Содержание

1 Common **2**
1 Vimrc . . . 2
2 Bash . . . 2
3 Template . . . 2
4 stress.sh . . . 2
5 Java . . . 3
6 Заметки . . . 4
7 Много делителей . . . 4
8 Формулы . . . 4

2 Numbers theory **7**
9 Fourier transform . . . 7
10 Factorization . . . 7
11 Chinese Remainder Theorem . . . 8

3 Graph **9**
12 Dominator tree . . . 9
13 Hungarian algorithm . . . 9
14 Edmonds . . . 10
15 Dinic and Gomory-Hu . . . 10
16 Linking-cutting tree . . . 12

4 Geometry **13**
17 halfplanes . . . 13
18 Delaunay . . . 14

5 Strings **16**
19 Aho-Corasick . . . 16
20 Palindromic Tree . . . 17
21 Suffix Automaton . . . 17
22 Suffix Tree . . . 18
23 Prefix function, Duval algorithm . . . 18

6 Algebra **19**
24 Simplex method . . . 19

7 Structures **20**
25 Treap . . . 20
26 Two Chinese . . . 21
27 Hexagonal paper . . . 23
28 Triangular paper . . . 24

1 Vimrc

```
filetype plugin indent on
syntax on

set nocp hls is et sw=2 sts=2 ts=2 nu ru so=5
nn <c-j> :w <bar> :!g % <cr>
nn <c-l> :w <bar> :!g % 2> err <cr>
nn <c-o> :! <cr>
nn <silent> <space> :nohls <bar> :echo <cr>

set clipboard=unnamedplus
```

2 Bash

```
export PATH=~/bin:$PATH


#!/bin/bash
g++ -Wall -DDEBUG -O2 -std=c++11 $1
              -o exec || exit 1
ulimit -s 64000
echo Run
./exec || exit 1
echo Out
tail -20 *.out
```

```
   ***template
#ifdef DEBUG
#define _GLIBCXX_DEBUG
#endif

#include <bits/stdc++.h>

using namespace std;

mt19937 mrand(random_device{} ());

int rnd(int x) {
| return mrand() % x;
}

typedef long double ld;
typedef long long ll;

#ifdef DEBUG
#define eprintf(...) fprintf(stderr, __VA_ARGS__),
fflush(stderr)
#else
#define eprintf(...) ;
#endif

#define pb push_back
#define mp make_pair
#define sz(x) ((int) (x).size())
#define TASK "text"

const int inf = (int) 1.01e9;
const ld eps = 1e-9;
const ld pi = acos((ld) -1.0);

int n;

int read() {
| if (scanf("%d", &n) < 1) {
| | return 0;
| }
| return 1;
}

void solve() {
}

int main() {
#ifdef DEBUG
| assert(freopen(TASK ".out", "w", stdout));
| assert(freopen(TASK ".in", "r", stdin));
#endif

| while (read()) {
| | solve();
| | eprintf("Time %.2f\n", (double) clock() /
CLOCKS_PER_SEC);
| }
| return 0;
}
```

4 stress.sh

```
#!/bin/bash
rm text.in text.out
g main.cpp
cp exec exec1
g main2.cpp
cp exec exec2
g gen.cpp
cp exec execg

for ((i=1;i<=10000;++i)); do
  ./execg || exit 1
  ./exec2 || exit 1
  cp text.out text.ans
  ./exec1 || exit 1
  diff text.out text.ans || exit 1
  echo Ok $i
done
```

5 Java

```
import java.util.*;
import java.io.*;
import java.math.*;

class FastScanner {
  BufferedReader br;
  StringTokenizer st;

  FastScanner(InputStream in) {
    br = new BufferedReader(new InputStreamReader(in));
  }

  String next() {
      while (st == null || !st.hasMoreTokens()) {
          try {
              st = new StringTokenizer(br.readLine());
          } catch (IOException e) {
              e.printStackTrace();
          }
      }
      return st.nextToken();
  }

  int nextInt() {
      return Integer.parseInt(next());
  }

  BigInteger nextBigInteger() {
    return new BigInteger(next());
  }
}

public class a {
  public static void main(String[] args) throws Exception {
    FastScanner in = new FastScanner(System.in/*new FileInputStream("text.in")*/);
    PrintWriter out = new PrintWriter(System.out/*new FileOutputStream("text.in")*/);

    Solver solver = new Solver();
    while (solver.solve(in, out)) {
    }

    out.close();
  }
}

class Solver {
  boolean solve(FastScanner in, PrintWriter out) {
    int n;
    try {
      n = in.nextInt();
    } catch (Exception e) {
      return false;
    }
    out.println(n);
    out.flush();
    return true;
  }
}
```

6 Заметки

- 1 января 2000 года — суббота, 1 января 1900 года — понедельник, 14 апреля 1961 года — пятница
- Високосные года: если $400|a$, либо если $4|a$ но не $100|a$.

7 Много делителей

- ≤ 20: $d(12) = 6$
- ≤ 50: $d(48) = 10$
- ≤ 100: $d(60) = 12$
- ≤ 1000: $d(840) = 32$
- $\le 10^4$: $d(9\,240) = 64$
- $\le 10^5$: $d(83\,160) = 128$
- $\le 10^6$: $d(720\,720) = 240$
- $\le 10^7$: $d(8\,648\,640) = 448$
- $\le 10^8$: $d(91\,891\,800) = 768$
- $\le 10^9$: $d(931\,170\,240) = 1344$
- $\le 10^{11}$: $d(97\,772\,875\,200) = 4032$
- $\le 10^{12}$: $d(963\,761\,198\,400) = 6720$
- $\le 10^{15}$: $d(866\,421\,317\,361\,600) = 26880$
- $\le 10^{18}$: $d(897\,612\,484\,786\,617\,600) = 103680$

8 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 $-\frac{\pi}{2}$ до $\frac{\pi}{2}$), φ — долготы (от $-\pi$ до π).
- Объём 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.
- ld Simple() { return F(0); }
- ld Simpson() { return (F(-1) + 4 * F(0) + F(1)) / 6; }
- ld Runge2() { return (F(-sqrt((ld) 1.0 / 3)) + F(sqrt((ld) 1.0 / 3))) / 2; }
- ld Runge3() { return (F(-sqrt((ld) 3.0 / 5)) * 5 + F(0) * 8 + F(sqrt((ld) 3.0 / 5)) * 5) / 18; }

Метод Рунге-Кутты: Пусть мы решаем диффур $y' = f(x, y)$, $y_n = y(nh)$. Тогда $y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$, где

$$k_1 = f(x_n, y_n),$$

$$k_2 = f(x_n + h/2, y_n + h/2k_1),$$

$$k_3 = f(x_n + h/2, y_n + h/2k_2),$$

$$k_4 = f(x_n + h, y_n + hk_3).$$

Числа Стирлинга: $s(n, k)$ — количество перестановок на n элементах, в которых ровно k циклов; $S(n, k)$ — количество способов разбить n-элементное множество на k непустых подмножеств.

$$s(n, k) = (n - 1)s(n - 1, k) + s(n - 1, k - 1),$$

$$S(n, k) = kS(n - 1, k) + S(n - 1, k - 1),$$

$$x^{\underline{n}} = x(x - 1)\cdots(x - n + 1) = \sum_{i=1}^{n}(-1)^{n-i}s(n, i)x^i.$$

$$x^n = \sum_{i=0}^{n} S(n, i)x^{\underline{i}}.$$

$$\sum_n S(n,k)x^n = \frac{x^k}{(1-x)(1-2x)\dots(1-kx)};$$

$$\sum_n S(n,k)\frac{x^n}{n!} = \frac{1}{k!}(\exp(x)-1)^k.$$

Интегралы:

$$\int \frac{1}{\sqrt{x^2-1}}\,dx = \ln\left(\sqrt{x^2-1}+x\right).$$

$$\int \sqrt{x^2-1}\,dx = \frac{1}{2}\left(x\sqrt{x^2-1} - \ln\left(\sqrt{x^2-1}+x\right)\right).$$

$$\int \sqrt{1-x^2}\,dx = \frac{1}{2}\left(x\sqrt{1-x^2} + \arcsin x\right).$$

$$\int \sqrt{1+x^2}\,dx = \frac{1}{2}\left(x\sqrt{1-x^2} + \operatorname{arcsh} x\right).$$

$$\int \frac{1}{\sin x}\,dx = -\ln\left(\operatorname{ctg} x + \frac{1}{\sin x}\right).$$

$$\int \frac{1}{\cos x}\,dx = \ln\left(\operatorname{tg} x + \frac{1}{\cos x}\right).$$

