Содержание

1 Общее

1.1 Пробный тур

1. Первым делом почекать, считаются ли WA и CE за попытки. Можно ли засылать в уже сданную задачу?
2. Пописать код и посдавать каждому члену команды.
3. Сравнить скорости компа и тестирующей системы на Флойде ($n \sim 1050$).
4. Запустить все IDE, проверить, что все настройки работают, проверить, что работает C++11.
5. Проверить, есть ли `int128`.
6. Проверить работу прагм в тестирующей системе.
7. Почекать максимальный размер отправляемого кода.
8. Проверить рабочесть стресса.

1.2 Чеклист при отправке

1. Протестить на всех тестах из примера и других рандомных тестах.
2. Протестить все крайние случаи.
3. Убрать дебаг вывод.
4. Точно убедиться в правильности ответа и формата вывода.
5. Перечитать формат ввода/вывода.
6. Проверить, все ли хорошо с мультитестом.

1.3 Факты из жизни IMO KINGS

- 1 января 2000 года – суббота, 1 января 1900 года – понедельник.
- Високосные года: если $400 \mid a$, либо если $4 \mid a$, но не $100 \mid a$.
- `INT_MIN` $= -2\,147\,483\,648$, `INT_MAX` $= 2\,147\,483\,647$, `UINT_MAX` $= 4\,294\,967\,295$,
- `SHRT_MIN` $= -32\,768$, `SHRT_MAX` $= 32\,767$, `LLONG_MIN` $= -9\,223\,372\,036\,854\,775\,808$,
- `LLONG_MAX` $= 9\,223\,372\,036\,854\,775\,807$, `ULLONG_MAX` $= 18\,446\,744\,073\,709\,551\,615$.

10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}	10^{11}	10^{12}	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
	−47	−63	−113	−123	−137	−111	−213	−267	−231	−231	−233	−447	−203	−429	−369	−413	−369
	−41	−59	−99	−119	−117	−99	−179	−261	−219	−179	−153	−411	−179	−423	−359	−273	−363
	−39	−53	−93	−99	−93	−93	−173	−249	−213	−171	−143	−357	−171	−357	−357	−261	−291
	−33	−47	−77	−93	−83	−71	−161	−243	−183	−167	−137	−341	−147	−341	−329	−239	−263
	−29	−33	−71	−77	−69	−69	−153	−239	−167	−149	−123	−299	−77	−191	−191	−177	−251
	−27	−29	−69	−71	−47	−63	−69	−203	−149	−129	−101	−267	−71	−173	−183	−81	−171
−8	−21	−23	−59	−39	−41	−57	−59	−117	−119	−93	−63	−237	−69	−123	−149	−57	−137
−7	−17	−17	−51	−29	−39	−29	−41	−107	−71	−57	−41	−201	−41	−117	−113	−39	−123
−5	−11	−9	−33	−11	−21	−27	−29	−71	−57	−53	−39	−137	−29	−53	−83	−23	−33
−3	−3	−3	−27	−9	−17	−9	−11	−63	−33	−23	−11	−29	−27	−11	−63	−3	−11

+1	+1	+9	+7	+3	+3	+19	+7	+7	+19	+3	+39	+37	+31	+37	+61	+3	+3
+3	+3	+13	+9	+19	+33	+79	+37	+9	+33	+19	+61	+51	+67	+91	+69	+13	+9
+7	+7	+19	+37	+43	+37	+103	+39	+21	+61	+57	+63	+99	+97	+159	+79	+19	+31
+9	+9	+21	+39	+49	+39	+121	+49	+33	+69	+63	+91	+129	+99	+187	+99	+21	+79
+13	+13	+31	+61	+57	+81	+139	+73	+87	+97	+69	+121	+183	+133	+223	+453	+49	+177
+19	+27	+33	+67	+69	+99	+141	+81	+93	+103	+73	+163	+259	+139	+241	+481	+81	+183
+21	+31	+39	+69	+103	+117	+169	+123	+97	+121	+91	+169	+267	+169	+249	+597	+99	+201
+27	+37	+49	+79	+109	+121	+189	+127	+103	+141	+103	+177	+273	+183	+259	+613	+141	+283
+31	+39	+51	+91	+129	+133	+223	+193	+123	+147	+129	+189	+279	+261	+273	+639	+181	+381
+33	+49	+61	+93	+151	+151	+229	+213	+181	+207	+171	+193	+283	+357	+279	+669	+337	+387

1.4 Формулы

- Расстояние между точками по сфере: $L = R \cdot \arccos(\cos\theta_1 \cdot \cos\theta_2 + \sin\theta_1 \cdot \sin\theta_2 \cdot \cos(\varphi_1 - \varphi_2))$, где θ – широты (от $-\pi$ до π), φ – долготы (от $-\pi$ до π)
- Объем шарового сегмента: $V = \pi h^2(R - \frac{1}{3}h)$, где h – высота от вершины сектора до секущей плоскости
- Площадь поверхности шарового сегмента: $S = 2\pi Rh$, где h – высота
- $2^{23} \cdot 7 \cdot 17 + 1 = 998\,244\,353$ – простое, первообразный корень – 3
- Код Грея: $g_n = n \text{ XOR } \frac{n}{2}$
- Числа Фибоначчи: $F_0 = 0, F_1 = 1, F_n = \dfrac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$

- Формула Кардано:
  - Переведём $ax^3+bx^2+cx+d=0$ при помощи $x=y-\frac{b}{3a}$ к виду $y^3+py+q=0$, где $p=\frac{c}{a}-\frac{b^2}{3a^2}=\frac{3ac-b^2}{3a^2}$, $q=\frac{2b^3}{27a^3}-\frac{bc}{3a^2}+\frac{d}{a}=\frac{2b^3-9abc+27a^2d}{27a^3}$
  - $Q=\left(\frac{p}{3}\right)^3+\left(\frac{q}{2}\right)^2$, $\alpha=\sqrt[3]{-\frac{q}{2}+\sqrt{Q}}$, $\beta=\sqrt[3]{-\frac{q}{2}-\sqrt{Q}}$, $\Delta=-108Q$, $\alpha\beta=-\frac{p}{3}$
  - $y_1=\alpha+\beta$, $y_{2,3}=-\frac{\alpha+\beta}{2}\pm i\frac{\alpha-\beta}{2}\sqrt{3}$
- Числа Стирлинга: $s(n,k)$ — количество перестановок на n элементах, в которых ровно k циклов. $S(n,k)$ – это количество способов разбить n-элементное множество на k непустых подмножеств.
  - † $s(n,k)=(n-1)\cdot s(n-1,k)+s(n-1,k-1)$
  - † $S(n,k)=k\cdot S(n-1,k)+S(n-1,k-1)$
  - † $x^{\underline{n}}=x\cdot(x-1)\cdot\ldots\cdot(x-n+1)=\sum\limits_{i=1}^{n}(-1)^{n-i}\cdot s(n,\ i)\cdot x^i$
  - † $x^n=\sum\limits_{i=0}^{n}S(n,\ i)\cdot x^{\underline{i}}$
- Число разбиений на неубывающие натуральные слагаемые:

