Monkeys LIB
Cerrado Edition

Monkeys

UFG
UNIVERSIDADE
FEDERAL DE GOIÁS

2024

Sumário

Capítulo I

Begin

1.1 Who did this

Sometimes saying thank you is important, this book have the name and the heart of a group, full of disagreements. We are **different** and that is what makes us **strong**. Believe in yourself, and when u can't, believe in the people which are in your side.

Thank you.

Here is a list of names which contributed to build this book (for those who are asking, it was sorted by random_shuffle in C++20 using srand(252)):

- Lesin
- Alunea
- Nopebi Lifesa
- Atak Kichan
- Faslecar
- Nollyad
- Laelovatsug
- Ognol Tohberum
- Nhotivew

1.2 tips

- Remember that binary lifting isn't just for trees.
- Expected value? contribution is the way.
- DP? maybe a slow solution can be optimized.
- Chill, the contest is long, starting slow is good...
- Breath, drink water, make jokes, eat chocolate, at the end, have fun with your friends.
- A Greedy is a risk, but can be done. No one knows how to proof this shit anyway...
- flow? is this bipartite?
- remember what u can do in bipartite graphs:
  - minimum vertex cover = maximum matching
  - clique maximum(same as previous one)
  - maximum independent set = complement of minimum vertex cover
- Remember, a binary search can simplify a lot!
- Can you model any recurrence?
- Write alot, Think out, Text.
- Don't be fixed in finding a fast solution, find one solution, and then, try to understand it.
- are there modulus? maybe splitting in cases can help.
- is this monotonic(increasing or decreasing)?

- Geometry has 4 options: Line sweep, binary search, Convex hull and Math.
- Game theory? just brute. recurrence in big num? just brute. Overall bruting is gud.
- BREATH!
- Your friends are here to help!!!
- Hug each other, spread love, not war.
- A funny joke doesn't have to offend a friend.
- YOU ARE A TEAM!!!
- idk maybe guessing that only checking primes is enough(if it doesn't work, try with powers of two)
- if u can't do $n \log$, just do $n \log^2$ u bitch
- remember, you can do some factorial stuff using convolutions.

1.3 template - makefile - terminaltricks

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

//template
#pragma region

#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;
template<class T> using ordset = tree<T,null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;

// #pragma GCC optimize("Ofast") // for fast N^2
// #pragma GCC target("avx2") // for fast N^2
// #pragma GCC target("popcnt") // for fast popcount

#define all(x) x.begin(), x.end()
#define lef(x) (x << 1)
#define rig(x) (lef(x) | 1)

using ll = long long int;
using ld = long double;
using pll = pair<ll,ll>;
using pii = pair<int,int>;
using pdb = pair<ld,ld>;

//read and print pair
template<typename T, typename T1>
ostream & operator<<(ostream &os, pair<T, T1> p){
    os << "(" << p.ft << "," << p.sd << ")";
    return os;
}
template<typename T, typename T1>
istream & operator>>(istream &is, pair<T, T1>& p){
    is >> p.ft >> p.sd;
    return is;
}

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
#pragma endregion

void test() {

}

int32_t main() {
    ios::sync_with_stdio(false), cin.tie(nullptr);
    int ttt_ = 1;
    // cin >> ttt_;
    while(ttt_--) test();
    return 0;
}
```

```
// flags for speed. Make the code run faster and don't debug things.
    Just using some of the debugging flags so you know that something is
    exploding.

CXX = g++
CXXFLAGS = -fsanitize=address -fno-omit-frame-pointer -g -O2 -Wall -
    Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare

// Flags for debug. Use it in case something goes very wrong. They are
    separated because those below makes the compiler and the program go
```

```
    absurdly slow. Use it to catch mistakes like segmentation fault,
    division by zero, and many other things. It's not that much flags,
    but there is no need to make the compiler and the code run very slow
    everytime.

CXX = g++
CXXFLAGS = -fsanitize=address -fno-omit-frame-pointer -g -Wall -Wshadow
    -std=c++17 -Wno-unused-result -Wno-sign-compare -Wno-char-subscripts
    -fsanitize=undefined -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
```

```
command time -v ./out //saber a alocacao e mais uma porrada de coisas
    roubadas
ulimit -s unlimited //remover o limite da stack
```

1.4 math formulas

- $a_n = a_{n-1} + r$
- $a_n = a_1 + (n-1) * r$
- $soma_{(l,r)} = \frac{(a_l + a_r) * (r - l + 1)}{2}$
- $a_n = a_{n-1} * q$
- $a_n = a_1 * q^{n-1}$
- $soma_{(l,r)} = \frac{a_l * (q^{(r-l+1)} - 1)}{q-1}$

Fórmula de Heron para Triângulos: $\sqrt{p \cdot (p-b) \cdot (p-c) \cdot (p-d)}$
Fórmula de Heron para Quadriláteros:

- $p = (a + b + c + d)/2$
- $A = \sqrt{(p-a) \cdot (p-b) \cdot (p-c) \cdot (p-d)}$
- $A \rightarrow$ Maior área formada por 4 lados de um quadrilátero.

Lagrange Multipliers:
We want to optimize a function $f(x_1, x_2, \ldots, x_n)$ subject to the constraints $g_i(x_1, x_2, \ldots, x_n) = k_i$ where $1 \leq i \leq m$ and m is the number of constraints, to solve it, we use the Method of Lagrange Multipliers. It consists in solving the following system of equations:

$$\begin{cases} \nabla f(x_1, x_2, \ldots, x_n) & = & \sum_{i=1}^{m} \lambda_i \nabla g_i(x_1, x_2, \ldots, x_n) \\ g_i(x_1, x_2, \ldots, x_n) & = & k_i \qquad \text{for } 1 \leq i \leq m \end{cases}$$

For the first equation, you derivate for each variable and solve for each derivation.

$$\frac{a}{sen\hat{A}} = \frac{b}{sen\hat{B}} = \frac{c}{sen\hat{C}} = 2r$$

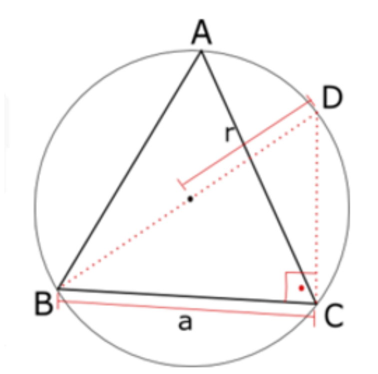


Figura I.1: Lei dos senos

$$a^2 = b^2 + c^2 - 2b \cdot c \cdot \cos \hat{A}$$
$$b^2 = a^2 + c^2 - 2a \cdot c \cdot \cos \hat{B}$$
$$c^2 = a^2 + b^2 - 2a \cdot b \cdot \cos \hat{C}$$

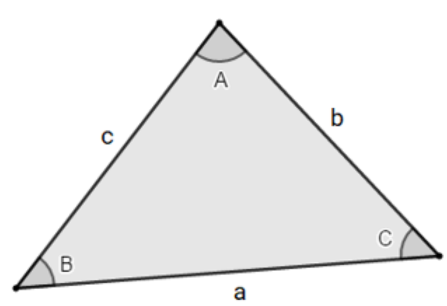


Figura I.2: Lei dos cossenos

1.5 Stress Test

During the contest, sometimes u will need to stress test an solution. the steps are:

- Code a Brute Force(do as lazy as possible, focus in not spend time coding it)
- Code a generator (in the following lines there are some generators for vectors, trees and graphs)
- run the stress test.
- To use this thing, just run "bash s.sh"and see everything working!

The run part will have some bash function like this:

```
#!/bin/bash

make brute
make gen
make at # codigo que muda

while true; do
    ./gen 2 > in

    start=$(date +%s)
    ./at < in > out
    end=$(date +%s)

    echo "Elapsed Time in test $c: $(($end - $start)) seconds."
    ./brute < in > aout
    diff -B out aout > /dev/null || break
    echo "Passou no caso."
    cat out
done

echo "WA on the following case:"
cat in
echo "Your answer is:"
cat out
echo "Correct answer is:"
cat aout

// Para mais de uma solucao: mandar para um validador e deixar ele
   resolver tudo.

#!/bin/bash

make brute          # fazer o codigo forca bruta do problema
make gen            # fazer o gerador do problema
make solution       # a sua solucao do problema
make validator      # o validador da sua solucao

for((c = 0; c <= 500; c++)); do
    ./gen 2 > in
    start = $(date +%s)
    ./solution < in > out
    end = $(date +%s)
    echo "Elapsed Time in test $c: $(($end - $start)) seconds."
    ./brute < in > a_in
    ./validator < a_in > aout
    diff -B out aout > /dev/null || break
    echo "Passou no caso $c."
done

echo "WA on the following case:"
cat in
echo "Your answer is:"
cat aout
echo "The possible solutions are:"
cat out
```

1.5.1 Vector Generator

Simple vector generator (just for remembering). Has two parameters, the quantity of elements the the biggest element.

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

```

```cpp
int main(int argc, char *argv[]) {
    int n = stoi(argv[1]);
    int lim = stoi(argv[2]);
    cout << n << '\n';
    rep(i, 0, n)
        cout << (rng()%lim) << ' ';
    cout << '\n';
    return 0;
}
```

1.5.2 Tree Generator

The following code generates good trees. It has two parameter, the quantity of vertices and the id of the test(used for generation).