Числа Белла (динамика по компонентам связности):

i	B_i	i	B_i
0	1	12	4,213,597
1	1	13	27,644,437
2	2	14	190,899,322
3	5	15	1,382,958,545
4	15	16	10,480,142,147
5	52	17	82,864,869,804
6	203	18	682,076,806,159
7	877	19	5,832,742,205,057
8	4,140	20	51,724,158,235,372
9	21,147	21	474,869,816,156,751
10	115,975	22	4,506,715,738,447,323
11	678,570	23	44,152,005,855,084,346

Число разбиений на неубывающие последовательности натуральных слагаемых:

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	5,604	40	37,338	50	204,226
1	1	11	56	21	792	31	6,842	41	44,583	51	239,943
2	2	12	77	22	1,002	32	8,349	42	53,174	52	281,589
3	3	13	101	23	1,255	33	10,143	43	63,261	53	329,931
4	5	14	135	24	1,575	34	12,310	44	75,175	54	386,155
5	7	15	176	25	1,958	35	14,883	45	89,134	55	451,276
6	11	16	231	26	2,436	36	17,977	46	105,558	56	526,823
7	15	17	297	27	3,010	37	21,637	47	124,754	57	614,154
8	22	18	385	28	3,718	38	26,015	48	147,273	58	715,220
9	30	19	490	29	4,565	39	31,185	49	173,525	59	831,820

$$p(100) = 190,569,292$$

Объёмы многомерных шаров:

$$V_{2k} = \frac{\pi^k}{k!}, \quad V_{2k+1} = \frac{2^{k+1}\pi^k}{(2k+1)!!}.$$

Код Грея: $n \operatorname{xor} \frac{n}{2}$.
$F(n)$ — n-ое число Фибоначчи ($F(0)=0, F(1)=1$).

$$F(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Пентагональная теорема Эйлера:

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{n=-\infty}^{\infty}(-1)^n x^{n(3n+1)/2}.$$

$$F_1 + F_2 + F_3 + \dots + F_n = F_{n+2} - 1$$

$$F_1^2 + F_2^2 + F_3^2 + \dots + F_n^2 = F_n F_{n+1}$$

$$F_n^2 - F_{n+1}F_{n-1} = (-1)^{n-1}$$

$$\gcd(F_n, F_m) = F_{\gcd(n,m)}$$

Gambler's ruin. У первого игрока есть n_1 монет, у второго n_2. На каждом шаге с вероятностью p второй отдаёт одну монету первому, а с вероятностью $q = 1 - p$ первый отдаёт одну монету второму. Игра заканчивается, когда у кого-нибудь не остаётся монет. Тогда первый выигрывает с вероятностью $\frac{1-\left(\frac{p}{q}\right)^{n_1}}{1-\left(\frac{p}{q}\right)^{n_1+n_2}}$.

***fft

```
struct comp {
    ld x, y;

    comp(ld x = 0, ld y = 0) : x(x), y(y) {}

    comp operator + (const comp &b) const {
        return comp(x + b.x, y + b.y);
    }

    comp operator - (const comp &b) const {
        return comp(x - b.x, y - b.y);
    }

    comp operator * (const comp &b) const {
        return comp(x * b.x - y * b.y, x * b.y + y * b.x);
    }
};

namespace FFT {
    const int maxn = (1 << 20);

    int rev[maxn], n, logn;
    comp a[maxn];
    comp w[maxn];

    int wasn = -1;

    void fft(comp *a) {
        if (wasn != n) {
            wasn = n;
            rev[0] = 0;
            comp tomult;
            w[1] = 1;
            for (int i = 1, j = -1; i < n; i++) {
                if (!(i & (i - 1))) {
                    j++;
                    tomult = comp(cos(pi / 2 / i), sin(pi / 2 /
i));
                }
                if (i < n / 2) {
                    w[i * 2] = w[i];
                    w[i * 2 + 1] = w[i] * tomult;
                }
                rev[i] = rev[i ^ (1 << j)] ^ (1 << (logn - 1 -
j));
            }
        }

        for (int i = 0; i < n; i++) {
            if (rev[i] < i) {
                swap(a[i], a[rev[i]]);
            }
        }

        for (int i0 = 1, pos = 1; i0 < n; i0 <<= 1) {
            for (int i = 0; i < n; i += i0) {
                for (int j = i + i0, it = 0; it < i0; it++, j++,
i++) {
                    comp toadd = a[j] * w[pos + it];
                    a[j] = a[i] - toadd;
                    a[i] = a[i] + toadd;
                }
            }
            pos += i0;
        }
    }

    void mult(int n1, int *a1, int n2, int *a2, ll *res) {
        for (n = 1, logn = 0; n < n1 + n2 - 1; n <<= 1,
logn++);

        for (int i = 0; i < n; i++) {
            a[i].x = (i < n1) ? a1[i] : 0;
            a[i].y = (i < n2) ? a2[i] : 0;
        }
        fft(a);
        for (int i = 0; i <= n - i; i++) {
            int j = i ? (n - i) : 0;
            auto c = a[i] * a[i], d = a[j] * a[j];
            a[i] = comp((c + d).y / 4, (d - c).x / 4);
            a[j] = comp(a[i].x, -a[i].y);
        }
        reverse(a + 1, a + n);
        fft(a);
        for (int i = 0; i < n1 + n2 - 1; i++) {
            res[i] = (ll) round(a[i].x / n);
        }
    }
};
```