i	p_i	i	p_i	i	p_i	i	p_i	i	p_i	i	p_i
0	1	10	42	20	627	30	5604	40	37 338	50	204 226
1	1	11	56	21	792	31	6842	41	44 583	51	**239**943
2	2	12	77	22	1002	32	8349	42	53 174	52	281 589
3	3	13	101	23	1255	33	10 143	43	63 261	53	329 931
4	5	14	135	24	1575	34	12 310	44	75 175	54	386 155
5	7	15	176	25	1958	35	14 883	45	89 134	55	451 276
6	11	16	231	26	2436	36	17 977	46	105 558	56	526 823
7	15	17	297	27	3010	37	21 637	47	124 754	57	614 154
8	22	18	385	28	3718	38	26 015	48	147 273	58	715 220
9	30	19	490	29	4565	39	31 185	49	173 525	59	831 820

$$p_{100}=190\,569\,292$$

1.5 Максимальное количество делителей

$\leqslant N$	n	Факторизация	$d(n)$
20	12	$2^2\cdot3^1$	6
50	48	$2^4\cdot3^1$	10
100	60	$2^2\cdot3^1\cdot5^1$	12
10^3	840	$2^3\cdot3^1\cdot5^1\cdot7^1$	32
10^4	7560	$2^3\cdot3^3\cdot5^1\cdot7^1$	64
10^5	83 160	$2^3\cdot3^3\cdot5^1\cdot7^1\cdot11^1$	128
10^6	720 720	$2^4\cdot3^2\cdot5^1\cdot7^1\cdot11^1\cdot13^1$	240
10^7	8 648 640	$2^6\cdot3^3\cdot5^1\cdot7^1\cdot11^1\cdot13^1$	448
10^8	91 891 800	$2^3\cdot3^3\cdot5^2\cdot7^1\cdot11^1\cdot13^1\cdot17^1$	768
10^9	931 170 240	$2^6\cdot3^2\cdot5^2\cdot7^1\cdot11^1\cdot13^1\cdot17^1\cdot19^1$	1344
10^{11}	97 772 875 200	$2^6\cdot3^3\cdot5^2\cdot7^2\cdot11^1\cdot13^1\cdot17^1\cdot19^1$	4032
10^{12}	963 761 198 400	$2^6\cdot3^4\cdot5^2\cdot7^1\cdot11^1\cdot13^1\cdot17^1\cdot19^1\cdot23^1$	6720
10^{15}	866 421 317 361 600	$2^6\cdot3^4\cdot5^2\cdot7^1\cdot11^1\cdot13^1\cdot17^1\cdot19^1\cdot23^1\cdot29^1\cdot31^1$	26 880
10^{18}	897 612 484 786 617 600	$2^8\cdot3^4\cdot5^2\cdot7^2\cdot11^1\cdot13^1\cdot17^1\cdot19^1\cdot23^1\cdot29^1\cdot31^1\cdot37^1$	103 680

- Gambler's ruin. У первого игрока есть n_1 монет, у второго n_2. На каждом шаге с вероятностью p второй отдает одну монету первому, а с вероятностью $q=1-p$ первый отдает одну монету второму. Игра заканчивается, когда у кого-нибудь не остается монет. Тогда первый выигрывает с вероятностью $\dfrac{1-\left(\frac{p}{q}\right)^{n_1}}{1-\left(\frac{p}{q}\right)^{n_1+n_2}}$.
- Пентагональная теорема Эйлера: $\text{partitions}(n)=[x^n]\frac{1}{\varphi(x)}$; $\varphi(x)=\prod_{k=1}^{+\infty}(1-x^k)=1+\sum_{k=1}^{+\infty}(-1)^k(x^{\frac{k(3k-1)}{2}}+x^{\frac{k(3k+1)}{2}})$

- Шар и сфера: $V_n = C_n R^n$, $S_n = \frac{dV_n}{dR} = nC_nR^{n-1}$, $C_n = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} = \begin{cases} \frac{\pi^k}{k!}, & n = 2k; \\ \frac{2^{k+1}\pi^k}{(2k+1)!!}, & n = 2k+1. \end{cases}$
- Штрассен $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, AB = \begin{pmatrix} D + D_1 + V_1 - H_1 & V_2 + H_1 \\ V_1 + H_2 & D + D_2 + V_2 - H_2 \end{pmatrix}$, $D = (A_{11} + A_{22})(B_{11} + B_{22})$, $D_1 = (A_{12} - A_{22})(B_{21} + B_{22})$, $D_2 = (A_{21} - A_{11})(B_{11} + B_{12})$, $H_1 = (A_{11} + A_{12})B_{22}$, $H_2 = (A_{21} + A_{22})B_{11}$, $V_1 = A_{22}(B_{21} - B_{11})$ $V_2 = A_{11}(B_{12} - B_{22})$
- Метод Ньютона:
  - † $\left(\frac{1}{Q}\right)_{n+1} = \left(\frac{1}{Q}\right)_n \left(2 - Q\left(\frac{1}{Q}\right)_n\right)$
  - † $\log(Q) = \int \frac{Q'}{Q}$
  - † $\exp(Q)_{n+1} = \exp(Q)_n(1 + Q - \log\exp(Q)_n)$
  - † $\sqrt[k]{Q}_{n+1} = \sqrt[k]{Q}_n \left(\frac{k-1}{k} + \frac{Q}{k}\sqrt[k]{Q}_n^k\right) = \exp\frac{\log(Q)}{k}$

1.6 Чеклист при WA

1. Распечатать.
2. Отдать комп следующему.
3. Проверить код по строчке, даже условия в форах и т.д.
4. Проверить, не забыл ли случаи.
5. Забрать комп, потестить, предварительно составив много тестов и ответы к ним.
6. Написать стресс.
7. Возможно, решение вообще неправильное.
8. Возможно, WA где-то не там, где ты думаешь.

2 Коды

2.1 Полезное

```
Прагмы:
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,avx,avx2")

Встроенный декартач:
#include <ext/pb_ds/assoc_container.hpp> // Общий файл.
#include <ext/pb_ds/tree_policy.hpp> // Содержит класс tree_order_statistics_node_update
//#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;
typedef
tree<
  int,
  null_type,
  less<int>,
  rb_tree_tag,
  tree_order_statistics_node_update>
            }
ordered_set;

ordered set q;
q.find_by_order(1);
q.order_of_key(2);

Atomic hashset:
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table;
-----------------------------------------------
```

```
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<key, int, chash> table;
```

Atomic hashmap:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef cc_hash_table<int, int, hash<int>> ht;
```

Фишки битсета:

```
bitset<N> a;
for (int i = a._Find_first(); i != a.size(); i = a._Find_next(i)) { cout << i };
```

Перебор всех подмасок:

```
for (int s=m; ; s=(s-1)&m) {
  ... можно использовать s ...
  if (s==0)  break;
}
```

2.2 Stress

В far. shift+f4 — создать файл. g++ a.cpp. Файл a.bat. Запустить — просто a.bat. Рандом хороший нужен, часто меняющийся.