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

vector<pii> edges;
int n;

void random_tree()
{
    rep(i, 1, n)
        edges.eb(i,rng()%i);
}
void catter()
{
    int k = rng()%(n/2) + n/2;
    rep(i, 1, k)
        edges.eb(i-1,i);
    rep(i, k, n)
        edges.eb(rng()%(i),i);
}

void star()
{
    rep(i, 1, n)
        edges.eb(0,i);
}

void print_tree()
{
    vector<int> p;
    rep(i, 1, n+1)
        p.pb(i);
    shuffle(all(p),rng);
    shuffle(all(edges),rng);
    cout << n << '\n';
    for(auto [u,v]:edges)
        cout << p[u] << ' ' << p[v] << '\n';
}

int32_t main(int argc, char *argv[])
{
    n = stoi(argv[1]);
    int tp = stoi(argv[2]);
    if(!tp)
        star();
    else
    {
        if(tp%3 == 0)
            catter();
        else
            random_tree();
```

```
    }
    print_tree();
    return 0;
}
```

Capítulo II

Algebra

2.1 Inteiro Modular

Classe para lidar com problemas relacionados a operações modulares.

```
const int MOD = 1'000'000'007;

struct mi {
    int v;
    explicit operator int() const { return v; }
    mi() { v = 0; }
    mi(long long int _v) : v(_v % MOD + (_v % MOD < 0) * MOD) {}

    friend mi operator -(const mi & at){
        return mi(-at.v);
    }

    void operator +=(const mi & ot){
        v += ot.v;
        v -= (v >= MOD) * MOD;
    }
    void operator -=(const mi & ot) {
        v -= ot.v;
        v += (v < 0) * MOD;
    }
    void operator *=(const mi & ot){
        v = 1ll * v * ot.v % MOD;
    }
    void operator /=(const mi & ot){
        v = 1ll * v * inv(ot).v % MOD;
    }
    friend mi operator +(const mi & a, const mi & b){
        return mi(a.v + b.v);
    }
    friend mi operator -(const mi & a, const mi & b){
        return mi(a.v - b.v);
    }
    friend mi operator *(const mi & a, const mi & b){
        return mi(1ll * a.v * b.v);
    }
    friend mi operator /(const mi & a, const mi & b){
        return mi(1ll * a.v * inv(b).v);
    }

    friend mi fexp(mi a, long long int b){
        mi ans(1);
        while(b){
            if(b&1) ans = ans*a;
            a *= a;
            b >>= 1;
        }
        return ans;
    }
    friend mi inv(const mi & a){
```

```
        assert(a.v != 0);
        return fexp(a, MOD - 2);
    }

    friend ostream& operator <<(ostream & out, const mi & at){
        return out << at.v;
    }
    friend istream& operator >>(istream & in, mi & at){
        return in >> at.v;
    }
};
```

2.2 Matrix Power

2.2.1 Graph Paths I

Consider a directed graph that has n nodes and m edges. Your task is to count the number of paths from node 1 to node n with exactly k edges.

```

typedef long long ll;
const int N = 100;
const ll mod = 1000000007LL;
ll mat[N][N], resp[N][N], aux[N][N];
void mult(ll a[N][N], ll b[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++){
            aux[i][j] = 0;
            for(int k = 0; k < N; k++)
                aux[i][j] = (aux[i][j] + a[i][k] * b[k][j] % mod) % mod;
        }

    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            a[i][j] = aux[i][j];
}
void show(ll a[N][N], int n)
{
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++)
            cout << a[i][j] << ' ';
        cout << '\n';
    }
}
void fasexp(ll b)
{
    for(int i = 0; i < N; i++){
        resp[i][i] = 1;
    }
    while(b)
    {
        if(b&1)
            mult(resp,mat);
        mult(mat,mat);
        b >>= 1;
    }
}
int dp[N];

int main()
{
    int n, m;
    ll k;
    cin >> n >> m >> k;
    while(m --)
```

```
    {
        int u, v;
        cin >> u >> v;
        u--, v--;
        mat[u][v]++;
    }
    fasexp(k);
    // show(resp, n);
    cout << resp[0][n-1] << '\n';
    return 0;
}
```

2.2.2 Graph Paths II

Consider a directed weighted graph having n nodes and m edges. Your task is to calculate the minimum path length from node 1 to node n with exactly k edges.

```

typedef long long ll;
const int N = 100;
ll mat[N][N], resp[N][N], aux[N][N];
void mult(ll a[N][N], ll b[N][N])
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++){
            aux[i][j] = -1;
            for(int k = 0; k < N; k++){
                if(a[i][k] == -1 || b[k][j] == -1)
                    continue;
                if(aux[i][j] == -1)
                    aux[i][j] = a[i][k] + b[k][j];
                aux[i][j] = min(aux[i][j] , a[i][k] + b[k][j]);
            }
        }

    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            a[i][j] = aux[i][j];
}
void fasexp(ll b)
{
    while(b)
    {
        if(b&1)
            mult(resp,mat);
        mult(mat,mat);
        b >>= 1;
    }
}
int dp[N];

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n, m;
    ll k;
    cin >> n >> m >> k;
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            mat[i][j] = -1;
            resp[i][j] = -1;
        }
        resp[i][i] = 0;
    }
    while(m --)
```

```
    {
        int u, v, c;
        cin >> u >> v >> c;
        u--, v--;
        if(mat[u][v] == -1)
            mat[u][v] = c;
        mat[u][v] = min(mat[u][v], (ll)c);
    }
    fasexp(k);
    cout << resp[0][n-1] << '\n';
    return 0;
}
```

2.3 FFT

2.3.1 Simple FFT

Multiplicação de dois polinômios em tempo $O((n+m) \cdot \log(n+m))$ O polinômio $a + bx + cx^2 + dx^3$ é representado na forma de vetor como $[a, b, c, d]$. Para usar a função *multiply*, crie dois vectors $vector < int > a, b$; apos fazer $multiply(a, b)$; tem-se o vector polinômio com os coeficientes da multiplicação de a e b.

FFT é tudo sobre convolução, a multiplicacao de dois polinômios é uma convolução. Convolução pode ser feita com funções contínuas ou discretas, a definição para funções contínuas seria: Dada duas funções contínuas por partes em $[0, \infty]$ a convolução de f e g é definida pela integral

$$(f * g)(t) = \int_0^t f(r)g(t-r)dr$$

Tem convolução normal, convolução com modulo random e NTT.

```
struct FFT{
    typedef complex<double> C;
    typedef vector<double> vd;
    typedef vector<long long int> vl;
    typedef vector<int> vi;

    /*
        * Author: Ludo Pulles, chilli, Simon Lindholm
        * Date: 2019-01-09
        * License: CC0
        * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf (do
    read, it's excellent)
        Accuracy bound from http://www.daemonology.net/papers/fft.pdf
        * Description: fft(a) computes $\hat f(k) = \sum_x a[x] \exp(2\
   pi i \cdot k x / N)$ for all $k$. N must be a power of 2.
        Useful for convolution:
        \texttt{conv(a, b) = c}, where $c[x] = \sum a[i]b[x-i]$.
        For convolution of complex numbers or more than two vectors: FFT
   , multiply
        pointwise, divide by n, reverse(start+1, end), FFT back.
        Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2{N} < 9\
   cdot10^{14}$
        (in practice $10^{16}$; higher for random inputs).
        Otherwise, use NTT/FFTMod.
        * Time: O(N \log N) with $N = |A|+|B|$ ($\tilde 1s$ for $N
   =2^{22}$)
        * Status: somewhat tested
        * Details: An in-depth examination of precision for both FFT and
    FFTMod can be found
        * here (https://github.com/simonlindholm/fft-precision/blob/
   master/fft-precision.md)
    */
    void fft(vector<C>& a) {
        int n = a.size(), L = 31 - __builtin_clz(n);
        static vector<complex<long double>> R(2, 1);
        static vector<C> rt(2, 1);  // (^ 10% faster if double)
```