***factor

```
ll mod;

void add(ll &x, ll y) {
    if ((x += y) >= mod) {
        x -= mod;
    }
}

ll mult(ll a, ll b) {
    ll res = a * b - (ll) ((ld) a * b / mod) * mod;
    if (res < 0) {
        res += mod;
    }
    if (res >= mod) {
        res -= mod;
    }
    return res;
}

ll gcd(ll a, ll b) {
    if (!b) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}

ll power(ll a, ll p) {
    ll res = 1;
    while (p) {
        if (p & 1) {
            res = mult(res, a);
        }
        a = mult(a, a);
        p >>= 1;
    }
    return res;
}

//assume p > 1
bool isprime(ll p) {
    const int a[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 0};
    ll d = p - 1;
    int cnt = 0;
    while (!(d & 1)) {
        d >>= 1;
        cnt++;
    }
    mod = p;
    for (int i = 0; a[i]; i++) {
```

```
        if (p == a[i]) {
            return true;
        }
        if (!(p % a[i])) {
            return false;
        }
    }
    for (int i = 0; a[i]; i++) {
        ll cur = power(a[i], d);
        if (cur == 1) {
            continue;
        }
        bool good = false;
        for (int j = 0; j < cnt; j++) {
            if (cur == p - 1) {
                good = true;
                break;
            }
            cur = mult(cur, cur);
        }
        if (!good) {
            return false;
        }
    }
    return true;
}

vector<pair<ll, int> > factor;

void pollard(ll n) {
    assert(n > 1);
    if (isprime(n)) {
        factor.pb(mp(n, 1));
        return;
    }
    mod = n;
    while (true) {
        ll x = rnd(n);
        ll y = x;
        int steps = 1 << 18;
        ll togcd = 1, g = 0;
        const int k = 1 << 7;
        bool easy = false;
        ll prx = x, pry = y;
        for (int i = 1; i < steps; i++) {
            x = mult(x, x);
            add(x, 1);
            togcd = mult(togcd, abs(x - y) % n);
            if (!(i & (i - 1))) {
                y = x;
            }
            if (easy || !(i & (k - 1))) {
                g = gcd(togcd, n);
                if (!easy && g == n) {
                    easy = true;
                    i -= k;
                    x = prx;
                    y = pry;
                } else if (g > 1) {
                    break;
                }
                togcd = 1;
                prx = x, pry = y;
            }
        }
        if (1 < g && g < n) {
            pollard(g);
            pollard(n / g);
            return;
        }
    }
}

const int maxk = 1100;
//big primes can be not grouped
vector<pair<ll, int> > factorize(ll n) {
    factor.clear();
    for (ll i = 2; i < maxk && i * i <= n; i++) {
        if (!(n % i)) {
            int cnt = 0;
            while (n % i == 0) {
                n /= i;
                cnt++;
            }
            factor.pb(mp(i, cnt));
        }
    }
    if (n > 1) {
        pollard(n);
    }
    return factor;
}



   ***crt
ll gcdext(ll a, ll b, ll & x, ll & y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    } else {
        ll res = gcdext(b, a % b, y, x);
        y -= x * (a / b);
        return res;
    }
}

bool crt(ll r1, ll m1, ll r2, ll m2, ll &r, ll &m) {
    ll g = gcd(m1, m2);
    if ((r2 - r1) % g != 0) {
        return false;
    }
    ll c1, c2;
    assert(gcdext(m1 / g, m2 / g, c1, c2) == 1);
    assert(c1 * (m1 / g) + c2 * (m2 / g) == 1);
    ll a = c1;
    a *= (r2 - r1) / g;
    a %= (m2 / g);
    m = m1 / g * m2;
    r = a * m1 + r1;
    r = r % m;
    if (r < 0) {
        r += m;
    }
    assert(r % m1 == r1 && r % m2 == r2);
    return true;
}

//x^p = a (mod), p is prime, 0 <= a < mod
vector<int> getrootsprime(int a, int p) {
    if (!a) {
        return {0};
    }
    int k = 0, pk = 1;
    int r = mod - 1;
    while (r % p == 0) {
        k++;
        pk = pk * p;
        r /= p;
    }
    if (k) {
        if (power(a, (mod - 1) / p) != 1) {
            return {};
```

```
    }
  }
  ll x, y;
  gcdext(p, r, x, y);
  if (x < 0) {
    x += r;
  }
  int res = power(a, x);
  if (!k) {
    return {res};
  }
  assert(mod >= 3);
  int inva = inv(a);
  int w;
  for (w = 1; power(w, (mod - 1) / p) == 1; w++);
  w = power(w, r);
  int bigw = power(w, pk / p);
  int topower = pk / p;
  for (int i = 0; i < k - 1; i++) {
    topower /= p;
    int cur = mult(power(res, p), inva);
    cur = power(cur, topower);
    while (cur != 1) {
      cur = mult(cur, bigw);
      res = mult(res, w);
    }
    w = power(w, p);
  }
  vector<int> ans;
  for (int i = 0; i < p; i++) {
    ans.pb(res);
    res = mult(res, bigw);
  }
  return ans;
}
```

***dominator

```
namespace DominatorTree {
  const int maxn = (int) 1e6;

  int n;
  vector<vector<int> > g, rg;

  int treepar[maxn];
  int par[maxn], label[maxn];
  int sdom[maxn];
  int perm[maxn], tin[maxn], t, used[maxn];

  pair<int, int> getpar(int v) {
    if (par[v] == v) {
      return mp(v, label[v]);
    } else {
      auto res = getpar(par[v]);
      par[v] = res.first;
      if (sdom[label[v]] < sdom[res.second]) {
        res.second = label[v];
      } else {
        label[v] = res.second;
      }
      return res;
    }
  }

  void dfs(int v) {
    used[v] = 1;
    perm[t] = v;
    tin[v] = t++;
    for (int to : g[v]) {
      if (!used[to]) {
        treepar[to] = v;
        dfs(to);
      }
    }
  }

  vector<int> dominator(const vector<vector<int> > & g_)
{
    g = g_;
    n = sz(g);
    rg.clear();
    rg.resize(n);
    for (int i = 0; i < n; i++) {
      for (auto j : g[i]) {
        rg[j].pb(i);
      }
    }
    for (int i = 0; i < n; i++) {
      used[i] = 0;
      par[i] = i;
      label[i] = sdom[i] = i;
    }
    vector<vector<int> > sdomof(n);
    t = 0;
    dfs(0);
    vector<int> idom(n, -1);
    for (int i = n - 1; i >= 0; i--) {
      int curv = perm[i];
      for (auto j : sdomof[i]) {
        idom[j] = getpar(j).second;
      }
      for (auto from : rg[curv]) {
        sdom[i] = min(sdom[i],
sdom[getpar(tin[from]).second]);
      }
      sdomof[sdom[i]].pb(i);
      if (i) {
        par[i] = tin[treepar[curv]];
      }
    }
    for (int i = 1; i < n; i++) {
      if (sdom[i] != idom[i]) {
        idom[i] = idom[idom[i]];
      }
    }
    vector<int> tmp(n, -1);
    for (int i = 1; i < n; i++) {
      tmp[perm[i]] = perm[idom[i]];
    }
    return tmp;
  }
}
```

***hungary

```
const int maxn = 100 + 10;
int addr[maxn], addc[maxn];
int pr[maxn];
int minc[maxn], wherec[maxn];
int used[maxn];

pair<int, int> hung(int n, int m, int a[maxn][maxn], int
ans[maxn]) {
  // n <= m !!!
  for (int i = 0; i < n; ++i) {
    ans[i] = -1;
    addr[i] = 0;
  }
  for (int i = 0; i <= m; ++i) {
    addc[i] = 0;
    pr[i] = -1;
  }
  int flow = 0;
  for (int r0 = 0; r0 < n; ++r0) {
```

```
    for (int i = 0; i < m; ++i) {
      used[i] = 0;
      minc[i] = inf, wherec[i] = -1;
    }
    int c = m;
    pr[c] = r0;
    while (1) {
      used[c] = 1;
      int r = pr[c];
      if (r == -1) {
        break;
      }
      int del = inf, nc = -1;
      for (int j = 0; j < m; ++j) {
        if (used[j]) {
          continue;
        }
        int value = a[r][j] + addr[r] + addc[j];
        if (minc[j] > value) {
          minc[j] = value, wherec[j] = c;
        }
        if (del > minc[j]) {
          del = minc[j];
          nc = j;
        }
      }
      assert(nc != -1);
      for (int j = 0; j <= m; ++j) {
        if (used[j]) {
          addc[j] += del;
          addr[pr[j]] -= del;
        } else {
          minc[j] -= del;
        }
      }
      c = nc;
    }
    ++flow;
    for (; c != m; c = wherec[c]) {
      pr[c] = pr[wherec[c]];
    }
  }
  for (int i = 0; i < m; ++i) {
    if (pr[i] != -1) {
      ans[pr[i]] = i;
    }
  }
  return mp(flow, addc[m]);
}
```

***edmonds

```
int match[maxn];
int st[maxn];

int base[maxn];
int pr[maxn];

int used[maxn];
int was[maxn];

void edmonds() {
  int maxw = 0;
  for (int i = 0; i < n; ++i) {
    match[i] = -1;
    was[i] = 0;
  }

  for (int root = 0; root < n; ++root) {
    if (match[root] != -1) {
      continue;
    }
    for (int i = 0; i < n; ++i) {
      base[i] = i;
      pr[i] = -1;
      used[i] = 0;
    }

    int r = 0;
    st[r++] = root;
    used[root] = 1;
    for (int l = 0; l < r && match[root] == -1; ++l) {
      int v = st[l];

      for (int u : es[v]) {
        if (base[v] == base[u] || match[v] == u) {
          continue;
        }
        if (u == root || (match[u] != -1 && pr[match[u]]
!= -1)) {
          ++maxw;
          for (int b = v;; b = pr[b]) {
            b = base[b];
            was[b] = maxw;
            b = match[b];
            if (b == -1) {
              break;
            }
          }
          int b;
          for (b = u;; b = pr[match[b]]) {
            b = base[b];
            if (was[b] == maxw) {
              break;
            }
          }

          for (int i = 0; i < 2; ++i, swap(v, u)) {
            for (int x = v, y = u; base[x] != b; x =
pr[y]) {
              pr[x] = y;
              y = match[x];
              base[x] = base[y] = b;
              if (!used[y]) {
                used[y] = 1;
                st[r++] = y;
              }
            }
          }
          continue;
        }
        if (pr[u] != -1) {
          continue;
        }
        pr[u] = v;
        if (match[u] == -1) {
          while (u != -1) {
            swap(u, match[match[u] = pr[u]]);
          }
          break;
        }

        u = match[u];
        st[r++] = u;
        used[u] = 1;
      }
    }
  }
}
```