```
:beg
gen.exe > in.txt
a.exe < in.txt > out1.txt
b.exe < in.txt > out2.txt
fc out1.txt out2.txt
if errorlevel 1 goto bug
goto beg
:bug
echo found!
```

RE stress:

```
:beg
gen.exe > in.txt
sol.exe < in.txt > out.txt || exit
goto beg
```

2.3 Дробь с минимальным знаменателем между двумя другими

```
rat find_best(rat l, rat r) {
    if (l.x >= l.y) {
        ll d = l.x / l.y;
        rat res = find_best(rat(l.x - d * l.y, l.y), rat(r.x - d * r.y, r.y));
        res.x += res.y * d;
        return res;
    }
    if (r.x > r.y)
        return rat(1, 1);
    rat res = find_best(rat(r.y, r.x), rat(l.y, l.x));
    return rat(res.y, res.x);
}
```

2.4 Битсет

```
const int SZ = 6;
const int BASE = (1 << SZ);
const int MOD = BASE - 1;

struct Bitset {
    typedef unsigned long long T;
    vector<T> data;
    int n;
    void resize(int nn) {
        n = nn;
        data.resize((n + BASE - 1) / BASE);
    }
    void set(int pos, int val) {
        int id = pos >> SZ;
        int rem = pos & MOD;
        data[id] ^= data[id] & (1LL << rem);
        data[id] |= val * (1LL << rem);
    }
    int get(int pos) {
        return (data[pos >> SZ] >> (pos & MOD)) & 1;
    }
    // k > 0 -> (*this) << k
    // k < 0 -> (*this) >> (-k)
    Bitset shift (int k) {
        Bitset res;
        res.resize(n);
        int s = k / BASE;
        int rem = k % BASE;
        if (rem < 0) {
            rem += BASE;
            s--;
        }
        int p1 = BASE - rem;
        T mask = (p1 == 64)? -1: (1LL << p1) - 1;
        for (int i = max(0, -s); i < sz(data) - max(s, 0); i++) {
            res.data[i + s] |= (data[i] & mask) << rem;
        }
        if (rem != 0) {
            for (int i = max(0, -s - 1); i < sz(data) - max(s + 1, 0); i++) {
                res.data[i + s + 1] |= (data[i] >> p1) & ((1LL << rem) - 1);
            }
        }
        int cc = data.size() * BASE - n;
        res.data.back() <<= cc;
        res.data.back() >>= cc;
        return res;
    }
};
```

2.5 FFT

```
namespace FFT {
using db = double;
class FFT {//NTT
//double vs long double!!!
public:
const db PI = 4 * atan2(1, 1);
/*
const int n = 20;
```

```
const int size = (1 << n);
const int MOD = 998244353; //g = 3
const int ROOT = 565042129;
*/
FFT(int _n) : size(1 << _n), n(_n) { //NTT()
    revers.resize(size), root.resize(size);
    fftA.resize(size), fftB.resize(size);
    for (int i = 0; i < size / 2; i++) {
        root[i] = Complex(cos(2 * PI * i / size), sin(2 * PI * i / size));
        root[i + size / 2] = Complex(-root[i].real, -root[i].image);
    }
    /*
    root[0] = 1;
    for (int i = 1; i < size; i++)
        root[i] = (ll) root[i - 1] * ROOT % MOD;
    */
}
void fft(vector <Complex> &poly, int newN) { //Complex -> int
    revers[0] = 0;
    for (int i = 1; i < (1 << newN); i++) {
        if (i % 2 == 0) revers[i] = revers[i / 2] / 2;
        else revers[i] = revers[i / 2] / 2 + (1 << (newN - 1));
        if (revers[i] < i) swap(poly[revers[i]], poly[i]);
    }
    for (int level = 1; level <= newN; level++)
        for (int block = 0; block < (1 << (newN - level)); block++)
            for (int step = 0; step < (1 << (level - 1)); step++) {
                int num1 = (block << level) + step;
                int num2 = num1 + (1 << (level - 1));
                Complex valA = poly[num1];                              //Complex -> int
                Complex valB = root[step << (n - level)] * poly[num2]; //Complex -> int,(ll),%MOD
                poly[num1] = valA + valB; // % MOD
                poly[num2] = valA - valB; // % MOD
            }
}
void rev_fft(vector <Complex> &poly, int step) { //Complex -> int
    fft(poly, step);
    reverse(poly.begin() + 1, poly.begin() + (1 << step));
    for (int i = 0; i < (1 << step); i++) poly[i] = poly[i] / (1 << step);
    /*
    int inv_size = binpow((1 << step), MOD - 2);
    for (int i = 0; i < (1 << step); i++) poly[i] = (ll) poly[i] * inv_size % MOD;
    */
}
vector <db> multiply(const vector <db> &A, const vector <db> &B, int step) { //db -> int
    fill(fftA.begin(), fftA.begin() + (1 << step), 0);
    fill(fftB.begin(), fftB.begin() + (1 << step), 0);
    for (int i = 0; i < (int) A.size(); i++) fftA[i] = A[i]; // % MOD
    for (int i = 0; i < (int) B.size(); i++) fftB[i] = B[i]; // % MOD
    fft(fftA, step);
    fft(fftB, step);
    for (int i = 0; i < (1 << step); i++) fftA[i] = fftA[i] * fftB[i]; // (ll), % MOD
    rev_fft(fftA, step);
    vector <db> result(1 << step); //db -> int
    for (int i = 0; i < (1 << step); i++) result[i] = fftA[i].real; // = fftA[i]
    return result;
}
private:
int size, n;
vector <Complex> root;
vector <int> revers;
```

```
vector <Complex> fftA, fftB;
//vector <int> root, revers, fftA, fftB;
};
}
```

2.6 OR-XOR-AND-свёртки

```
namespace Convolution {
class Convolution {
    using Int = int;
public:
    Convolution(int _n) : n(_n), size(1 << _n) {
        helpA.resize(size), helpB.resize(size);
    }
    // OR  -- (1  0), REV -- ( 1 0)
    //        (1  1)         (-1 1)
    // AND -- (1  1), REV -- (1 -1)
    //        (0  1)         (0  1)
    // XOR -- (1  1), REV -- (1  1), /= size
    //        (1 -1)         (1 -1)
    vector <Int> convolution(const vector <Int> &A, const vector <Int> &B) {
        for (int i = 0; i < size; i++) helpA[i] = A[i];
        for (int i = 0; i < size; i++) helpB[i] = B[i];
        transform(helpA);
        transform(helpB);
        for (int i = 0; i < size; i++) helpA[i] = helpA[i] * helpB[i];
        rev_transform(helpA);
        return helpA;
    }
private:
    vector <Int> helpA, helpB;
    int n, size;
};
}
```

2.7 Суффиксное дерево