```
     for (static int k = 2; k < n; k *= 2) {
         R.resize(n); rt.resize(n);
         auto x = polar(1.0L, acos(-1.0L) / k);
         for(int i=k; i<2*k; i++) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
     }
     vi rev(n);
     for(int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
     for(int i = 0; i < n; i++)  if (i < rev[i]) swap(a[i], a[rev[i]]);
     for (int k = 1; k < n; k *= 2)
         for (int i = 0; i < n; i += 2 * k) for(int j = 0; j < k; j++) {
             // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)  /// include-line
             auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];     /// exclude-line
             C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);        /// exclude-line
             a[i + j + k] = a[i + j] - z;
             a[i + j] += z;
         }
 }
 // multiplica dois polinomios
 vd conv(const vd& a, const vd& b) {
     if (a.empty() || b.empty()) return {};
     vd res(a.size() + b.size() - 1);
     int L = 32 - __builtin_clz(res.size()), n = 1 << L;
     vector<C> in(n), out(n);
     copy(a.begin(),a.end(), in.begin());
     for(int i = 0; i < b.size(); i++) in[i].imag(b[i]);
     fft(in);
     for (C& x : in) x *= x;
     for(int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i]);
     fft(out);
     for(int i = 0; i < res.size(); i++) res[i] = imag(out[i]) / (4 * n);
     return res;
 }
 vl conv(const vl& a, const vl& b) {
     if (a.empty() || b.empty()) return {};
     vl res(a.size() + b.size() - 1);
     int L = 32 - __builtin_clz(res.size()), n = 1 << L;
     vector<C> in(n), out(n);
     copy(a.begin(),a.end(), in.begin());
     for(int i = 0; i < b.size(); i++) in[i].imag(b[i]);
     fft(in);
     for (C& x : in) x *= x;
     for(int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i]);
     fft(out);
     for(int i = 0; i < res.size(); i++) res[i] = round(imag(out[i]) / (4 * n));
     return res;
 }

 /*
     * Author: chilli
     * Date: 2019-04-25
     * License: CC0
     * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf
     * Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers
     * as long as $N\log_2N\cdot \text{mod} < 8.6 \cdot 10^{14}$ (in
```

```
practice $10^{16}$ or higher).
     * Inputs must be in $[0, \text{mod})$.
     * Time: O(N \log N), where $N = |A|+|B|$ (twice as slow as NTT
or FFT)
     * Status: stress-tested
     * Details: An in-depth examination of precision for both FFT and
 FFTMod can be found
     * here (https://github.com/simonlindholm/fft-precision/blob/
master/fft-precision.md)
 */
 // multiplica dois polinomios modulo algum inteiro
 template<int M> vl convMod(const vl &a, const vl &b) {
     if (a.empty() || b.empty()) return {};
     vl res(a.size() + b.size() - 1);
     int B=32-__builtin_clz(res.size()), n=1<<B, cut=int(sqrt(M));
     vector<C> L(n), R(n), outs(n), outl(n);
     for(int i = 0; i < a.size(); i++) L[i] = C((int)a[i] / cut, (int
)a[i] % cut);
     for(int i = 0; i < b.size(); i++) R[i] = C((int)b[i] / cut, (int
)b[i] % cut);
     fft(L), fft(R);
     for(int i = 0; i < n; i++) {
         int j = -i & (n - 1);
         outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
         outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
     }
     fft(outl), fft(outs);
     for(int i = 0; i < res.size(); i++) {
         ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
         ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
         res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
     }
     return res;
 }

 /*
     * Author: chilli
     * Date: 2019-04-16
     * License: CC0
     * Source: based on KACTL's FFT
     * Description: ntt(a) computes $\hat f(k) = \sum_x a[x] g^{xk}$
for all $k$, where $g=\text{root}^{(mod-1)/N}$.
     * N must be a power of 2.
     * Useful for convolution modulo specific nice primes of the form
 $2^a b+1$,
     * where the convolution result has size at most $2^a$. For
arbitrary modulo, see FFTMod.
     \texttt{conv(a, b) = c}, where $c[x] = \sum a[i]b[x-i]$.
     For manual convolution: NTT the inputs, multiply
     pointwise, divide by n, reverse(start+1, end), NTT back.
     * Inputs must be in [0, mod).
     * Time: O(N \log N)
     * Status: stress-tested
 */
 const ll mod = (119 << 23) + 1, root = 62; // = 998244353
 // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
 // and 483 << 21 (same root). The last two are > 10^9.
 void ntt(vl &a) {
     int n = a.size(), L = 31 - __builtin_clz(n);
     static vl rt(2, 1);
     for (static int k = 2, s = 2; k < n; k *= 2, s++) {
         rt.resize(n);
         ll z[] = {1, modpow(root, mod >> s)};
         for(int i = k; i < 2*k; i++) rt[i] = rt[i / 2] * z[i & 1] %
mod;
     }
```

```
        vi rev(n);
        for(int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
        for(int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
            for (int i = 0; i < n; i += 2 * k) for(int j = 0; j < k; j++) {
                ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
                a[i + j + k] = ai - z + (z > ai ? mod : 0);
                ai += (ai + z >= mod ? z - mod : z);
            }
    }
    vl conv_ntt(const vl &a, const vl &b) {
        if (a.empty() || b.empty()) return {};
        int s = a.size() + b.size() - 1, B = 32 - __builtin_clz(s),
            n = 1 << B;
        int inv = modpow(n, mod - 2);
        vl L(a), R(b), out(n);
        L.resize(n), R.resize(n);
        ntt(L), ntt(R);
        for(int i = 0; i < n; i++)
            out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
        ntt(out);
        return {out.begin(), out.begin() + s};
    }
    ll modpow(ll b, ll e) {
        ll ans = 1;
        for (; e; b = b * b % mod, e /= 2)
            if (e & 1) ans = ans * b % mod;
        return ans;
    }
};
```

2.3.2 FFT modular (NTT)

To use a modular FFT, also known as NTT, you need a n-th root of ur modulus and then define some values in general u can say : $mod = x * 2^k$

$G = 3$ or 5

$root = G^{mod/(2^k)}$

$root_1 = root^{-1}$

$root_{pw} = 2^k$

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

typedef long long ll;

const ll mod = 998244353;
const int root = 565042129;//3^(mod/root_pw)
const int root_1 = 950391366;//inv(root)
const int root_pw = 1<<20;
const int N=100100;

int fat[N];
vector<int> a[N];
int exp(int n,int b)
{
  int rs=1;
  while(b)
  {
    if(b&1)
      rs=(int)(1LL*rs*n%mod);
    n=(int)(1LL*n*n%mod);
    b>>=1;
  }
```

```
  return rs;
}

void fft(vector<int> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);

        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
                int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
                a[i+j] = u + v < mod ? u + v : u + v - mod;
                a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
                w = (int)(1LL * w * wlen % mod);
            }
        }
    }

    if (invert) {
        int n_1 = exp(n, mod-2);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}

void multiply(vector<int>  &a, vector<int>  &b) {
    if(b.size()<=30)
    {
      vector<int> res(b.size()+a.size(),0);
      for (int i = 0; i < a.size(); ++i)
      {
        for (int j = 0; j < b.size(); ++j)
        {
          res[i+j]+=(int)(1LL*a[i]*b[j]%mod);
          if(res[i+j]>mod)
            res[i+j]-=mod;
        }
      }
      a=res;
      return ;
    }
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    a.resize(n);
    b.resize(n);

    fft(a, 0);
    fft(b, 0);
    for (int i = 0; i < n; i++)
        a[i] = (int)(1LL*a[i]*b[i]%mod);
    fft(a, 1);
```

```
}



int main(int argc, char const *argv[])
{
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  // cout<<exp(3,952LL)<<" "<<exp(exp(3,952LL),mod-2)<<'\n';

  fat[0]=1;
  for (int i = 1; i < N; ++i)
  {
    fat[i]=(int)(1LL*fat[i-1]*i%mod);
  }
  int q;
  cin>>q;
  while(q--)
  {
    int n;
    cin>>n;
    for (int i = 0; i < n; ++i)
    {
      a[i]=vector<int>(2);
      int x;
      cin>>x;
      a[i][0]=(1);
      a[i][1]=(x%mod);
    }
    for (int i = 1; i < n; i<<=1)
    {
      for (int j = 0; j+i < n ; j+=(i<<1))
      {
        multiply(a[j],a[j+i]);
      }
    }
    int rs=0;
    for (int i = 1; i <= n; ++i)
    {
      rs += (int)(1LL*(1LL*a[0][i]*fat[i]%mod)*fat[n-i]%mod);
      if(rs>mod)
        rs-=mod;
    }
    rs = (int)(1LL*rs*exp(fat[n],mod-2)%mod);
    cout<<rs<<"\n";
  }
  return 0;
}
```

2.3.3 FFT To solve String with Wildcards

Given two strings s and t, with t having wildcards that can match with any character, returns the number of ocurrence of t in s. Solution $O(nlogn)$.

(Original Problem: Maraton Nacional de Programacion Colombia 2016 - Wildcards).