***dinic-gomory

```
namespace Flow {
```

```
const static long long inff = (long long) 1e18;
struct Edge {
  int t;
  long long f, c;

  Edge() {}
  Edge(int _t, long long _f, long long _c) : t(_t), f(_f), c(_c) {}
};

struct Graph {
  vector<Edge> edges;

  int n;
  vector<vector<int> > es;

  Graph(int _n = 0, int _m = 0) : n(_n), es(_n) {
    edges.reserve(2 * _m);
    edges.clear();

    st.reserve(n);
    dist.reserve(n);
    its.reserve(n);
  }

  void adde(int s, int t, long long c, long long c2 = 0) {
    assert(0 <= s && s < n);
    assert(0 <= t && t < n);
    //eprintf("%d -> %d (%lld,%lld)\n", s, t, c, c2);
    es[s].push_back(sz(edges));
    edges.push_back(Edge(t, 0, c));
    es[t].push_back(sz(edges));
    edges.push_back(Edge(s, 0, c2));
  }

  vector<int> st, its;
  vector<long long> dist;

  long long dfs(int v, int T, long long maxPush) {
    if (v == T) {
      return maxPush;
    }
    long long res = 0;
    for (int &it = its[v]; it < sz(es[v]); ++it) {
      int eid = es[v][it];
      Edge &e = edges[eid];
      if (e.c == e.f) {
        continue;
      }
      int u = e.t;
      if (dist[u] != dist[v] + 1) {
        continue;
      }
      long long push = dfs(u, T, min(maxPush, e.c - e.f));
      if (push) {
        res += push;
        maxPush -= push;
        e.f += push;
        edges[eid ^ 1].f -= push;
        if (maxPush == 0) {
          break;
        }
      }
    }
    return res;
  }

  void reset() {
    for (auto &e : edges) {
      e.f = 0;
    }
  }

  long long dinic(int S, int T) {
    its.resize(n);
    dist.resize(n);

    long long res = 0;
    while (1) {
      for (int i = 0; i < n; ++i) {
        dist[i] = inff;
      }
      dist[S] = 0;
      st.clear();
      st.push_back(S);
      for (int l = 0; l < sz(st); ++l) {
        int v = st[l];
        for (int eid : es[v]) {
          Edge &e = edges[eid];
          if (e.f == e.c) {
            continue;
          }
          int u = e.t;
          if (dist[u] <= dist[v] + 1) {
            continue;
          }
          dist[u] = dist[v] + 1;
          st.push_back(u);
        }
      }

      if (dist[T] == inff) {
        break;
      }

      for (int i = 0; i < n; ++i) {
        its[i] = 0;
      }
      while (1) {
        long long push = dfs(S, T, inff);
        if (!push) {
          break;
        }
        res += push;
      }
    }
    return res;
  }

  vector<vector<long long> > prec;

  void buildTree() {
    vector<int> p(n, 0);
    prec = vector<vector<long long> >(n, vector<long long>(n, inff));
    for (int i = 1; i < n; i++) {
      reset();
      long long f = dinic(i, p[i]);
      for (int j = 0; j < n; j++) {
        if (j != i && dist[j] < inff && p[j] == p[i]) {
          p[j] = i;
        }
      }
      prec[p[i]][i] = prec[i][p[i]] = f;
      for (int j = 0; j < i; j++) {
        prec[i][j] = prec[j][i] = min(prec[j][p[i]], f);
      }
      {
        int j = p[i];
```

```
                if (dist[p[j]] < inff) {
                    p[i] = p[j];
                    p[j] = i;
                }
            }
        }
    }

    long long fastFlow(int S, int T) {
        return prec[S][T];
    }
};
```

```
   ***lct
struct Node {
    Node *l, *r, *pr, *link;
    int linksum, sum;

    int val;
    pair<int, Node*> mx;

    int eid;
    bool rot;

    Node() {}
    Node(int _linksum, int _eid, int _val) : l(0), r(0),
pr(0), link(0), linksum(_linksum), val(_val), eid(_eid),
rot(0) {
        update();
    }

    void push() {
        if (rot) {
            rot = 0;
            swap(l, r);
            if (l) {
                l->rot ^= 1;
            }
            if (r) {
                r->rot ^= 1;
            }
        }
    }

    Node* update() {
        sum = linksum, mx = mp(val, this);
        if (l) {
            sum += l->sum;
            mx = max(mx, l->mx);
            l->pr = this;
        }
        if (r) {
            sum += r->sum;
            mx = max(mx, r->mx);
            r->pr = this;
        }
        return this;
    }
};
```

```
void rotate(Node *v) {
    Node *u = v->pr;
    if (u->l == v) {
        u->l = v->r;
        v->r = u;
    } else {
        u->r = v->l;
        v->l = u;
    }
    swap(v->link, u->link);
    swap(v->pr, u->pr);
    if (v->pr) {
        (v->pr->l == u ? v->pr->l : v->pr->r) = v;
    }
    u->update(), v->update();
}

void bigRotate(Node *v) {
    Node *u = v->pr;
    if (u->pr) {
        u->pr->push();
    }
    u->push(), v->push();
    if (u->pr) {
        if ((u->l == v) == (u->pr->l == u)) {
            rotate(u);
        } else {
            rotate(v);
        }
    }
    rotate(v);
}

void splay(Node *v) {
    while (v->pr) {
        bigRotate(v);
    }
}

void cutTail(Node *v) {
    splay(v);
    v->push();
    if (v->r) {
        v->r->pr = 0;
        v->r->link = v;
        v->linksum += v->r->sum;
        v->r = 0;
    }
    v->update();
}



void expose(Node *v0) {
    cutTail(v0);
    for (Node *v = v0; v->link;) {
        Node *u = v->link;
        u->linksum -= v->sum;
        v->link = 0;
        cutTail(u);
        u->r = v;
        v = u->update();
    }
    splay(v0);
}

void makeRoot(Node *v) {
    expose(v);
    v->rot ^= 1;
}

void link(Node *from, Node *to) {
    makeRoot(from), makeRoot(to);
    from->link = to;
    to->linksum += from->sum;
    to->update();
}

void cut(Node *from, Node *to) {
    makeRoot(to);
```

```
| expose(from);
| assert(from->l == to);
| to->pr = 0, from->l = 0;
| from->update();
}


   ***2dgeom
struct line {
| point p, v;

| // p + v * alf

| line() {}
| line(const point &p_, const point &v_) : p(p_), v(v_)
{}

| ld get(const point &x) const {
| | return v ^ (x - p);
| }

| // half: get > 0

| bool operator < (const line &l) const {
| | if (v.type() != l.v.type()) {
| | | return v.type() < l.v.type();
| | }
| | return (v ^ l.v) > (ld) 0.5;
| }
};

//intersect two lines, assume they do not coincide
bool intersect(const line &a, const line &b, point &res)
{
| ld d = (a.v ^ b.v);
| if (abs(d) < 0.5) { // eps!!!
| | return false;
| }
| res = a.p + a.v * (((b.p - a.p) ^ b.v) / d);
#ifdef DEBUG
| assert(abs(a.get(res)) < 1e-3 && abs(b.get(res)) <
1e-3);
#endif
| return true;
}


//a, b, c are on the same line, check if c is in [a,b]
bool inside(point a, point b, point c) {
| if (b < a) {
| | swap(a, b);
| }
| return !(c < a || b < c);
}

//intersect [a,b] and [c,d]
//assume they are not on the same line and have positive
length
bool intersect(point a, point b, point c, point d, point
&res) {
| if (!intersect(line(a, b), line(c, d), res)) {
| | return false;
| }
| return inside(a, b, res) && inside(c, d, res);
}

//intersect line and circle
bool intersect(line l, point o, ld r, point res[2]) {
| point u = l.v.norm();
| point v = u.ort();
| ld d = (l.p - o) * v;
| if (abs(d) > r + eps) {
| | return false;
| }

| ld x = sqrt(max((ld) 0, r * r - d * d));

| for (int it = 0; it < 2; ++it) {
| | res[it] = o + v * d + u * (it ? -x : x);
| }
| return true;
}

//intersect two circles
bool intersect(point o1, ld r1, point o2, ld r2, point
res[2]) {
| if (r1 < r2 - eps) {
| | swap(o1, o2);
| | swap(r1, r2);
| }
| ld d = (o2 - o1).len();
| if (r1 + r2 < d - eps || r1 - r2 > d + eps) {
| | return false;
| }

| ld cosa = (r1 * r1 + d * d - r2 * r2) / ((ld) 2 * r1 *
d);
| ld sina = sqrt(max((ld) 0, (ld) 1 - cosa * cosa));
| point v1 = (o2 - o1).norm() * r1;
| point v2 = v1.ort();

| for (int it = 0; it < 2; ++it) {
| | res[it] = o1 + v1 * cosa + v2 * (it ? -sina : sina);
| }
| return true;
}

int commonTangents(point o1, ld r1, point o2, ld r2,
point *res) {
| ld d = (o1 - o2).len();
| if (d < eps) {
| | return 0;
| }
| int cnt = 0;
| for (int i = 0; i < 2; ++i) {
| | ld cosa = (r1 + (i ? -r2 : r2)) / d;
| | if (abs(cosa) < (ld) 1 + eps) {
| | | ld sina = sqrt(max((ld) 0, (ld) 1 - cosa * cosa));
| | | point v1 = (o2 - o1).norm() * r1;
| | | point v2 = v1.ort();
| | | for (int it = 0; it < 2; ++it) {
| | | | res[cnt++] = o1 + v1 * cosa + v2 * (it ? -sina :
sina);
| | | }
| | }
| }
| return cnt;
}

namespace Halfplane {
| const static int maxn = (int) 1e5 + 10;
| line ls[maxn];
| int st[maxn * 2];
| point p[maxn];

| int intersect(int n, line *_ls, point *res) {
| | for (int i = 0; i < n; ++i) {
| | | ls[i] = _ls[i];
| | }

| | static point box[4] = {
| | | point(inf, -inf),
| | | point(inf, inf),
```

```
        point(-inf, inf),
        point(-inf, -inf)
    };
    for (int i = 0; i < 4; ++i) {
        ls[n++] = line(box[i], box[(i + 1) % 4] - box[i]);
    }

    sort(ls, ls + n);

    int n0 = n;
    n = 0;
    for (int iter = 0; iter < 2; ++iter) {
        for (int i = 0; i < n0; ++i) {
            bool fail = 0;
            while (n) {
                int p = st[n - 1];
                if (abs(ls[i].v ^ ls[p].v) < 0.5) {
                    if (ls[i].v.type() != ls[p].v.type()) {
                        return -1;
                    }
                    if (ls[p].get(ls[i].p) <= eps) {
                        fail = 1;
                        break;
                    }
                    --n;
                    continue;
                }

                if (n > 1) {
                    point q;
                    assert(intersect(ls[i], ls[p], q));
                    int pp = st[n - 2];
                    if (ls[pp].get(q) < -eps) {
                        --n;
                        continue;
                    }
                }
                break;
            }
            if (!fail) {
                st[n++] = i;
            }
        }
    }

    vector<int> when(n0, -1);
    bool ok = 0;
    for (int i = 0; i < n; ++i) {
        auto &cur = when[st[i]];
        if (cur == -1) {
            cur = i;
            continue;
        }
        int s = cur, t = i;
        rotate(st, st + s, st + t);
        n = t - s;
        ok = 1;
        break;
    }
    if (!ok) {
        return -1;
    }
    st[n] = st[0];
    for (int i = 0; i < n; i++) {
        assert(intersect(ls[st[i]], ls[st[i + 1]],
res[i]));
    }
    return n;
  }
};
```

```
struct SVG {
  FILE *out;
  ld sc = 50;

  void open() {
    out = fopen("image.svg", "w");
    fprintf(out, «svg xmlns='http://www.w3.org/2000/svg'
viewBox='-1000 -1000 2000 2000'>\n");
  }

  void line(point a, point b) {
    a = a * sc, b = b * sc;
    fprintf(out, «line x1='%f' y1='%f' x2='%f' y2='%f'
stroke='black'/>\n", a.x, -a.y, b.x, -b.y);
  }

  void circle(point a, ld r = -1, string col = "red") {
    r = sc * (r == -1 ? 0.3 : r);
    a = a * sc;
    fprintf(out, «circle cx='%f' cy='%f' r='%f'
fill='%s'/>\n", a.x, -a.y, r, col.c_str());
  }


  void text(point a, string s) {
    a = a * sc;
    fprintf(out, «text x='%f' y='%f'
font-size='100px'>%s</text>\n", a.x, -a.y, s.c_str());
  }

  void close() {
    fprintf(out, «/svg>\n");
    fclose(out);
    out = 0;
  }

   SVG() {
    if (out) {
      close();
    }
  }
} svg;
```

```
   ***delaunay
struct line {
  long long a, b, c;

  line(long long _a, long long _b, long long _c) : a(_a),
b(_b), c(_c) {}

  long long dist(const point &p) const {
    return a * p.x + b * p.y + c;
  }
};

inline bool intersectr(const line &a, const line &b, long
long &x, long long &y, long long &denom) {
  long long d = a.a * b.b - b.a * a.b;
  if (!d) {
    return 0;
  }
  denom = d;
  x = (a.b * b.c - a.c * b.b);
  y = (a.c * b.a - a.a * b.c);
  if (denom < 0) {
    denom = -denom, x = -x, y = -y;
  }
  return 1;
}

int sgn(long long x) {
```

```
| return (x < 0) ? -1 : !!x;
}

bool inside(const point &a, const point &b, const point
&p) {
| return !((p - a) ^ (b - a)) && ((p - a) * (p - b)) <=
0ll;
}

bool inside(const point &a, const point &b, const point
&c, const point &p) {
| //eprintf("%s %s %s %s\n", a.str(), b.str(), c.str(),
p.str());
| if (sgn((b - a) ^ (p - a)) * sgn((p - a) ^ (c - a)) ==
-1) {
| | return 0;
| }
| if (sgn((c - b) ^ (p - b)) * sgn((p - b) ^ (a - b)) ==
-1) {
| | return 0;
| }
| if (sgn((a - c) ^ (p - c)) * sgn((p - c) ^ (b - c)) ==
-1) {
| | return 0;
| }
| return 1;
}

int superComp(const long long &cenx, const long long
&ceny, const long long &denom, const point &a, const
point &b) {
| long long a1 = (cenx + cenx - (a.x + b.x) * denom);
| long long b1 = (ceny + ceny - (a.y + b.y) * denom);
| int a2 = (b.x - a.x);
| int b2 = (b.y - a.y);
| __int128 tmp = (__int128) a1 * a2 + (__int128) b1 * b2;
| return tmp < 0 ? -1 : !!tmp;
}

bool insideCircle(const point &a, const point &b, const
point &c, const point &p) {
| line sab(2 * (b.x - a.x), 2 * (b.y - a.y), 0);
| sab.c = -(sab.dist(b) + sab.dist(a)) / 2;
| line sac(2 * (c.x - a.x), 2 * (c.y - a.y), 0);
| sac.c = -(sac.dist(c) + sac.dist(a)) / 2;

| long long cenx, ceny, denom;
| assert(intersectr(sab, sac, cenx, ceny, denom));
#ifdef DEBUG
| assert(!superComp(cenx, ceny, denom, a, b));
| assert(!superComp(cenx, ceny, denom, a, c));
#endif
| return superComp(cenx, ceny, denom, p, a) == -1;
}



struct Vertex {
| Vertex *go[3];
| int deg;

| point ps[3];
| Vertex *op[3];
| int opid[3];
|
| Vertex() {}
| Vertex(const point &a, const point &b, const point &c)
: deg(0) {
| | ps[0] = a, ps[1] = b, ps[2] = c;
| | memset(op, 0, sizeof(op));
| | memset(opid, -1, sizeof(opid));
| | val = -1;
| }

| int val;

| inline void* operator new(size_t cnt) ;
};

const int maxv = maxn * 10;
Vertex _vs[maxv];
int cntv;

inline void* Vertex::operator new(size_t cnt) {
| assert(cntv < maxv);
| void* res = _vs + cntv;
| ++cntv;
| return res;
}

Vertex* findPoint(Vertex *v, const point &p) {
| while (v->deg) {
| | assert(inside(v->ps[0], v->ps[1], v->ps[2], p));
| | bool found = 0;
| | for (int i = 0; i < v->deg; ++i) {
| | | Vertex *u = v->go[i];
| | | if (inside(u->ps[0], u->ps[1], u->ps[2], p)) {
| | | | found = 1;
| | | | v = u;
| | | | break;
| | | }
| | }
| | assert(found);
| }
| return v;
}

void addEdge(Vertex *v, int vi, Vertex *u, int ui) {
| if (v) {
| | v->op[vi] = u;
| | v->opid[vi] = ui;
| | assert(v->deg <= 3);
| }
| if (u) {
| | u->op[ui] = v;
| | u->opid[ui] = vi;
| | assert(u->deg <= 3);
| }
}

void flip(Vertex *v, int vi) {
| Vertex *u = v->op[vi];
| int ui = v->opid[vi];
| if (!u) {
| | return;
| }
| if (!insideCircle(v->ps[0], v->ps[1], v->ps[2],
u->ps[ui])) {
| | return;
| }
| Vertex* nv1 = new Vertex(v->ps[vi], v->ps[(vi + 1) %
3], u->ps[ui]);
| Vertex* nv2 = new Vertex(v->ps[vi], u->ps[ui],
v->ps[(vi + 2) % 3]);

| for (int iter = 0; iter < 2; ++iter) {
| | Vertex *from = !iter ? v : u;
| | assert(!from->deg);
| | from->deg = 2;
| | from->go[0] = nv1, from->go[1] = nv2;
| }
```

```
| addEdge(nv1, 0, u->op[(ui + 1) % 3], u->opid[(ui + 1) %
3]);
| addEdge(nv2, 0, u->op[(ui + 2) % 3], u->opid[(ui + 2) %
3]);

| addEdge(nv1, 1, nv2, 2);

| addEdge(nv1, 2, v->op[(vi + 2) % 3], v->opid[(vi + 2) %
3]);
| addEdge(nv2, 1, v->op[(vi + 1) % 3], v->opid[(vi + 1) %
3]);

| flip(nv1, 0);
| flip(nv2, 0);
}

int perm[maxn];

const int maxc = (int) 2e5;
Vertex* buildDelaunay(int n, point ps[maxn]) {
| for (int i = 0; i < n; ++i) {
| | assert(abs(ps[i].x) < maxc && abs(ps[i].y) < maxc);
| }
| Vertex* root = new Vertex(point(-maxc, -maxc),
point(maxc * 3, -maxc), point(-maxc, maxc * 3));
| for (int i = 0; i < n; ++i) {
| | perm[i] = i;
| }
| random_shuffle(perm, perm + n, rnd);
| for (int iterv = 0; iterv < n; ++iterv) {
| | int v = iterv;
| | v = perm[iterv];

| | Vertex *leaf = findPoint(root, ps[v]);
| | assert(inside(leaf->ps[0], leaf->ps[1], leaf->ps[2],
ps[v]));

| | bool onEdge = 0;
| | for (int i = 0; i < 3; ++i) {
| | | point &p1 = leaf->ps[(i + 1) % 3];
| | | point &p2 = leaf->ps[(i + 2) % 3];
| | | if (inside(p1, p2, ps[v])) {
| | | | onEdge = 1;

| | | | Vertex* vs[2] = {leaf, leaf->op[i]};
| | | | int is[2] = {i, leaf->opid[i]};
| | | | assert(vs[1]);
| | | | Vertex *nvs[2][2];
| | | | for (int iter = 0; iter < 2; ++iter) {
| | | | | nvs[iter][0] = new Vertex(ps[v],
vs[iter]->ps[is[iter]], vs[iter]->ps[(is[iter] + 1) %
3]);
| | | | | nvs[iter][1] = new Vertex(ps[v],
vs[iter]->ps[(is[iter] + 2) % 3],
vs[iter]->ps[is[iter]]);
| | | | | addEdge(nvs[iter][0], 2, nvs[iter][1], 1);
| | | | | addEdge(nvs[iter][0], 0, vs[iter]->op[(is[iter]
+ 2) % 3], vs[iter]->opid[(is[iter] + 2) % 3]);
| | | | | addEdge(nvs[iter][1], 0, vs[iter]->op[(is[iter]
+ 1) % 3], vs[iter]->opid[(is[iter] + 1) % 3]);

| | | | | vs[iter]->deg = 2;
| | | | | for (int i = 0; i < 2; ++i) {
| | | | | | vs[iter]->go[i] = nvs[iter][i];
| | | | | }
| | | | }
| | | | addEdge(nvs[0][0], 1, nvs[1][1], 2);
| | | | addEdge(nvs[0][1], 2, nvs[1][0], 1);

| | | | for (int iter = 0; iter < 2; ++iter) {
| | | | | for (int i = 0; i < 2; ++i) {
| | | | | | flip(nvs[iter][i], 0);
| | | | | }
| | | | }
| | | | break;
| | | }
| | }
| | if (!onEdge) {
| | | leaf->deg = 3;

| | | Vertex *nvs[3];
| | | for (int i = 0; i < 3; ++i) {
| | | | nvs[i] = new Vertex(ps[v], leaf->ps[(i + 1) % 3],
leaf->ps[(i + 2) % 3]);
| | | | addEdge(nvs[i], 0, leaf->op[i], leaf->opid[i]);
| | | | leaf->go[i] = nvs[i];
| | | }
| | | for (int i = 0; i < 3; ++i) {
| | | | addEdge(nvs[i], 1, nvs[(i + 1) % 3], 2);
| | | }

| | | for (int i = 0; i < 3; ++i) {
| | | | flip(nvs[i], 0);
| | | }
| | }
| }
| return root;
}


   ***aho-corasick
const int maxc = 10;

struct Vert {
| int go[maxc];

| int pr;
| int prc;

| int suf;
| bool end;
|
| Vert(int _pr = -1, int _prc = -1) : pr(_pr), prc(_prc),
suf(-1), end(0) {
| | memset(go, -1, sizeof(go));
| }
};

const int maxd = 50 + 5;
const int maxn = (int) 1e3 + 10, maxv = maxn * (maxd +
1);

Vert vs[maxv];

int cntv, root;
void addStr(int m, char *s) {
| int v = root;
| for (int i = 0; i < m; ++i) {
| | int c = s[i] - '0';
| | if (vs[v].go[c] != -1) {
| | | v = vs[v].go[c];
| | } else {
| | | vs[cntv] = Vert(v, c);
| | | vs[v].go[c] = cntv;
| | | v = cntv++;
| | }
| }
| vs[v].end = 1;
}

int countSuf(int v) ;

int countGo(int v, int c) {
```

```
| int &res = vs[v].go[c];
| if (res != -1) {
| | return res;
| }

| int u = countSuf(v);
| if (v != u) {
| | res = countGo(u, c);
| } else {
| | res = root;
| }

| return res;
}

int countSuf(int v) {
| int &suf = vs[v].suf;
| if (suf != -1) {
| | return suf;
| }

| int c = vs[v].prc;
| int u = vs[v].pr;
| if (u != root) {
| | suf = countGo(countSuf(u), c);
| } else {
| | suf = root;
| }
| return suf;
}

void solve() {
| int slen = (int) strlen(s);
| cntv = 1;
| root = 0;
| vs[root] = Vert();

| int d = (int) strlen(x);
| for (int i = 0; i <= slen - (d / 2); ++i) {
| | addStr(d / 2, s + i);
| }

| vs[root].suf = root;
| for (int v = 1; v < cntv; ++v) {
| | countSuf(v);
| | if (vs[vs[v].suf].end) {
| | | vs[v].end = 1;
| | }
| | for (int c = 0; c < maxc; ++c) {
| | | countGo(v, c);
| | }
| }
```