```
namespace SuffixTree {
struct Position {
int vertex, shift;
};
const int ALPHABET = 26;
class SuffixTree {
public:
SuffixTree(const vector <int> &s) {
    str = s;
    int n = (int) str.size();
    str_bounds.resize(2 * n, mp(0, 0)), suflink.resize(2 * n, -1), parent.resize(2 * n, mp(-1, -1));
    for (int i = 0; i < ALPHABET; i++) next_step[i].resize(2 * n, -1);
    quant++;
    int root = 0;
    str_bounds[root] = mp(0, -1);
    suflink[root] = root;
    Position cur = {root, 0};
    for (int i = 0; i < n; i++) {
        int symb = str[i];
        bool do_move = true;
        Position old = {-1, -1};
        while (!can_move(cur, symb)) {
            if (cur.vertex == root) do_move = false;
            cur = create_edge(cur, symb, i, n - 1);
```

```
            if (old.vertex != -1) suflink[old.vertex] = cur.vertex;
            old = cur;
            cur = jump_suflink(cur);
        }
        if (old.vertex != -1) suflink[old.vertex] = cur.vertex;
        if (do_move) cur = move_next(cur, symb);
    }
}
bool can_move(Position cur, int symb) const {
    if (cur.shift == 0) return next_step[symb][cur.vertex] != -1;
    else {
        int next_symb = str[str_bounds[cur.vertex].x + cur.shift];
        return next_symb == symb;
    }
}
Position create_edge(Position cur, int symb, int lef, int rig) {
    if (cur.shift != 0) {
        quant++;
        int mid_vertex = quant - 1;
        parent[mid_vertex] = parent[cur.vertex];
        next_step[parent[mid_vertex].x][parent[mid_vertex].y] = mid_vertex;
        int next_symb = str[str_bounds[cur.vertex].x + cur.shift];
        parent[cur.vertex] = mp(next_symb, mid_vertex);
        next_step[next_symb][mid_vertex] = cur.vertex;
        str_bounds[mid_vertex] = mp(str_bounds[cur.vertex].x, str_bounds[cur.vertex].x + cur.shift - 1);
        str_bounds[cur.vertex] = mp(str_bounds[cur.vertex].x + cur.shift, str_bounds[cur.vertex].y);
        cur = {mid_vertex, 0};
    }
    quant++;
    int new_vertex = quant - 1;
    next_step[symb][cur.vertex] = new_vertex;
    parent[new_vertex] = mp(symb, cur.vertex);
    str_bounds[new_vertex] = mp(lef, rig);
    return cur;
}
Position jump_suflink(Position cur) const {
    int lef = 0, rig = 0;
    if (cur.shift == 0) {
        if (suflink[cur.vertex] != -1) return {suflink[cur.vertex], 0};
        lef = str_bounds[cur.vertex].x, rig = str_bounds[cur.vertex].y;
    } else {
        lef = str_bounds[cur.vertex].x, rig = str_bounds[cur.vertex].x + cur.shift - 1;
    }
    if (parent[cur.vertex].y == 0) lef++;
    Position new_cur = {suflink[parent[cur.vertex].y], 0};
    int tek = lef;
    while (tek <= rig) {
        int next_vert = next_step[str[tek]][new_cur.vertex];
        if (str_bounds[next_vert].y - str_bounds[next_vert].x + 1 <= rig - tek + 1) {
            tek += str_bounds[next_vert].y - str_bounds[next_vert].x + 1;
            new_cur = {next_vert, 0};
        } else {
            new_cur = {next_vert, rig - tek + 1};
            tek = rig + 1;
        }
    }
    return new_cur;
}
Position move_next(Position cur, int symb) const {
    if (cur.shift == 0) {
        int next_vert = next_step[symb][cur.vertex];
```

```
        if (str_bounds[next_vert].y == str_bounds[next_vert].x) return {next_vert, 0};
        else return {next_vert, 1};
    } else {
        if (str_bounds[cur.vertex].y - str_bounds[cur.vertex].x == cur.shift) return {cur.vertex, 0};
        else return {cur.vertex, cur.shift + 1};
    }
}
private:
int quant = 0;
vector <int> str;
vector <pair <int, int> > str_bounds, parent;
vector <int> next_step[ALPHABET + 3], suflink;
};
}
```

2.8 Суффиксный автомат

```
namespace SuffixAutomaton {
const int ALPHABET = 26;
class SuffixAutomaton {
public:
SuffixAutomaton(const vector <int> &s) {
    int n = (int) s.size();
    int max_states = max(2 * n - 1, 5);
    len.resize(max_states, 0), suflink.resize(max_states, -1);
    for (int i = 0; i < ALPHABET; i++) next_step[i].resize(max_states, -1);
    int root = 0;
    quant++;
    int cur = root;
    for (int x : s) cur = add_symbol(cur, x);
    last_vert = cur;
}
int add_symbol(int last, int c) {
    quant++;
    int cur = quant - 1;
    len[cur] = len[last] + 1;
    int p = last;
    while (p != -1 && next_step[c][p] == -1) {
        next_step[c][p] = cur;
        p = suflink[p];
    }
    if (p == -1) {
        suflink[cur] = 0;
        return cur;
    }
    int q = next_step[c][p];
    if (len[q] == len[p] + 1) {
        suflink[cur] = q;
        return cur;
    }
    quant++;
    int clone = quant - 1;
    len[clone] = len[p] + 1;
    while (p != -1 && next_step[c][p] == q) {
        next_step[c][p] = clone;
        p = suflink[p];
    }
    for (int i = 0; i < ALPHABET; i++) next_step[i][clone] = next_step[i][q];
    suflink[clone] = suflink[q];
    suflink[q] = clone;
    suflink[cur] = clone;
```

```
    return cur;
}
private:
int quant = 0, last_vert = 0;
vector <int> len, suflink;
vector <int> next_step[ALPHABET + 3];
};
}
```

2.9 Два китайца

```
namespace TwoChinese {
struct Edge {
int from, to, weight;
};
class TwoChinese {
public:
TwoChinese(int _n, const vector <Edge> &_edges, int _root) : n(_n), edges(_edges), root(_root) {
    komp.resize(n), metka.resize(n), in_edges.resize(n);
    delta.resize(n), parent.resize(n), used.resize(n), finish.resize(n);
    for (int i = 0; i < n; i++) metka[i] = i, komp[i].pb(i);
    for (int i = 0; i < (int) edges.size(); i++)
        in_edges[edges[i].to].insert(mp(edges[i].weight, i));
    finish[root] = true;
    for (int vert = 0; vert < n; vert++)
        if (!finish[vert]) {
            path.clear();
            build_path(metka[vert], -1);
        }
    for (int i = 0; i < n; i++)
        if (i != root) total += edges[parent[i]].weight;
}
void build_path(int v, int last) {
    path.pb(mp(v, last));
    used[v] = true;
    while (metka[edges[in_edges[v].begin()->y].from] == v)
        in_edges[v].erase(in_edges[v].begin());
    delta[v] = in_edges[v].begin()->x;
    int num = in_edges[v].begin()->y;
    int u = metka[edges[num].from];
    if (finish[u]) {
        for (const auto &[v0, nm] : path)
            for (int u0 : komp[v0])
                finish[u0] = true;
        path.pb(mp(u, num));
        for (int i = 0; i < (int) path.size() - 1; i++)
            parent[path[i].x] = path[i + 1].y;
        return;
    }
    if (!used[u]) {
        build_path(u, num);
        return;
    }
    vector <pair <int, int> > cycle;
    int new_vert = u;
    while (true) {
        pair <int, int> A = path.back();
        cycle.pb(A);
        path.pop_back();
        if ((int) komp[A.x].size() > (int) komp[new_vert].size()) new_vert = A.x;
        if (A.x == u) break;
```