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

using namespace std;


#ifdef __WIN32__
#define getchar_unlocked getchar
#define putchar_unlocked putchar
#endif

inline void writeChar(char c) {
```

```
    putchar_unlocked(c);
}

bool readChar(char &c) {
    c = getchar_unlocked();
    return c != EOF;
}

template<typename T>
inline void writeInt(T n){
    register int idx = 20;
    if( n < 0 ) putchar_unlocked('-');
    n = abs(n);
    char out[21];
    out[20] = ' ';
    do{
        idx--;
        out[idx] = n % 10 + '0';
        n/= 10;
    }while(n);
    do{ putchar_unlocked(out[idx++]); } while (out[idx] != ' ');
}

struct complex_t {
  double a {0.0}, b {0.0};
  complex_t(){
  }
  complex_t(double na, double nb = 0.0) : a{na}, b{nb} {}
  const complex_t operator+(const complex_t &c) const {
    return complex_t(a + c.a, b + c.b);
  }
  const complex_t operator-(const complex_t &c) const {
    return complex_t(a - c.a, b - c.b);
  }
  const complex_t operator*(const complex_t &c) const {
    return complex_t(a * c.a - b * c.b, a * c.b + b * c.a);
  }
  const complex_t operator/(double r) const {
    return complex_t(a / r, b / r);
  }
};

using cd = complex_t;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
```

```
                a[i+j+len/2] = u - v;
                w = w * wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x = x / n;
    }
}

void multiply(vector< cd > const &a, vector< cd > const &b, vector< int
   > &result){
  vector< cd > fa(a.begin(), a.end()), fb(b.begin(), b.end());
  int n = 1;
  while(n < a.size() + b.size()){
    n <<= 1;
  }

  fa.resize(n);
  fb.resize(n);

  fft(fa, false);
  fft(fb, false);

  for(int i = 0 ; i < n ; i++){
    fa[i] = fa[i] * fb[i];
    // printf("%lf\n", fa[i].a);
  }

  fft(fa, true);

  for(int i = 0 ; i < result.size() ; i++){
    result[i] = floor(fa[i].a + 1e-5);
  }
}

vector< int > sum;
const int N = int(1e5 + 10);
vector< cd > a, b;
vector< int > r;

char s[N], t[N];
double pre[30];

int main(){
  int n = 0, m = 0;
  char c;

  for(int i = 0 ; i < 26 ; i++){
    pre[i] = (2.0 * PI * i) / 26.0;
  }

  while(readChar(c)){
    n = m = 0;

    s[n++] = c;

    while(readChar(c)){
      if(c == '\n')
        break;

      s[n++] = c;
    }

```

```
    while(readChar(c)){
      if(c == '\n'){
        break;
      }

      t[m++] = c;
    }

    if(m > n){
      writeChar('0');
      writeChar('\n');
      continue;
    }

    a.resize(n);
    b.resize(m);
    r.resize(n);

    sum.resize(n);

    reverse(t, t + m);

    for(int i = 0 ; i < n ; i++){
      double alphai = pre[s[i] - 'a'];

      a[i] = {cos(alphai), sin(alphai)};
    }

    int c = 0;

    for(int i = 0 ; i < m ; i++){
      if(t[i] == '?'){
        c++;
        b[i] = {0.0, 0.0};

        continue;
      }

      double bi = pre[t[i] - 'a'];

      b[i] = {cos(bi), -sin(bi)};
    }

    multiply(a, b, r);

    int cnt = 0;

    for(int i = m - 1 ; i < n ; i++){
      // printf("%d\n", r[i]);

      if(r[i] == m - c){
        cnt++;
        // printf("%d\n", i);
      }
    }

    writeInt(cnt);
    writeChar('\n');

    sum.clear();
  }

  return 0;
}
```

2.3.4 FWHT (fft using xor/and/or)

FFT but the expoent operation is xor/and/or. There are some crazy problems where u need to multiply N polynomials of the form $x^0 + x^i$ and the i can be up to N. the main approach is to use some D&C to multiply those and to keep the polynomials divide in two parts. It is also coded down here. Operations can be modular.

```


void FWHT(vector<int>& a, bool inv) {
    int n = a.size();
  for (int step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step)
            rep(j,i,i+step) {
                int &u = a[j], &v = a[j + step];
                tie(u, v) = inv ? mp(v - u, u) : mp(v, u + v); // AND
                // tie(u, v) = inv ? mp(v, u - v) : mp(u + v, u); // OR
   /// include-line
                // tie(u, v) = mp(u + v, u - v); // XOR /// include-line
            }
  }
  // if (inv) for (int& x : a) x /= n; // XOR only /// include-line
}
vector<int> multiply(vector<int> a, vector<int> b) {
  int n = 1;
    while(n < a.size())
        n<<=1;
    a.resize(n);
    b.resize(n);
    FWHT(a, 0);
    FWHT(b, 0);
  rep(i,0,n)
        a[i] *= b[i];
  FWHT(a, 1);
    return a;
}

pii a[N];
// multiply alot of polinoms of the form (x^0 + x^i) / can be done in
   Alog(A)^2, where A is max i
// probably it will be modular
pair<vector<int>,vector<int>> Multi_FWHT(int l, int r)
{
  if(l == r)
  {
    vector<int> x(1,a[l].ft);
    vector<int> y(1,a[l].sd);
    return mp(x,y);
  }
  int mid = (l+r) >> 1;
  auto [a,c] = Multi_FWHT(l,mid);
  auto [b,d] = Multi_FWHT(mid+1,r);
  int m = r-l+1;
  vector<int> ab,bc,ad,cd;
  ab = multiply(a,b);
  bc = multiply(b,c);
  ad = multiply(a,d);
  cd = multiply(c,d);
  ab.resize(m);
  bc.resize(m);
  rep(i,0,(m)/2){
    ab[i+m/2] = cd[i];
    bc[i+m/2] = ad[i];
  }
  return mp(ab,bc);
}
```

2.4 Gauss

2.4.1 Markov Chains

If we have $E(i) = 1 + \sum_{j=0}^{N-1} P[i][j] * E(j)$
We can model as a System of linear equations of the form:
$E(i) - \sum_{j=0}^{N-1} P[i][j] * E(j) = 1$ So:

$$\begin{bmatrix} 1-P[0][0] & -P[0][1] & \cdots & -P[0][N-1] \\ -P[1][0] & 1-P[1][1] & \cdots & -P[1][N-1] \\ \vdots & \vdots & \ddots & \vdots \\ -P[N-1][0] & -P[N-1][1] & \cdots & 1-P[N-1][N-1] \end{bmatrix} \times \begin{bmatrix} E(0) \\ E(1) \\ \vdots \\ E(N-1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Works with probabilities as well, but without the 1 in the sum.
Gauss Solves a System of linear equations in $O(min(N, M) * N * M)$, with N being the number of variables and M being the number of equations.

```
struct Gauss{
    typedef vector<double> vd;
    typedef vector<long long int> vl;
    typedef vector<int> vi;
    const double eps = 1e-12;

    /**
     * Author: Per Austrin, Simon Lindholm
     * Date: 2004-02-08
     * License: CC0
     * Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned.
     *  Returns rank, or -1 if no solutions. Data in $A$ and $b$ is lost.
     * Time: O(n^2 m)
     * Status: tested on kattis:equationsolver, and bruteforce-tested mod 3 and 5 for n,m <= 3
     */
    int solveLinear(vector<vd>& A, vd& b, vd& x) {
        int n = A.size(), m = x.size(), rank = 0, br, bc;
        if (n) assert(A[0].size() == m);
        vi col(m); iota(col.begin(),col.end(), 0); // fills the range with increasing values starting with value 0

        for(int i = 0; i < n; i++){
            double v, bv = 0;

            // busca pelo maior elemento
            for(int r = i; r < n; r++) for(int c = i; c < m; c++)
                if ((v = fabs(A[r][c])) > bv)
                    br = r, bc = c, bv = v;
            // bv eh o maior elemento, se todos os elementos sao iguais a zero
            if (bv <= eps) {
                // se no vetor de resposta tiver um cara diferente de zero, nao existe x tal que 0*x != 0
                for(int j = i; j < n; j++) if (fabs(b[j]) > eps) return -1;
                break;
            }

            swap(A[i], A[br]); // trocar a linha atual pela linha que tem o maior elemento
            swap(b[i], b[br]); // trocar a linha pra ficar igual no vetor de resposta
            swap(col[i], col[bc]); // trocar no vetor de coluna
            for(int j = 0; j < n; j++) swap(A[j][i], A[j][bc]); // trocar os elementos da coluna atual pelos elementos da coluna do maior elemento

```

```
            bv = 1/A[i][i]; // a razao necessaria pra fazer o elemento atual da diagonal ser 1
            // eu nao atualizo de verdade os valores de A[i][i] e b[i] nao sei pq, deve ser pq fica mais rapido nao atualizar eles de vdd
            for(int j = i+1; j < n; j++){
                double fac = A[j][i] * bv; // para zerar o elemento A[j][i] precisamos subtair dele A[i][i]*(1/A[i][i] * A[j][i]) = A[i][i]*fac
                // esse fac eh o valor que eu vou usar pra eliminar a linha j, ou seja, Linha_j <- Linha_j - Linha_i * fac, e eh isso que
                // ta fazendo aqui embaixo
                b[j] -= fac * b[i];
                for(int k = i+1; k < m; k++) A[j][k] -= fac*A[i][k];
            }
            // se eu consegui zerar essa coluna, entao a variavel dessa coluna eh diferente de zero (eh uma equacao linearmente independente)
            rank++;
        }

        x.assign(m, 0);
        for (int i = rank; i--;) {
            // atribuo as respostas
            b[i] /= A[i][i];
            x[col[i]] = b[i];
            for(int j = 0; j < i; j++) b[j] -= A[j][i] * b[i]; // o valor da variavel x_i eu descobri agora eu desconto o valor dela nas equacoes acima
        }
        return rank; // (multiple solutions if rank < m)
    }
    // tem que passar o modulo
    template <int M> int solveLinear(vector<vl>& A, vl& b, vl& x) {
        int n = A.size(), m = x.size(), rank = 0, br, bc;
        if (n) assert(A[0].size() == m);
        vi col(m); iota(col.begin(),col.end(), 0); // fills the range with increasing values starting with value 0

        for(int i = 0; i < n; i++) for(int j = 0; j < m; j++) A[i][j] = (A[i][j]%M + M)%M;
        for(int i = 0; i < n; i++) b[i] = (b[i]%M + M)%M;

        for(int i = 0; i < n; i++){
            ll v, bv = 0;