***pal

```
struct Node {
| map<char, Node*> go;
| int len;

| Node *suf;

| Node() {}
| Node(int _len) : len(_len), suf(0) {
| | go.clear();
| }

| inline void* operator new(size_t cnt) ;
};

const int maxtmp = (int) 1.1e6;
Node tmp[maxtmp];
int tmppos = 0;

inline void* Node::operator new(size_t cnt) {
| assert(tmppos < maxtmp);
| void* res = tmp + tmppos++;
| return res;
}
// tmppos := 0

struct PalTree {
| Node *minus, *zero;

| PalTree() {
| | minus = new Node(-1);
| | zero = new Node(0);
| | zero->suf = minus;
| }

| void addStr(char *s) {
| | Node* last = zero;
| | for (int i = 0; s[i]; ++i) {
| | | while (1) {
| | | | assert(last);
| | | | int left = i - 1 - last->len;
| | | | if (left < 0 || s[left] != s[i]) {
| | | | | last = last->suf;
| | | | | continue;
| | | | }
| | | | {
| | | | | auto iter = last->go.find(s[i]);
| | | | | if (iter != last->go.end()) {
| | | | | | last = iter->second;
| | | | | | break;
| | | | | }
| | | | }
| | | | Node *prev = new Node(last->len + 2);
| | | | swap(prev, last);
| | | | prev->go[s[i]] = last;

| | | | last->suf = zero;
| | | | for (prev = prev->suf; prev; prev = prev->suf) {
| | | | | if (s[i] == s[i - 1 - prev->len]) {
| | | | | | auto iter = prev->go.find(s[i]);
| | | | | | if (iter != prev->go.end()) {
| | | | | | | last->suf = iter->second;
| | | | | | | break;
| | | | | | }
| | | | | }
| | | | }
| | | | break;
| | | }
| | }
| }
};
```

