichto**

→ Reply