            // busca pelo maior elemento
            for(int r = i; r < n; r++) for(int c = i; c < m; c++)
                if ((v = A[r][c]))
                    br = r, bc = c, bv = v;
            // bv eh o maior elemento, se todos os elementos sao iguais a zero
            if (bv == 0) {
                // se no vetor de resposta tiver um cara diferente de zero, nao existe x tal que 0*x != 0
                for(int j = i; j < n; j++) if (b[j]) return -1;
                break;
            }

            swap(A[i], A[br]); // trocar a linha atual pela linha que tem o maior elemento
            swap(b[i], b[br]); // trocar a linha pra ficar igual no vetor de resposta
            swap(col[i], col[bc]); // trocar no vetor de coluna
            for(int j = 0; j < n; j++) swap(A[j][i], A[j][bc]); // trocar os elementos da coluna atual pelos elementos da coluna do maior elemento

```

```cpp
            bv = inv(A[i][i],M); // a razao necessaria pra fazer o elemento atual da diagonal ser 1
            // eu nao atualizo de verdade os valores de A[i][i] e b[i] nao sei pq, deve ser pq fica mais rapido nao atualizar eles de vdd
            for(int j = i+1; j < n; j++){
                ll fac = A[j][i] * bv % M; // para zerar o elemento A[j][i] precisamos subtair dele A[i][i]*(1/A[i][i] * A[j][i]) = A[i][i]*fac
                // esse fac eh o valor que eu vou usar pra eliminar a linha j, ou seja, Linha_j <- Linha_j - Linha_i * fac, e eh isso que
                // ta fazendo aqui embaixo
                b[j] = (b[j] - fac * b[i]%M + M)%M;
                for(int k = i+1; k < m; k++) A[j][k] = (A[j][k] - fac*A[i][k]%M + M)%M;
            }
            // se eu consegui zerar essa coluna, entao a variavel dessa coluna eh diferente de zero (eh uma equacao linearmente independente)
            rank++;
        }

        x.assign(m, 0);
        for (int i = rank; i--;) {
            // atribuo as respostas
            b[i] = b[i] * inv(A[i][i],M)%M;
            x[col[i]] = b[i];
            for(int j = 0; j < i; j++) b[j] = (b[j] - A[j][i] * b[i]%M + M)%M; // o valor da variavel x_i eu descobri agora eu desconto o valor dela nas equacoes acima
        }
        return rank; // (multiple solutions if rank < m)
    }

    void show(vector<vl>& A, vl& b){
        int n = A.size(), m = b.size();
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++) cout << A[i][j] << ' ';
            cout << "= " << b[i] << '\n';
        }
    }

    ll modpow(ll b, ll e, ll mod) {
        ll ans = 1;
        for (; e; b = b * b % mod, e /= 2)
            if (e & 1) ans = ans * b % mod;
        return ans;
    }

    ll inv(ll x, ll mod){
        return modpow(x,mod-2,mod);
    }
};
```

2.4.2 Modular 2 (bitset)

Problem: a grid of switchs and each press in a switch i,j swaps the state of himself and the ones which are adjacent to it.

If there is a way to make all on, print the switchs pressed.

```cpp
//https://www.spoj.com/problems/DFLOOR/en/
#include <bits/stdc++.h>

using namespace std;

const int N = 300;

int gauss(vector< bitset< N > > a, int n, int m, bitset< N > &ans){
```

```
    vector< int > where(m, -1);
    for(int col = 0, row = 0 ; col < m && row < n ; ++col){
        for(int i = row ; i < n ; ++i){
            if(a[i][col]){
                swap(a[i], a[row]);
                break;
            }
        }

        if(!a[row][col]){
            continue;
        }

        where[col] = row;

        for(int i = 0 ; i < n ; i++){
            if(i != row && a[i][col]){
                a[i] ^= a[row];
            }
        }

        ++row;
    }

    for(int i = 0 ; i < m ; i++){
        if(where[i] != -1){
            ans[i] = a[where[i]][m];
        }
    }

    // cout << " ------- \n";
    // for(auto u: a){
    //     cout << u << "\n";
    // }

    for(int i = 0 ; i < n ; i++){
        int sum = 0;
        for(int j = 0 ; j < m ; j++){
            sum += ans[j] * a[i][j];
        }

        if((sum % 2) != a[i][m]){
            return 0;
        }
    }

    return 1;
}

char tab[20][20];
int vi[4] = {0, 0, 1, -1};
int vj[4] = {1, -1, 0, 0};
int x, y;

bool ok(int a, int b){
    return 0 <= a && a < x && 0 <= b && b < y;
}

int main(){
    while(scanf("%d %d", &x, &y) != EOF){
        if(x == 0 && y == 0){
            break;
        }

        swap(x, y);

```

```
        for(int i = 0 ; i < x ; i++){
            for(int j = 0 ; j < y ; j++){
                scanf("\n%c", &tab[i][j]);
            }
        }

        vector< bitset< N > > q;

        for(int i = 0 ; i < x ; i++){
            for(int j = 0 ; j < y ; j++){
                bitset< N > v;

                for(int k = 0 ; k < 4 ; k++){
                    int xi = vi[k] + i;
                    int xj = vj[k] + j;

                    if(ok(xi, xj)){
                        v[xi * y + xj] = 1;
                    }
                }

                v[i * y + j] = 1;
                v[x * y] = !(tab[i][j] - '0');
                q.push_back(v);
                // cout << v << "\n";
            }
        }

        bitset< N > ans;
        if(gauss(q, q.size(), x * y, ans)){
            vector< int > ians;
            for(int i = 0 ; i < x * y ; i++){
                if(ans[i]){
                    ians.push_back(i);
                }
            }

            printf("%lu\n", ians.size());

            for(auto u: ians){
                printf("%d %d\n", (u % y) + 1, (u / y) + 1);
            }
        }else{
            printf("-1\n");
        }
    }

    return 0;
}
```

2.4.3 Matrix rank

The rank of a matrix is the largest number of linearly independent rows/columns of the matrix. The rank is not only defined for square matrices. We can see that the other variables are ”free”, so can be uses to combinatorial problems.

Example: Given a set of n integers, how many subsets have xor not equal to 0?

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 200100;
const int M = 60;

const ll bmod = 1000'000'007LL;
const ll mod = 2;

```

```
ll a[N][M];

ll fexp(ll x, ll b, ll mod)
{
    ll rs = 1;
    while(b)
    {
        if(b&1)
            rs = rs * x % mod;
        x = x * x % mod;
        b >>= 1;
    }
    return rs;
}
ll inv(ll x)
{
    return 1;
}
int n;
ll compute_rank() {

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < M; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && a[j][i] > 0)
                break;
        }
        // cout << i << endl;
        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < M; ++p)
                a[j][p] = a[j][p] * inv(a[j][i]) % mod;
            for (int k = 0; k < n; ++k) {
                if (k != j && a[k][i] > 0) {
                    for (int p = i + 1; p < M; ++p)
                        a[k][p] = (a[k][p] - a[j][p] * a[k][i] % mod + mod) % mod;
                }
            }
        }
    }
    // error(rank);
    return fexp(mod,n-rank,bmod);
}


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    cin >> n;
    for(int i = 0; i < n; i++){
        ll x;
        cin >> x;
        for(int j = 0; j < M; j++)
        {
            if(x & (1LL<<j))
                a[i][j] = 1;
        }

    }
    ll at = compute_rank();
```

```
    // error(at);
    cout << (fexp(mod, n, bmod) - at + bmod) %bmod  << '\n';

    return 0;
}
```

2.4.4 Xor Basis

Cebolinha me mostrou essa parada, fazendo uma solução pro mesmo problema acima usando isso.

Da pra gente saber se um X pertence ao XOR de algum subconjunto dos seus caras. o total de caras q pertence ao subconjunto é $2^{B.size()}$.

```
struct Basis {
  vector<ll> B;
  ll reduce(ll x) {
    for (auto b : B) x = min(x, x^b);
    return x;
  }
  void insert(ll x) {
    ll r = reduce(x);
    if (r) B.push_back(r);
  }
};
```

2.5 Chinese Remainder Theorem

2.5.1 Algorithm

Given (a, n, b, m), find a X such that $X = a \pmod n$ and $X = b \pmod m$. X is given mod LCM(n, m).