***sufauto

```
const int maxn = (int) 1e5 + 10;
const int maxc = 26;

struct Node {
| Node *go[maxc];
| Node *suf;
| int len;

| Node(int _len = 0) : suf(0), len(_len) {
| | memset(go, 0, sizeof(go));
| }

| inline void* operator new(size_t sz) ;
};

const int maxv = (int) maxn * 2;
```

```
Node tmp[maxv];
int cntv;

inline void* Node::operator new(size_t sz) {
  assert(cntv < maxv);
  Node* res = tmp + cntv;
  ++cntv;
  return res;
}

Node *root, *last;

void precalc() {
  cntv = 0;
  root = last = new Node();
}

void addChar(int ch) {
  Node *v = new Node(last->len + 1);
  swap(v, last);
  assert(v != last);

  for (; v && !v->go[ch]; v = v->suf) {
    v->go[ch] = last;
  }

  if (!v) {
    last->suf = root;
  } else {
    Node *u = v->go[ch];
    if (v->len + 1 == u->len) {
      last->suf = u;
    } else {
      Node *clone = new Node(v->len + 1);
      memcpy(clone->go, u->go, sizeof(u->go));
      clone->suf = u->suf, u->suf = last->suf = clone;
      for (; v && v->go[ch] == u; v = v->suf) {
        v->go[ch] = clone;
      }
    }
  }
}
```