```
    }
    int old = cycle.back().y;
    cycle.back().y = num;
    vector <pair <int, int> > otkat;
    for (const auto &[v0, nm] : cycle) {
        if (v0 == new_vert) continue;
        for (int u0 : komp[v0]) {
            otkat.pb(mp(u0, metka[u0]));
            metka[u0] = new_vert, komp[new_vert].pb(u0);
        }
        for (const auto &[w, nums] : in_edges[v0])
            in_edges[new_vert].insert(mp(w - delta[v0] + delta[new_vert], nums));
    }
    build_path(new_vert, old);
    int need = parent[new_vert];
    for (int i = 0; i < (int) cycle.size() - 1; i++)
        parent[cycle[i + 1].x] = cycle[i].y;
    parent[cycle[0].x] = cycle.back().y;
    while (!otkat.empty()) {
        pair <int, int> A = otkat.back();
        otkat.pop_back();
        metka[A.x] = A.y;
    }
    parent[metka[edges[need].to]] = need;
}
private:
int n;
vector <Edge> edges;
int root;
ll total = 0;
vector <int> parent, metka;
vector <vector <int> > komp;
vector <ll> delta;
vector <set <pair <ll, int> > > in_edges;
vector <bool> finish, used;
vector <pair <int, int> > path;
};
}
```

2.10 Сжатие соцветий

```
namespace GeneralMatching {
class GeneralMatching {
public:
GeneralMatching(int _n, const vector <vector <int> > &_g) : n(_n), g(_g) {
    match.resize(n, -1), used.resize(n), base.resize(n), parent.resize(n), blossom.resize(n);
    color.resize(n);
    for (int v = 0; v < n; v++)
        for (int u : g[v])
            if (match[v] == -1 && match[u] == -1)
                match[v] = u, match[u] = v;
    for (int v = 0; v < n; v++)
        if (match[v] == -1)
            find_path(v);
}
void mark_path(int v, int b, int pr) {
    while (base[v] != b) {
        blossom[base[v]] = blossom[base[match[v]]] = true;
        parent[v] = pr, pr = match[v], v = parent[pr];
    }
}
```

```
int lca(int a, int b) {
    color_count++;
    for (a = base[a]; match[a] != -1; a = base[parent[match[a]]]) color[a] = color_count;
    for (b = base[b]; a != b && color[b] != color_count; b = base[parent[match[b]]]);
    return b;
}
void find_path(int root) {
    for (int i = 0; i < n; i++)
        base[i] = i, used[i] = 0, parent[i] = -1;
    queue <int> qu;
    qu.push(root), used[root] = true;
    while (!qu.empty()) {
        int v = qu.front();
        qu.pop();
        for (int u : g[v]) {
            if (base[u] == base[v] || match[v] == u) continue;
            if (u != root && match[u] == -1) {
                while (u != -1) {
                    int old = match[v];
                    match[u] = v, match[v] = u;
                    u = old, v = parent[old];
                }
                return;
            }
            if (u == root || parent[match[u]] != -1) {
                int curbase = lca(v, u);
                for (int i = 0; i < n; i++) blossom[i] = false;
                mark_path(v, curbase, u);
                mark_path(u, curbase, v);
                for (int i = 0; i < n; i++)
                    if (blossom[base[i]]) {
                        base[i] = curbase;
                        if (!used[i]) qu.push(i), used[i] = true;
                    }
            } else if (parent[u] == -1) {
                parent[u] = v, u = match[u];
                qu.push(u), used[u] = true;
            }
        }
    }
}
vector <int> matching() const {
    return match;
}
private:
int n;
vector <vector <int> > g;
vector <int> match, base, parent;
vector <bool> used, blossom;
int color_count = 0;
vector <int> color;
};
}
```

2.11 Дерево доминаторов

```
namespace DominatorsTree {
const int H = 20;
class DominatorsTree {
public:
void dfs(int v) {
```

```
    renum[v] = timer++;
    revnum.pb(v);
    for (int u : g[v])
        if (renum[u] == -1) {
            dfs(u);
            sons[v].pb(u);
        }
    return;
}
int find_parent(int v) {
    if (v == parent[v]) return v;
    int p = parent[v];
    parent[v] = find_parent(p);
    min_sdom[v] = min(min_sdom[v], min_sdom[p]);
    return parent[v];
}
int get_minimum(int v) {
    find_parent(v);
    return min_sdom[v];
}
void unions(int a, int b) {
    parent[b] = a;
    return;
}
void dfsik(int v, int pr) {
    dv[0][v] = mp(pr, v);
    for (int i = 1; i <= H; i++)
        if (dv[i - 1][v].x != -1) {
            dv[i][v].x = dv[i - 1][dv[i - 1][v].x].x;
            int a = dv[i - 1][v].y;
            int b = dv[i - 1][dv[i - 1][v].x].y;
            if (sdom[a] > sdom[b]) swap(a, b);
            dv[i][v].y = a;
        }
        else break;
    if (pr != -1) {
        int cur = v, mn = v;
        for (int i = H; i >= 0; i--)
            if (dv[i][cur].x != -1 && renum[dv[i][cur].x] >= sdom[v]) {
                int a = dv[i][cur].y;
                if (sdom[a] < sdom[mn]) mn = a;
                cur = dv[i][cur].x;
            }
        if (sdom[v] == sdom[mn]) dom[v] = sdom[v];
        else dom[v] = dom[mn];
    }
    for (int u : sons[v])
        dfsik(u, v);
    return;
}
DominatorsTree(int n, const vector <vector <int> > _g, int start) : g(_g) {
    rev.resize(n), sons.resize(n);
    renum.resize(n, -1), dom.resize(n, -1), sdom.resize(n, n);
    for (int v = 0; v < n; v++)
        for (int u : g[v])
            rev[u].pb(v);
    dfs(start);
    parent.resize(n), min_sdom.resize(n);
    for (int i = 0; i < n; i++) parent[i] = i;
    for (int i = (int) revnum.size() - 1; i >= 0; i--) {
        int v = revnum[i];
```