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

#define ll long long

ll gcd(ll a, ll b, ll &x, ll &y) {
    if(a == 0){
        x = 0;
        y = 1;
        return b;
    }
    ll x0, y0;
    ll g = gcd(b%a, a, x0, y0);
    y = x0;
    x = y0-x0*(b/a);
    return g;
}

// Return (X, Y) where Y = lcm(n, m) and X = a mod n and X = b mod m and
    0 <= X < Y
// X = -1 if there is not solution
pair<ll, ll> crt(ll a, ll n, ll b, ll m){
  if(a < 0) a += n;
  if(a >= n) a %= n;
  if(b < 0) b += m;
  if(b >= m) b %= m;

  ll x1, x2;
  ll d = gcd(n, m, x1, x2);
  ll lcm = n * m / d;
  if((a - b) % d != 0)
    return {-1, lcm};

  ll x = (a + (n*x1) * (__int128)((b - a)/d)) % lcm;
  if(x < 0) x += lcm;
```

```
  return {x, lcm};
}

int main(){
  ios::sync_with_stdio(0);
  cin.tie(0);

  int T;
  cin >> T;
  while(T--){
    ll n, m, a, b;
    cin >> a >> n >> b >> m;

    pair<ll, ll> ans = crt(a, n, b, m);
    if(ans.first == -1)
      cout << "no solution\n";
    else cout << ans.first << " " << ans.second << "\n";
  }



  return 0;
}
```

2.5.2 Lucas Theorem

Given n, k and m, find $\binom{n}{k}$ mod m, where $(n, k \leq 10^9)$, m is a square-free number less than or equal to 10^9 and each prime that appears on m is less than 50. A Square-Free number is a number that, in his prime factorization, each prime appears exactly once.

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

#define ft first
#define sd second
#define mp make_pair
#define mt make_tuple
#define eb emplace_back
#define pb push_back

using ll = long long;

vector<ll> fat, ifat, primes;

ll gcd(ll a, ll b, ll &x, ll &y) {
    if(a == 0){
        x = 0;
        y = 1;
        return b;
    }
    ll x0, y0;
    ll g = gcd(b%a, a, x0, y0);
    y = x0;
    x = y0-x0*(b/a);
    return g;
}

ll fexp(ll a, ll b, ll p) {
    a %= p;
    ll rs = 1;
    while(b>0){
        if(1ll&b) rs = rs*a%p;
        a = a*a%p;
        b>>=1ll;
    }
    return rs;
```

```
}

void _set(int p) {
    fat.assign(p, 0);
    ifat.assign(p, 0);

    fat[0]=fat[1]=1;
    ifat[0]=ifat[1]=1;
    for(ll i=2; i<p; i++){
        fat[i]=(fat[i-1]*i)%p;
        ifat[i]=fexp(fat[i],p-2, p);
    }
}

void factorize(int m) {
    primes.clear();
    for(int i=2; i<= 50; i++) {
        if(m%i==0){
            primes.pb(i);
            while(m%i==0) m/=i;
        }
    }
}

ll Lucas(ll n, ll r, ll p) {
    if(r<0 || r>n) return 0;
    if(r==0 || r==n) return 1;

    if(n>=p) return (Lucas(n/p,r/p,p)*Lucas(n%p,r%p,p))%p;
    return ((fat[n]*ifat[r])%p*ifat[n-r])%p;
}

pair<ll,ll> crt(ll a, ll n, ll b, ll m) {
    if(a<0) a += n;
    if(a>=n) a %= n;
    if(b<0) b += m;
    if(b >= m) b %= m;
    ll x1, x2;
    ll d = gcd(n, m, x1, x2);
    ll lcm = n*m/d;
    if((a-b)%d != 0){
        return mp(-1, lcm);
    }

    ll x = (a+(n*x1)*(__int128_t)((b-a)/d))%lcm;
    if(x<0) x+=lcm;
    return mp(x, lcm);
}

void test() {
    ll n, r, m;
    cin >> n >> r >> m;
    factorize(m);
    ll rs=0, M = 1;
    for(int i=0; i<primes.size(); i++) {
        _set(primes[i]);
        tie(rs, M) = crt(rs, M, Lucas(n, r, primes[i]), primes[i]);
    }
    cout<<rs<<"\n";
}

int32_t main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    // cout<<fixed<<setprecision(10);

    int t = 1;
```

```
    cin >> t;

    for(int i=1; i<=t; i++){
        // cout<<"Case "<<i<<": ";
        test();
    }

    return 0;
}
```

2.6 Simplex

Maximize $\mathbf{c}^{\mathbf{T}}\mathbf{x}$ subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$

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

const int MAX_N = 505;
const double eps = 1e-9;
const int inf = 1e9;
const int MAX_M = 20005;
class Simplex
{
  public:
    int n, m;
    double A[MAX_M][MAX_N], B[MAX_M], C[MAX_N];
    void pivot(int l, int e)
    {
        B[l] /= A[l][e];

        for (int i = 1; i <= n; ++i)
            if (i != e)
                A[l][i] /= A[l][e];

        A[l][e] = 1 / A[l][e];

        for (int i = 1; i <= m; ++i)
        {
            if (i != l && fabs(A[i][e]) > 0)
            {
                B[i] -= B[l] * A[i][e];
                for(int j = 1; j <= n; ++j)
                    if (j != e)
                        A[i][j] -= A[l][j] * A[i][e];
                 A[i][e] = -A[i][e] * A[l][e];
            }
        }
        for (int i = 1; i <= n; ++i)
            if (i != e)
                C[i] -= C[e] * A[l][i];
        C[e] = -C[e] * A[l][e];
    }
    double simplex()
    {
        double res = 0;
        while (true)
        {
            double tmp = 0;
            int e = 0, l = 0;
            for (int i = 1; i <= n; ++i)
                if (C[i] > tmp)
                    e = i, tmp = C[i];

            if (!e)
                return res;
```

```
                tmp = inf;
                for (int i = 1; i <= m; ++i)
                {
                    if(A[i][e] > 0 && tmp > B[i] / A[i][e])
                    {
                        tmp = B[i] / A[i][e];
                        l = i;
                    }
                }
                if(tmp == inf)
                    return tmp;
                else
                    res += tmp * C[e], pivot(l, e);
            }
        }
} solve;

int x[MAX_N];
int y[MAX_N];

const double pi = acos(-1);

double sq(double a)
{
    return a*a;
}

double dist(int i, int j)
{
    return sqrt(sq(x[i] - x[j]) + sq(y[i] - y[j]));
}

int main()
{
    int n;

    scanf("%d",&n);

    for(int i = 1; i <= n; i++)
    {
        scanf("%d %d",&x[i], &y[i]);
    }
    int idx = 1;
    solve.n = n;
    for(int i = 1; i <= n; i++)
    {
        for(int j = i+1; j <= n; j++)
        {
            solve.A[idx][i] = 1;
            solve.A[idx][j] = 1;
            solve.B[idx] = dist(i,j);
            idx++;
        }
        solve.C[i] = 1;
    }
    solve.m = idx;

    printf("%.10lf\n",2*pi*solve.simplex());
    return 0;
}
```

2.7 GCD Array Trick

Way to store all gcds in a array range . For each i you store every different gcd sub-segment that exists starting at i.

In this problem you need to find a $Max(gcd(A[l], \cdots, A[r]) * (r - l + 1))$.

```
//UVALive-6582
#include <bits/stdc++.h>
using namespace std;

const int N=100010;
typedef long long ll;
const ll mod=998244353LL;

map<ll, ll> sub_gcd[N];
ll a[N];
int main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int q;
    cin>> q;
    while(q--)
    {
        int n;
        cin>>n;
        for (int i = 0; i < n; ++i)
        {
            cin>>a[i];
            sub_gcd[i].clear();
        }
        sub_gcd[0][a[0]] = 1;
        ll ans=0;
        for(int i = 1; i < n; i++)
        {
            sub_gcd[i][a[i]] = 1;
            for(auto it: sub_gcd[i - 1])
            {
                ll new_gcd = __gcd(it.first, a[i]);
                sub_gcd[i][new_gcd] = max(sub_gcd[i][new_gcd], it.second
    + 1);
            }
        }
        for (int i = 0; i < n; ++i)
            for(auto it: sub_gcd[i])
                ans=max(ans,it.first*it.second);
        cout<<ans<<'\n';
    }
    return 0;
}
```

2.8 Implementation of Polynomials

Algorithm to find a root, and also to divide a poly by some (x+c).