***ukkonen

```
struct Node {
  map<char, Node*> go;
  int l, r;
  Node *pr, *suf;
  int len;

  Node() : l(0), r(0), pr(0), suf(this), len(0) {}
  Node(int _l, int _r, Node *_pr) : l(_l), r(_r),
pr(_pr), suf(0) {
    pr->go[s[l]] = this;
    len = pr->len + r - l;
  }
};

struct Pos {
  Node *v;
  int up;

  Pos() {}
  Pos(Node *_v, int _up) : v(_v), up(_up) {}

  Pos go(char ch) const {
    if (up) {
      if (ch == s[v->r - up]) {
        return Pos(v, up - 1);
      }
      return Pos(0, 0);
    }
    auto iter = v->go.find(ch);
    if (iter == v->go.end()) {
      return Pos(0, 0);
    }
    return Pos(iter->second, iter->second->r -
iter->second->l - 1);
  }

  Node* splitEdge() const {
    if (!up) {
      return v;
    }
    Node *mid = new Node(v->l, v->r - up, v->pr);
    v->pr = mid, mid->go[s[mid->r]] = v, v->l = mid->r;
    return mid;
  }
};

struct SufTree {
  Node *root;
  Pos last;

  SufTree() {
    root = new Node();
    last = Pos(root, 0);
  }

  void addCh(int pos) {
    while (1) {
      Pos nlast = last.go(s[pos]);
      if (nlast.v) {
        last = nlast;
        return;
      }

      Node *v = last.splitEdge();
      new Node(pos, maxn, v);

      Node *&suf = v->suf;
      if (!suf) {
        suf = v->pr->suf;
        int left = v->l + !(v->pr->len);
        while (left < v->r) {
          suf = suf->go[s[left]];
          left += suf->r - suf->l;
        }
        suf = Pos(suf, left - v->r).splitEdge();
      }
      last = Pos(suf, 0);
      if (!v->len) {
        return;
      }
    }
  }
} S;
```