```
        for (int u : rev[v]) {
            if (renum[u] == -1 || u == v) continue;
            if (renum[u] < i) sdom[v] = min(sdom[v], renum[u]);
            else sdom[v] = min(sdom[v], get_minimum(u));
        }
        min_sdom[v] = sdom[v];
        for (int u : sons[v])
            unions(v, u);
    }
    sdom[start] = 0;
    for (int i = 0; i <= H; i++) dv[i].resize(n);
    dfsik(start, -1);
    for (int i = 0; i < n; i++)
        if (dom[i] != -1) dom[i] = revnum[dom[i]];
}
vector <int> dominator_tree() const {
    return dom;
}
private:
vector <vector <int> > g, rev, sons;
vector <int> dom, sdom;
vector <int> renum, revnum;
int timer = 0;
vector <int> parent, min_sdom;
vector <pair <int, int> > dv[H + 2];
};
}
```

2.12 Dynamic CHT

```
struct Line {
    ll k, m, p;
    bool operator < (const Line &o) const { return k < o.k; }
    bool operator < (ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;

    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b);
    }

    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }

    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }

    ll query(ll x) {
        assert(!empty());
```

```
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

2.13 Пересечение матроидов

```
// check(ctaken, 1) -- first matroid
// check(ctaken, 2) -- second matroid
vector<char> taken(m);
while (1) {
    vector<vector<int>> e(m);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            if (taken[i] && !taken[j]) {
                auto ctaken = taken;
                ctaken[i] = 0;
                ctaken[j] = 1;
                if (check(ctaken, 2)) {
                    e[i].push_back(j);
                }
            }
            if (!taken[i] && taken[j]) {
                auto ctaken = taken;
                ctaken[i] = 1;
                ctaken[j] = 0;
                if (check(ctaken, 1)) {
                    e[i].push_back(j);
                }
            }
        }
        }
    vector<int> type(m);
    // 0 -- cant, 1 -- can in \2, 2 -- can in \1
    for (int i = 0; i < m; i++) {
        if (!taken[i]) {
            auto ctaken = taken;
            ctaken[i] = 1;
            if (check(ctaken, 2)) type[i] |= 1;
        }
        if (!taken[i]) {
            auto ctaken = taken;
            ctaken[i] = 1;
            if (check(ctaken, 1)) type[i] |= 2;
        }
    }
    vector<int> w(m);
    for (int i = 0; i < m; i++) {
        w[i] = taken[i] ? ed[i].c : -ed[i].c;
    }
    vector<pair<int, int>> d(m, {INF, 0});
    for (int i = 0; i < m; i++) {
        if (type[i] & 1) d[i] = {w[i], 0};
    }
    vector<int> pr(m, -1);
    while (1) {
        vector<pair<int, int>> nd = d;
        for (int i = 0; i < m; i++) {
            if (d[i].first == INF) continue;
            for (int to : e[i]) {
                if (nd[to] > make_pair(d[i].first + w[to], d[i].second + 1)) {
```

```
                    nd[to] = make_pair(d[i].first + w[to], d[i].second + 1);
                    pr[to] = i;
                }
            }
        }
        if (d == nd) break;
        d = nd;
    }
    int v = -1;
    for (int i = 0; i < m; i++) {
        if ((d[i].first < INF && (type[i] & 2)) && (v == -1 || d[i] < d[v])) v = i;
    }
    if (v == -1) break;
    while (v != -1) {
        sum += w[v];
        taken[v] ^= 1;
        v = pr[v];
    }
    ans[--cnt] = sum;
}
```

2.14 Быстрый ввод-вывод

```
#include <cstdio>

/** Interface */
inline int readChar();
template <class T = int> inline T readInt();
template <class T> inline void writeInt( T x, char end = 0 );
inline void writeChar( int x );
inline void writeWord( const char *s );

/** Read */
static const int buf_size = 4096;
inline int getChar() {
    static char buf[buf_size];
    static int len = 0, pos = 0;
    if (pos == len) pos = 0, len = fread(buf, 1, buf_size, stdin);
    if (pos == len) return -1;
    return buf[pos++];
}
inline int readChar() {
    int c = getChar();
    while (c <= 32)
        c = getChar();
    return c;
}
template <class T>
inline T readInt() {
    int s = 1, c = readChar();
    T x = 0;
    if (c == '-')
        s = -1, c = getChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    return s == 1 ? x : -x;
}

/** Write */
static int write_pos = 0;
static char write_buf[buf_size];
```

```
inline void writeChar( int x ) {
    if (write_pos == buf_size)
        fwrite(write_buf, 1, buf_size, stdout), write_pos = 0;
    write_buf[write_pos++] = x;
}
template <class T>
inline void writeInt( T x, char end ) {
    if (x < 0)
        writeChar('-'), x = -x;

    char s[24];
    int n = 0;
    while (x || !n)
        s[n++] = '0' + x % 10, x /= 10;
    while (n--)
        writeChar(s[n]);
    if (end)
        writeChar(end);
}
inline void writeWord( const char *s ) {
    while (*s)
        writeChar(*s++);
}
struct Flusher {
    ~Flusher() {
        if (write_pos)
            fwrite(write_buf, 1, write_pos, stdout), write_pos = 0;
    }
} flusher;

// cin - 3.02, scanf - 1.2, cin+sync - 0.71
// fastRead getchar - 0.53, fastRead fread - 0.15
```

2.15 Пересечение полуплоскостей

```
namespace Halfplane
{
// bounding box ~ MAXC^3
const ll INF = (ll) 1e18;
const ld EPS = 1e-9;
struct Halfplane
{
    // Ax + By + C >= 0
    int A, B;
    ll C;
    bool outside(pair <ld, ld> point) const {
        return A * point.x + B * point.y + C < -EPS;
    }
    bool operator < (const Halfplane &other) const {
        bool fl1 = (B > 0) || (B == 0 && A > 0);
        bool fl2 = (other.B > 0) || (other.B == 0 && other.A > 0);
        if (fl1 != fl2) return fl1;
        ll v = vecs(mp(A, B), mp(other.A, other.B));
        if (v != 0) return v > 0;
        return 1.0 * C / sqrtl((ll) A * A + (ll) B * B) <
               1.0 * other.C / sqrtl((ll) other.A * other.A + (ll) other.B * other.B);
    }
};
pair <ld, ld> intersect(Halfplane L1, Halfplane L2) {
    ll X = L2.B * L1.C - L1.B * L2.C;
```

```
    ll Y = L1.A * L2.C - L2.A * L1.C;
    ll Z = (ll) L2.A * L1.B - (ll) L1.A * L2.B;
    return mp(1.0 * X / Z, 1.0 * Y / Z);
}
vector <pair <ld, ld> > intersection(const vector <Halfplane> &halfplanes) {
    /*vector <Halfplane> bounding_box = { {0, 1, INF}, {-1, 0, INF}, {0, -1, INF}, {1, 0, INF}, };*/
    vector <Halfplane> hp = halfplanes;
    //hp.insert(hp.end(), bounding_box.begin(), bounding_box.end());

    sort(hp.begin(), hp.end());
    deque <Halfplane> dq;
    for (const Halfplane &h : hp) {
        while ((int) dq.size() >= 2 && h.outside(intersect(dq.rbegin()[0], dq.rbegin()[1]))) dq.pop_back();
        while ((int) dq.size() >= 2 && h.outside(intersect(dq[0], dq[1]))) dq.pop_front();
        if (!dq.empty() && vecs(mp(h.A, h.B), mp(dq.back().A, dq.back().B)) == 0)
        {
            if (scal(mp(h.A, h.B), mp(dq.back().A, dq.back().B)) < 0) return {};
            continue;
        }
        dq.pb(h);
    }

    while ((int) dq.size() > 2 && dq[0].outside(intersect(dq.rbegin()[0], dq.rbegin()[1]))) dq.pop_back();
    while ((int) dq.size() > 2 && dq.back().outside(intersect(dq[0], dq[1]))) dq.pop_front();

    if ((int) dq.size() < 3) return {};
    vector <pair <ld, ld> > result((int) dq.size());
    for (int i = 0; i < (int) dq.size(); i++) {
        int nxt = (i + 1) % (int) dq.size();
        if (vecs(mp(dq[i].A, dq[i].B), mp(dq[nxt].A, dq[nxt].B)) <= 0) return {};
        result[i] = intersect(dq[i], dq[nxt]);
    }
    return result;
}
}
```