```
typedef complex<double> cdouble;
int cmp(cdouble x, cdouble y = 0) {
    return cmp(abs(x), abs(y));
}
const int TAM = 200;
struct poly {
    cdouble poly[TAM]; int n;
    poly(int n = 0): n(n) { memset(p, 0, sizeof(p)); }
    cdouble& operator [](int i) { return p[i]; }
    poly operator ~() {
        poly r(n-1);
        for (int i = 1; i <= n; i++)
            r[i-1] = p[i] * cdouble(i);
        return r;
    }
    //divide poly by binom
```

```
    pair<poly, cdouble> ruffini(cdouble z) {
        if (n == 0) return make_pair(poly(), 0);
        poly r(n-1);
        for (int i = n; i > 0; i--) r[i-1] = r[i] * z + p[i];
        return make_pair(r, r[0] * z + p[0]);
    }
    cdouble operator ()(cdouble z) { return ruffini(z).second; }

    cdouble find_one_root(cdouble x) {
        poly p0 = *this, p1 = ~p0, p2 = ~p1;
        int m = 1000;
        while (m--) {
            cdouble y0 = p0(x);
            if (cmp(y0) == 0) break;
            cdouble G = p1(x) / y0;
            cdouble H = G * G - p2(x) - y0;
            cdouble R = sqrt(cdouble(n-1) * (H * cdouble(n) - G * G));
            cdouble D1 = G + R, D2 = G - R;
            cdouble a = cdouble(n) / (cmp(D1, D2) > 0 ? D1 : D2);
            x -= a;
            if (cmp(a) == 0) break;
        }
        return x;
    }
    vector<cdouble> roots() {
        poly q = *this;
        vector<cdouble> r;
        while (q.n > 1) {
            cdouble z(rand() / double(RAND_MAX), rand() / double(
  RAND_MAX));
            z = q.find_one_root(z); z = find_one_root(z);
            q = q.ruffini(z).first;
            r.push_back(z);
        }
        return r;
    }
};
```

2.9 Combinatorial

Multiple combinatorial solutions.
Number of ways to take K objects, between $N- > toma(N,K)$
Number of ways to put N objects in K boxes (stars and bars) $- > toma(N+K-1, K-1)$
Count how to label $N+K$ pairs of parentheses with K '('s already fixed $- > catalan(N,K)$

```
    struct Comb{
    vector<ll> fato,fatoinv;
    vector<vector<ll>> tomatoma;

    // setar so os fatorias e inversos
    Comb(int n){
        set_tam(n);
    }

    // setar so o triangulo de pascal
    Comb(int n,int m){
        set_tomatoma(n,m);
    }

    // setar os fatorias e triangulo de pascal
    Comb(int n2,int n,int m){
        set_tam(n2);
        set_tomatoma(n,m);
    }

```

```
    Comb(){}

    void set_tam(int n){
        fato.resize(n+10,-1);
        fatoinv.resize(n+10,-1);
    }

    void set_tomatoma(int n,int m){
        tomatoma.resize(n+10,vector<ll>(m+10));
        for(int i=0; i<=m; i++) tomatoma[0][i]=0;
        for(int i=0; i<=n; i++) tomatoma[i][0]=1%mod;
        for(int i=1; i<=n; i++) for(int j=1; j<=m; j++) tomatoma[i][j] =
    (tomatoma[i-1][j-1] + tomatoma[i-1][j])%mod;
    }

    ll inv(ll a){
        return fexp(a,mod-2);
    }

    ll fexp(ll a, ll b){
        ll ans=1;
        while(b){
            if(b&1) ans=ans*a%mod;
            b>>=1;
            a=a*a%mod;
        }
        return ans;
    }

    ll fat(ll x){
        if(ll(fato.size()) <= x){
            fato.resize(x+10,-1);
        }
        if(x <= 1) return 1;
        if(fato[x] != -1) return fato[x];
        return fato[x]=fat(x-1)*x%mod;
    }

    ll fatinv(ll x){
        if(ll(fatoinv.size()) <= x){
            fatoinv.resize(x+10,-1);
        }
        if(x <= 1) return 1;
        if(fatoinv[x] != -1) return fatoinv[x];
        return fatoinv[x]=fexp(fat(x),mod-2);
    }

    ll toma(ll n, ll k){
        if(n < 0) return 0;
        if(k > n) return 0;
        ll ans;
        if(tomatoma.size() > n && tomatoma[0].size() > k){
            ans = tomatoma[n][k];
        }else{
            ans=fat(n);
            ans=ans*fatinv(k)%mod;
            ans=ans*fatinv(n-k)%mod;
        }
        return ans;
    }

    //Count how to label N + K pairs of parentheses with K left ones
   already fixed
    //(k=0 means normal catalan)
    ll catalan(ll n, ll k){
        ll ans = (k+1) * toma(2*n + k, n) % mod;
```

```
        ans = ans * inv(n+k+1) % mod;
        return ans;
    }
};
```

2.9.1 Derangements

The number of derangements of n numbers, expressed as $!n$, is the number of permutations such that no element appears in its original position. Informally, it is the number of ways n hats can be returned to n people such that no person recieves their own hat.

Principle of Inclusion-Exclusion:

Suppose we had events $E_1, E_2, \ldots, E_n$, where event E_i corresponds to person i recieving their own hat. We would like to calculate $n! - |E_1 \cup E_2 \cup \cdots \cup E_n|$.

We subtract from $n!$ the number of ways for each event to occur; that is, consider the quantity $n! - |E_1| - |E_2| - \cdots - |E_n|$. This undercounts, as we are subtracting cases where more than one event occurs too many times. Specifically, for a permutation where at least two events occur, we undercount by one. Thus, add back the number of ways for two events to occur. We can continue this process for every size of subsets of indices. The expression is now of the form:

$$n! - |E_1 \cup E_2 \cup \cdots \cup E_n| = \sum_{k=1}^{n} (-1)^k \cdot (\text{number of permutations with } k \text{ fixed points})$$

For a set size of k, the number of permutations with at least k indices can be computed by choosing a set of size k that are fixed, and permuting the other indices. In mathematical terms:

$$\binom{n}{k}(n-k)! = \frac{n!}{k!(n-k)!}(n-k)! = \frac{n!}{k!}$$

Thus, the problem now becomes computing

$$n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$$

which can be done in linear time.

2.10 Lagrange Interpolation

Suppose u have a polynomial of order K, and want to find $f(N)$, and N is really big, with Lagrange Interpolation, we can define any $f(x)$ as a linear combination of k+1 $f(u)$'s. We can calculate $f(x)$ in $O(K)$ for each x.

We find an polynomial p_i satisfying $f(x_i) = y_i, f(x_j) = 0$ for $0 \leq j \leq K, j \neq i$ and all x_i are distincts values (we take $0, 1, 2, \ldots, K$ as values), these polynomials are $p_i = y_i * \prod_{j=0,j\neq i}^{K} \frac{x-x_j}{x_i-x_j}$ where y_i is the value calculated for the function $f(x)$ with $x = x_i$ and x is the value we want to calculate, the desired interpolation is gotten by summing these p_i)

$$f(x) = \sum_{i=0}^{K} \left(y_i * \prod_{j=0,j\neq i}^{K} \frac{x - x_j}{x_i - x_j} \right)$$

```
/*
    Alteracoes:

    Alterar a funcao g de dentro da funcao de lagrange
    Na funcao que faz o lagrange mesmo, alterar os parametros da funcao
  g de acordo com o que faz
*/
struct Lagrange{
    vector<ll> pref,suf;
    vector<ll> invfat;
    vector<ll> y;
    // valor x que quer calcular e o grau do polinomio n
    ll x,n;
    ll mod;
    // geralmente o negocio da funcao de dentro precisa de mais
  informacao
    ll pte;

```

```
    void set__(){
        pref[0]=1;
        suf[n+2]=1;
        for(ll i=1; i<=n+1; i++) pref[i]=pref[i-1]*((x-i)%mod)%mod;
        for(ll i=n+1; i>0; i--) suf[i]=suf[i+1]*((x-i)%mod)%mod;
        ll fat=1;
        invfat[0]=1;
        for(ll i=1; i<=n+1; i++){
            fat=fat*i%mod;
            invfat[i]=fexp(fat,mod-2);
        }
        y[0]=0;
        for(int i=1; i<=n+1; i++) y[i]=(y[i-1]+g())%mod;
    }

    void set2__(){
        pref[0]=1;
        suf[n+2]=1;
        for(ll i=1; i<=n+1; i++) pref[i]=pref[i-1]*((x-i)%mod)%mod;
        for(ll i=n+1; i>0; i--) suf[i]=suf[i+1]*((x-i)%mod)%mod;
    }

    Lagrange(ll x, ll n,ll mod, ll pte){
        pref.resize(n+3);
        suf.resize(n+3);
        invfat.resize(n+3);
        y.resize(n+3);

        this->x=x;
        this->n=n;
        this->mod=mod;
        this->pte=pte;

        set__();
    }

    // exponenciacao binaria
    ll fexp(ll a, ll b){
        a%=mod;
        ll ans=1;
        while(b){
            if(b&1) ans=ans*a%mod;
            b>>=1;
            a=a*a%mod;
        }
        return ans;
    }

    // se eu so quiser mudar o x que eu calculo do polinomio, so preciso
    alterar o prefixo e o sufixo
    void mudax(ll x){
        this->x = x;

        set2__();
    }

    // funcao de dentro do polinomio de lagrange
    ll g(){

    }

    // o solve, calcula o valor do polinomio no ponto x, com grau n (lg
    - lagrange)
    ll lg(){
        if(x <= n+1){
            ll ans=0;
```