***easy

```
void prefix() {
  pr[0] = -1;
  for (int i = 0; i < s[i]; i++) {
    int &x = pr[i + 1];
    for (x = pr[i]; x >= 0 && s[i] != s[x]; x = pr[x]) ;
    ++x;
  }
}

char t[maxn * 2];

void duval() {
  for (int i = 0; i < 2 * n; i++) {
```

```
    t[i] = s[i % n];
  }
  int res = 0;
  for (int i = 0; i < 2 * n;) {
    int l = i, r = i + 1;
    for (; r < 2 * n && t[r] >= t[l]; r++) {
      if (t[r] > t[l]) {
        l = i;
      } else {
        l++;
      }
    }
    for (; i <= l; i += r - l) {
      if (i < n) {
        res = i;
      }
    }
  }
  t[res + n] = 0;
  printf("%s\n", t + res);
}
```

```
   ***simplex
namespace Simplex {
  const int maxn = ::maxn;

  const ld inf = 1e18;
  int ns[maxn], ms[maxn];

  void pivot(int m, int n, ld a[maxn][maxn], int y, int
x) {
    {
      ld coef = a[y][x];
      a[y][x] /= coef;
      for (int j = 0; j <= n; ++j) {
        a[y][j] /= coef;
      }
    }
    for (int i = 0; i <= m; ++i) {
      if (i == y) {
        continue;
      }
      ld coef = a[i][x];
      a[i][x] = 0;
      for (int j = 0; j <= n; ++j) {
        a[i][j] -= a[y][j] * coef;
      }
    }
    swap(ns[x], ms[y]);
  }

  ld simplex(int m, int n, ld a[maxn][maxn], ld *ans = 0)
{
    /*eprintf("m = %d, n = %d\n", m, n);
    for (int i = 0; i <= m; ++i) {
      if (!i) {
        eprintf("func = ");
      } else {
        eprintf("ys{%d}: ", ms[i]);
      }
      for (int j = 0; j <= n; ++j) {
        eprintf("%.3f*x{%d}%c", (double) a[i][j], (j ?
ns[j] : -1), "\n"[j == n]);
      }
    }*/
    while (1) {
      int x = 0;
      for (int j = 1; j <= n; ++j) {
        if (a[0][j] > eps) {
          x = j;
          break;
        }
      }
      if (!x) {
        break;
      }
      int y = 0;
      ld del = inf;
      for (int i = 1; i <= m; ++i) {
        if (a[i][x] > eps) {
          ld val = -a[i][0] / a[i][x]; // minus!!!
          if (del > val) {
            del = val;
            y = i;
          }
        }
      }
      if (!y) {
        return inf;
      }
      pivot(m, n, a, y, x);
    }
    if (ans) {
      for (int i = 1; i <= n; ++i) {
        if (ns[i] <= n) {
          ans[ns[i] - 1] = 0;
        }
      }
      for (int i = 1; i <= m; ++i) {
        if (ms[i] <= n) {
          ans[ms[i] - 1] = -a[i][0];
        }
      }
    }
    return a[0][0];
  }

  ld c0[maxn];
  bool init(int m, int n, ld a[maxn][maxn]) {
    for (int i = 1; i <= n; ++i) {
      ns[i] = i;
    }
    for (int i = 1; i <= m; ++i) {
      ms[i] = i + n;
    }
    int mxpos = 1;
    for (int i = 1; i <= m; ++i) {
      if (a[mxpos][0] < a[i][0]) {
        mxpos = i;
      }
    }
    if (a[mxpos][0] <= eps) {
      return 1;
    }

    ns[n + 1] = 0;
    for (int i = 0; i <= m; ++i) {
      a[i][n + 1] = -1;
    }
    for (int i = 0; i <= n; ++i) {
      c0[i] = a[0][i];
      a[0][i] = 0;
    }

    pivot(m, n + 1, a, mxpos, n + 1);
    ld res = simplex(m, n + 1, a);
    if (res < -eps) {
      return 0;
    }
    assert(res <= eps);
    {
```

```
        int y = find(ms + 1, ms + m + 1, 0) - ms;
        if (y != m + 1) {
          int x;
          for (x = 1; x <= n + 1 && abs(a[y][x]) < eps;
++x) ;
          assert(x <= n + 1);
          pivot(m, n + 1, a, y, x);
        }
      }
      {
        int col = find(ns + 1, ns + n + 2, 0) - ns;
        assert(col <= n + 1);
        for (int i = 1; i <= m; ++i) {
          swap(a[i][col], a[i][n + 1]);
        }
        swap(ns[col], ns[n + 1]);
      }

      a[0][0] = c0[0];
      for (int i = 1; i <= n; ++i) {
        a[0][i] = 0;
      }

      for (int i = 1; i <= n; ++i) {
        if (ns[i] <= n) {
          a[0][i] += c0[ns[i]];
        }
      }
      for (int i = 1; i <= m; ++i) {
        if (ms[i] <= n) {
          ld coef = c0[ms[i]];
          for (int j = 0; j <= n; ++j) {
            a[0][j] -= coef * a[i][j];
          }
        }
      }
      return 1;
    }

    /*
    |* m equations, n variables,
    |* x >= 0
    |* [v0 + cx] -> max
    |* [b + Ax] <= 0
    |* a = Matrix[m + 1][n + 1]
    |* */
    ld solve(int m, int n, ld a[maxn][maxn], ld *ans) {
      if (!init(m, n, a)) {
        return -inf;
      }
      return simplex(m, n, a, ans);
    }
};


   ***treap
typedef pair<int, Hash> H;

H concat(const H &a, const H &b) {
    return H(a.first + b.first, a.second * qs[b.first] +
b.second);
}

struct Node {
    Node *l, *r;
    int cnt;

    int val;
    H h;

    Node *pr;

    Node(int _val = 0) : l(0), r(0), val(_val) {
      update();
    }

    Node* update() {
      pr = 0;
      h = H(0, 0);
      cnt = 1;
      if (l) {
        l->pr = this;
        h = l->h;
        cnt += l->cnt;
      }
      h = concat(h, mp(1, val));
      if (r) {
        r->pr = this;
        h = concat(h, r->h);
        cnt += r->cnt;
      }
      return this;
    }
};

Node *merge(Node *l, Node *r) {
    if (!l && !r) {
      return 0;
    }
    if (!l) {
      return r->update();
    }
    if (!r) {
      return l->update();
    }

    if (rnd(l->cnt + r->cnt) < l->cnt) {
      l->r = merge(l->r, r);
      return l->update();
    } else {
      r->l = merge(l, r->l);
      return r->update();
    }
}

void split(Node *v, int key, Node *&left, Node *&right) {
    if (!v) {
      assert(!key);
      left = right = 0;
      return;
    }
    int notmore = (v->l ? v->l->cnt : 0) + 1;
    if (notmore > key) {
      split(v->l, key, left, v->l);
      right = v->update();
    } else {
      split(v->r, key - notmore, v->r, right);
      left = v->update();
    }
}

int getId(Node *v) {
    assert(v);

    int res = -1;
    for (Node *last = v->r; v; last = v, v = v->pr) {
      if (v->r == last) {
        res += (v->l ? v->l->cnt : 0) + 1;
      }
    }
    assert(res >= 0);
    return res;
}
```

```
   ***twochinese
const int maxn = 1e3 + 10;
const int maxe = 1e4 + 10;

int n, m;

int a[maxe];
int b[maxe];
int l[maxe];

struct mycomp {
| bool operator()(const int & e, const int & f) {
| | if (l[e] != l[f]) {
| | | return l[e] < l[f];
| | }
| | return e < f;
| }
};

struct myset {
| set<int, mycomp> *s;
| int d;

| myset() {
| | s = new set<int, mycomp>;
| | d = 0;
| }

| void add(int e) {
| | l[e] -= d;
| | s->insert(e);
| }

| int min() {
| | assert(sz(*s));//assume answer exists
| | int e = *(s->begin());
| | s->erase(s->begin());
| | l[e] += d;
| | return e;
| }

| void merge(myset & a) {
| | if (sz(*s) < sz(*a.s)) {
| | | swap(s, a.s);
| | | swap(d, a.d);
| | }
| | for (auto e : (*a.s)) {
| | | l[e] += a.d;
| | | add(e);
| | }
| }
};

bitset<maxn> in[maxn];
int pr[maxn];
myset c[maxn];

bool read() {
| if (scanf("%d%d", &n, &m) < 2) {
| | return false;
| }
| for (int i = 0; i < n; ++i) {
| | c[i] = myset();
| | in[i].reset();
| | in[i][i] = 1;
| }
| for (int i = 0; i < m; ++i) {
| | scanf("%d%d%d", &a[i], &b[i], &l[i]);
| | --a[i], --b[i];
| | c[b[i]].add(i);
| }
| return true;
}

int used[maxn];
set<int> ans;

vector<int> st;

void go(int v) {
| if (used[v] == 2) {
| | return;
| }
| int e, u;
| do {
| | e = c[v].min();
| | u = a[e];
| } while (in[v][u]);
| c[v].d -= l[e];
| ans.insert(e);
| pr[v] = e;
| st.pb(v);
| used[v] = 1;
| if (used[u] == 1) {
| | int x = n++;
| | vector<int> cycle;
| | c[x] = myset();
| | used[x] = 0;
| | in[x].reset();
| | while (true) {
| | | int w = st.back();
| | | c[x].merge(c[w]);
| | | in[x] |= in[w];
| | | cycle.pb(w);
| | | st.pop_back();
| | | if (in[w][u]) {
| | | | break;
| | | }
| | }
| | go(x);
| | --n;
| | int w = b[pr[x]];
| | for (int i = 0; i < sz(cycle); ++i) {
| | | int y = cycle[i];
| | | if (in[y][w]) {
| | | | ans.erase(pr[y]);
| | | | pr[y] = pr[x];
| | | }
| | | used[y] = 2;
| | }
| } else {
| | go(u);
| | if (st.back() == v) {
| | | used[v] = 2;
| | | st.pop_back();
| | }
| }
}

void solve() {
| memset(used, 0, sizeof(used));
| ans.clear();
| st.clear();
| used[0] = 2;//root
| for (int i = 1; i < n; ++i) {
| | if (!used[i]) {
| | | go(i);
| | }
| }

| assert(sz(ans) == n - 1);
```

```
    printf("%d\n", sz(ans));
    for (auto a : ans) {
        printf("%d ", a + 1);
    }
    printf("\n");
}
```