2.16 Симплекс-метод

```
#pragma once
/* Thanks to Um_nik! problem: x >= 0 and A x <= b
 * returns: [inf if unbounded above]; [-inf if infeasible] [max (cT x) otherwise]
 */
template<class db>
struct Simplex {
    using VD = vector<db>;
    using VVD = vector<VD>;
    // EPS = 1e-7 for float, EPS = 1e-12 for double
    const db EPS, INF = numeric_limits<db>::infinity();
    int m, n; vi B, N; VVD D;
    Simplex(const VVD& A, const VD& b, const VD& c, db eps) : m(b.size()), n(c.size()), B(m), N(n + 1, -1),
        for (int i = 0; i < m; ++i) {
            copy(A[i].begin(), A[i].end(), D[i].begin());
            B[i] = n + i;
            D[i][n] = -1;
            D[i][n + 1] = b[i];
        }
        for (int j = 0; j < n; ++j) {
            N[j] = j;
            D[m][j] = -c[j];
        }
        D[m + 1][n] = 1;
```

```
    }
    void pivot(int r, int s) {
        for (int i = 0; i < m + 2; ++i) if (i != r)
            for (int j = 0; j < n + 2; ++j) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] / D[r][s];
            for (int j = 0; j < n + 2; ++j) if (j != s)
                D[r][j] /= D[r][s];
            for (int i = 0; i < m + 2; ++i) if (i != r)
                D[i][s] /= -D[r][s];
            D[r][s] = 1. / D[r][s];
            swap(B[r], N[s]);
    }
    bool simplex(int p) {
        int x = m + (p == 1);
        while (true) {
            int s = -1; db v = -1;
            for (int j = 0; j <= n; ++j) {
                if (p == 2 && N[j] == -1) continue;
                db q = 0;
                for (int k = 0; k <= m; ++k)
                    q += pow(D[k][j], 2);
                q = max(q, EPS);
                db vj = D[x][j] / sqrt(q);
                if (s == -1 || mp(vj, N[j]) < mp(v, N[s])) s = j, v = vj;
            }
            if (D[x][s] >= -EPS) return true;
            int r = -1;
            for (int i = 0; i < m; ++i) if (D[i][s] > EPS)
                if (r == -1 || mp(D[i][n + 1] / D[i][s], B[i]) < mp(D[r][n + 1] / D[r][s], B[r])) r = i;
            if (r == -1) return false;
            pivot(r, s);
        }
    }
    db solve(VD& x) {
        int r = 0;
        for (int i = 1; i < m; ++i) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] <= -EPS) {
            pivot(r, n);
            if (!simplex(1) || D[m + 1][n + 1] < -EPS) return -INF;
            for (int i = 0; i < m; ++i) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; ++j)
                    if (s == -1 || mp(D[i][j], N[j]) < mp(D[i][s], N[s])) s = j;
                pivot(i, s);
            }
        }
        if (!simplex(2)) return INF;
        x = VD(n);
        for (int i = 0; i < m; ++i) if (B[i] < n)
            x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};
```

2.17 Дерево палиндромов

```
struct PalindromeAutomaton {
    int alpha;
    char a0;
    vi length, occurences, suflink;
    vvi go;
```

```
    int n() const {
        return length.size();
    }
    PalindromeAutomaton(const int alpha, const char a0, const string& s,
                        const int count_occurences_starting_with) :
            alpha(alpha), a0(a0), length{-1, 0}, occurences{0, 0}, suflink{-1, 0}, go(2, vi(alpha, -1)) {
        int state = 1;
        for (int symbol_index = 0; symbol_index < ssize(s); ++symbol_index)
            state = add_symbol(s, state, symbol_index, symbol_index >= count_occurences_starting_with);
    }
    int add_symbol(const string& s, int state, int symbol_index, bool count_occurences) {
        int symbol = s[symbol_index] - a0;
        while (s[symbol_index] != s[symbol_index - 1 - length[state]])
            state = suflink[state];
        if (go[state][symbol] == -1) {
            go[state][symbol] = n();
            length.push_back(length[state] + 2);
            occurences.push_back(0);
            suflink.push_back(1);
            go.push_back(vi(alpha, -1));
            if (state) {
                int current_state = state;
                do {
                    current_state = suflink[current_state];
                } while (s[symbol_index] != s[symbol_index - 1 - length[current_state]]);
                suflink.back() = go[current_state][symbol];
            }
        }
        int new_state = go[state][symbol];
        if (count_occurences) {
            ++occurences[new_state];
        }
        return new_state;
    }
    int recalculate_occurences(int max_length) {
        int ans = 0;
        for (int i = n() - 1; i >= 2; --i) {
            occurences[suflink[i]] = sum(occurences[suflink[i]], occurences[i]);
            if (length[i] <= max_length)
                ans = sum(ans, prod(length[i], prod(occurences[i], occurences[i])));
        }
        return ans;
    }
};
```

2.18 1D1D

```
pll find_ans_1d1d(const vi& a, const int k,
                  const ll lambda) {
  const int n = a.size();
  vl s(n + 1);
  for (int i = 0; i < n; ++i)
    s[i + 1] = s[i] + a[i];
  auto cost = [&](int j, int i) {
    return lambda +
           (ll)(s[i] - s[j]) * (i - j - 1);
  };
  deque<pii> d;
  d.emplace_back(1, 0);
  vl dp(n + 1, INF);
  vl ds(n + 1, INF);
  dp[0] = 0;
  ds[0] = 0;
  auto cut = [&](int j, int i) {
    return dp[j] + cost(j, i);
  };
  for (int j = 0; j < n; ++j) {
    // dp[j + 1] = tmp[j + 1][j] =
    // = dp[k[j + 1][j]] + cost(k[j + 1][j], j + 1)
    dp[j + 1] = cut(d.front().Y, j + 1);
    ds[j + 1] = 1 + ds[d.front().Y];
    if (j == n - 1) break;
    ++d.front().X;
    if (d.size() >= 2 && d[1].X == d[0].X) d.pop_front();
    int l = j + 1; int r = n + 1;
    while (!d.empty()) {
      if (cut(d.back().Y, d.back().X) >
          cut(j + 1, d.back().X)) {
        r = d.back().X; d.pop_back();
      } else {
        l = d.back().X; break;
      }
    }
    while (r - l >= 2) {
      int m = (l + r) / 2;
      if (cut(d.back().Y, m) > cut(j + 1, m)) r = m;
      else l = m;
    }
    if (r <= n)
      d.emplace_back(r, j + 1);
  }
  return { dp.back(), ds.back() };
}
```

2.19 Сумма линейных функций по модулю

```
// sum(i=0..n-1) (a + b * i) div m
ll solve(ll n, ll a, ll b, ll m) {
    if (b == 0)
       return n * (a / m);
    if (a >= m)
       return n * (a / m) + solve(n, a % m, b, m);
    if (b >= m)
       return n * (n - 1) / 2 * (b / m) +
                  solve(n, a, b % m, m);
    return solve((a + b * n) / m,
                 (a + b * n) % m, m, b);
}
```

2.20 Венгерка

```
// graph matrix - a[][]
// 1-indexation
// p - matching
vector<int> u (n+1), v (m+1);
vector<int> p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
        used[j0] = true;
        int i0 = p[j0],  delta = INF,  j1;
        for (int j=1; j<=m; ++j)
            if (!used[j]) {
                int cur = a[i0][j]-u[i0]-v[j];
                if (cur < minv[j]) {
                    minv[j] = cur;
                    way[j] = j0;
                }
                if (minv[j] < delta) {
                    delta = minv[j];
                    j1 = j;
                }
            }
        for (int j=0; j<=m; ++j)
            if (used[j]) {
                u[p[j]] += delta;
                v[j] -= delta;
            }
            else
                minv[j] -= delta;
        j0 = j1;
    } while (p[j0] != 0);
    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}
```

2.21 Берликамп

```
#define pb push_back
#define SZ 233333
const int MOD=1e9+7; //or any prime
ll qp(ll a,ll b)
{
  ll x=1; a%=MOD;
  while(b)
  {
    if(b&1) x=x*a%MOD;
    a=a*a%MOD; b>>=1;
  }
  return x;
}
```

```
namespace linear_seq {
inline vector<int> BM(vector<int> x)
{
  //ls: (shortest) relation sequence
    // (after filling zeroes) so far
  //cur: current relation sequence
  vector<int> ls,cur;
  //lf: the position of ls (t')
  //ld: delta of ls (v')
  int lf,ld;
  for(int i=0;i<int(x.size());++i)
  {
    ll t=0;
    //evaluate at position i
    for(int j=0;j<int(cur.size());++j)
      t=(t+x[i-j-1]*(ll)cur[j])%MOD;
    if((t-x[i])%MOD==0) continue; //good so far
    //first non-zero position
    if(!cur.size())
    {
      cur.resize(i+1);
      lf=i; ld=(t-x[i])%MOD;
      continue;
    }
    //cur=cur-c/ld*(x[i]-t)
    ll k=-(x[i]-t)*qp(ld,MOD-2)%MOD/*1/ld*/;
    vector<int> c(i-lf-1); //add zeroes in front
    c.pb(k);
    for(int j=0;j<int(ls.size());++j)
      c.pb(-ls[j]*k%MOD);
    if(c.size()<cur.size()) c.resize(cur.size());
    for(int j=0;j<int(cur.size());++j)
      c[j]=(c[j]+cur[j])%MOD;
    //if cur is better than ls, change ls to cur
    if(i-lf+(int)ls.size()>=(int)cur.size())
      ls=cur,lf=i,ld=(t-x[i])%MOD;
    cur=c;
  }
  for(int i=0;i<int(cur.size());++i)
    cur[i]=(cur[i]%MOD+MOD)%MOD;
  return cur;
}
int m; //length of recurrence
//a: first terms, h: relation
ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
//calculate p*q mod f
inline void mull(ll*p,ll*q)
{
  for(int i=0;i<m+m;++i) t_[i]=0;
  for(int i=0;i<m;++i) if(p[i])
    for(int j=0;j<m;++j)
      t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
  for(int i=m+m-1;i>=m;--i)
        if(t_[i])
    //miuns t_[i]x^{i-m}(x^m-\sum_{j=0}^{m-1}
        // x^{m-j-1}h_j)
    for(int j=m-1;~j;--j)
      t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
  for(int i=0;i<m;++i) p[i]=t_[i];
}
inline ll calc(ll K)
{
  for(int i=m;~i;--i)
    s[i]=t[i]=0;
  //init
  s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
  //binary-exponentiation
  while(K)
  {
    if(K&1) mull(s,t);
    mull(t,t); K>>=1;
  }
  ll su=0;
  for(int i=0;i<m;++i) su=(su+s[i]*a[i])%MOD;
  return (su%MOD+MOD)%MOD;
}
inline int work(vector<int> x,ll n)
{
  if(n<int(x.size())) return x[n];
  vector<int> v=BM(x); m=v.size(); if(!m) return 0;
  for(int i=0;i<m;++i) h[i]=v[i],a[i]=x[i];
  return calc(n);
}
}
using linear_seq::work;
int main()
{
    vector<int> x = {1, 2, 4, 8, 16};
    for (int i = 0; i < 10; i++)
        cout << work(x, i) << endl;
}
```

2.22 Правила

1. Перед написанием кода проверить решение на сэмплах.

2. Если задача неочевидная, рассказать решение другому перед написанием, подумать над тестами перед написанием.

3. В первый час 5 минут на исправление бага. Во второй час 10. В третий час 15.

4. После 3 неправильных посылок необходимо обсудить с кем-то решение и провести полное тестирование перед посылкой.

5. В первой половине контеста скипать задачу, если ничего не получается 15 минут.

6. Все задачи должны быть кем-то прочитаны в первые полтора часа.

7. Если кто-то уже прочитал задачу с длинной легендой и не занят, то узнай условие у него вместо того, чтобы читать самому, если он уверен, что правильно понял задачу.

8. В табличке отмечать краткое описание задачи в пару слов.

9. Через 3,5 часа все должны знать все несданные задачи.

10. Перед написанием адовой жести надо пообмениваться идеями по остальным задачам с остальными.

11. Пытаться применять стандартные трюки, решать более простые версии задачи.

12. 30 секунд ни на что не влияют, можно остановиться и подумать над происходящим.

13. Если придумал решение, но не можешь сейчас написать, продумай детали, проверь, все ли правильно.

14. Если не решается, посмотри на тесты, попридумывать тесты, возможно, ты не видишь какую-то очевидную вещь.

15. Если ты думал, что уже точно доказал свое решение, а после этого вылез еще какой-то случай/баг, то надо рассказать решение другому / не подходить к компу еще 15 минут.

16. Когда в конце контеста кто-то пишет код, другой пока делает тесты, чекает, нет ли других случаев.

17. После посылки минуту не чекать, зашло ли.