```
            // calcula o somatorio ate o n normal
            for(int i=1; i<=x; i++) ans=(ans+g())%mod;
            return ans;
        }else{
            ll ans=0;
            for(int i=1; i<=n+1; i++){
                // g(i)*produtorio(x-j)/produtorio(i-j)
                ll aux=y[i]*pref[i-1]%mod;
                aux=aux*suf[i+1]%mod;
                aux=aux*invfat[n+1-i]%mod;
                aux=aux*invfat[i-1]%mod;

                // o -1 do produtorio(i-j)
                if((n+1-i)&1) aux*=-1;
                while(aux < 0) aux+=mod;

                // adiciona esse cara no polinomio de lagrange
                ans=(ans+aux)%mod;
            }

            return ans;
        }
    }
};
```

2.11 Discrete Log

You have an modular equation with $a, b, x \in \mathbb{Z}$, you want to find a value for x

$a^x \equiv b \pmod{m}$

If you only want to find a solution, you just have to consider values between $[0, m-1]$ because, by the Pigeonhole Principle, $a^0, a^1, \ldots, a^m$ you can only have m possible values, so minimally a^m it's going to be a value that already appeared and it's going to cycle. So you have to consider values between $[0, m-1]$.

However, the values of m can be big, $1e9$, there's a way to solve it in $s\sqrt{m}$, the Baby-step giant-step.

Let's consider that $gcd(a, m) = 1$ and rewrite the expression (remember you can write every integer with $x = q*p+r$, where $0 \le r < p$, and with that you can also write $x = (q+1)*p-r$, where $0 \le r < p$):

$$a^x \equiv b \pmod{m}$$
$$a^{q*p-r} \equiv b \pmod{m}$$
$$a^{q*p} * a^{-r} \equiv b \pmod{m}$$
$$a^{q*p} \equiv b * a^r \pmod{m}$$

If you determine a good value for p, you can decrease the range of possible values of both q and r, let $p = \sqrt{m}$, then $0 \le p \le \sqrt{m}+1$ (the plus 1 comes from the minus from the r) and $0 \le r < \sqrt{m}$, so you can precalculate the values of a^{q*p} for all $0 \le p \le \sqrt{m}$ and $b * a^r$ for all $0 \le r < \sqrt{m}$, so in the end you have $O(\sqrt{m})$. And to find values of q and r, you go through the values of a^{q*p} and use binary search or two point to find a value $b * a^r$ that is equal.

If $gcd(a, m) \ne 1$, then you can't calculate the inverse of a^r, however you can still try to solve it, let $g = gcd(a, m)$, if $g \nmid b$ then the equation doesn't have solution (if $g \mid a$ and $g \mid m$, then certainly, by linear combination, $g \mid b$). If $g \mid b$, then you can divide everyone by g. And do it until $gcd(a, m) \ne 1$. But after every time you do this operation you eliminate one a from a^x.

$$a^x \equiv b \pmod{m}$$
$$a^{x-1} * a \equiv b \pmod{m}$$
$$a^{x-1} * \frac{a}{g} \equiv \frac{b}{g} \left(\bmod \frac{m}{g}\right)$$
$$\ldots$$
$$a^{x-add} * \frac{a^{add}}{k} \equiv \frac{b}{k} \left(\bmod \frac{m}{k}\right)$$

After applying the operation add times you have to solve the problem with new values where $gcd(a, \frac{m}{k}) \ne 1$.

```cpp
// a^x === b (mod m)
ll lgd(ll a, ll b, ll m){
    a%=m;
    b%=m;
    if(a == 0 && b != 0) return -1;
    ll k=1,add=0,g;
    while((g = __gcd(a,m)) > 1LL){
        if(b == k) return add;
        if(b%g) return -1;
        b/=g;
        m/=g;
        add++;
        k = k*(a/g)%m;
    }
    ll n = sqrt(m)+1;

    ll fat=1;
    for(int i=0; i<n; i++) fat=fat*a%m;

    vector<pair<ll,ll>> big;
    ll aux=1;
    for(int i=0; i<n+10; i++){
        aux=aux*fat%m;
        big.push_back({aux*k%m,i+1});
    }
    // os caras que sao k*a^(p*n)
    sort(big.begin(),big.end(),[&](pair<ll,ll> x, pair<ll,ll> y){
        return x.first < y.first;
    });

    aux=1;
    vector<pair<ll,ll>> small;
    for(int i=0; i<=n; i++){
        small.push_back({b*aux%m,i});
        aux=aux*a%m;
    }
    // os caras que sao b*a^q
    sort(small.begin(),small.end(), [&] (pair<ll,ll> x, pair<ll,ll> y){
        return x.first < y.first;
    });

    // se ele quiser que a resposta tenha alguma propriedade
    ll ans=-1;
    int i=0,j=0;
    while(i < big.size() && j < small.size()){
        if(big[i].first == small[j].first) {
            if(ans == -1) ans=big[i].second*n-small[j].second+add;
            else ans=big[i].second*n-small[j].second+add;
            i++,j++;
        }else if(big[i].first < small[j].first) i++;
        else j++;
    }
    return ans;
}
```

2.12 Nelsi's anotations

2.12.1 Kaplansky's Lemma

Lemma 1

Given a set of size N, how many subsets of size K that do not contain consecutive elements exist?

Basically, we will rewrite the elements of our set as $+$ and $-$. Such that between each pair of $+$, there is a $-$. Initially, we will choose K plus signs. Then, we will have x_1 minus signs before the first plus, x_2

between the first and the second, and so on, up to x_{K+1}.

$$x_1, x_{K+1} \geq 0$$
$$x_i > 0 \ \forall\ 2 \leq i \leq K$$

We have $N - K$ minus signs to distribute. Since x_2 to x_K are positive integers, we will express x_1 and x_{K+1} as positive integers in the form of $x'_1 - 1$ and $x'_{K+1} - 1$, resulting in the equation:
$x'_1 - 1 + x_2 + x_3 + ... + x_{K-1} + x_K + x'_{K+1} - 1 = N - K$
$x'_1 + x_2 + x_3 + ... + x_{K-1} + x_K + x'_{K+1} = N - K + 2$
We know the formula for calculating how many integer solutions there are for a positive integer.
$\binom{M-1}{P}$ Assuming M is the desired answer and P is the number of coefficients.
In our problem, it takes the following form: $\binom{N-K+1}{K}$

Lemma 2

Given a circle with numbers from 1 to N, in how many ways can we choose K numbers without selecting neighboring ones?
We can divide the problem into 2 cases:
If N belongs to the set, we need to choose $K-1$ elements from $N-3$ options. Following the previous lemma, our answer is:
$\binom{N-K-1}{K-1}$
If N does not belong to the set, we have to choose K from $N - 1$ options.Following the previous lemma, our answer is:
$\binom{N-K}{K}$
Summing up the two answers we obtained, we have: $\binom{N-K-1}{K-1}\binom{N-K}{K} = \frac{N}{N-K}\binom{N-K}{K}$

2.12.2 Divisor Analysis - CSES

Number of Divisors

Each divisor of the number can be written as $\prod_{i=1}^{N} x_i^{\alpha_i}$ where $0 \leq \alpha_i \leq k_i$.
Since there are $k_i + 1$ choices for α_i, the number of divisors is simply $\prod_{i=1}^{N}(k_i + 1)$.
We can calculate this by iterating through the prime factors in $\mathcal{O}(N)$ time.

Sum of divisors

Let the sum of divisors when only considering the first i prime factors be S_i. The answer will be S_N.

$$S_i = S_{i-1} \sum_{j=0}^{k_i} x_i^j$$
$$= S_{i-1} \cdot \frac{x_i^{k_i+1} - 1}{x_i - 1}$$

We can calculate each S_i using fast exponentiation and modular inverses in $\mathcal{O}(N \log(\max(k_i)))$ time.

Product of divisors

Let the product and number of divisors when only considering the first i prime factors be P_i and C_i respectively. The answer will be P_N.

$$P_i = P_{i-1}^{k_i+1} \left(x_i^{k_i(k_i+1)/2}\right)^{C_{i-1}}$$

Again, we can calculate each P_i using fast exponentiation in $\mathcal{O}(N \log(\max(k_i)))$ time, but there's a catch! It might be tempting to use C_{i-1} from your previously-calculated values in part 1 of this problem, but those values will yield wrong answers.
This is because $a^b \not\equiv a^{b \bmod p} \pmod p$ in general. However, by Fermat's little theorem, $a^b \equiv a^{b \bmod (p-1)} \pmod p$ for prime p, so we can just store C_i modulo $10^9 + 6$ to calculate P_i.

2.12.3 Bracket Sequences II - CSES

For the "Bracket Sequences II"solution, we will first calculate the factorials and perform the following checks. Initially, we will check if the bracket sequence (BS) becomes irregular at any point; if it does, we return 0. Additionally, we will verify if our right bracket sequence (RBS) has all N elements; if it does, we return 1.
After these base case checks, we will proceed to calculate the Catalan number for this value.
The standard formula for the Catalan number is:

$C_n = \binom{2n}{n} - \binom{2n}{n-1}$

This formula means that out of the $2n$ opening parentheses, we will choose $n-1$ to close. We will make a small modification to it. We will keep track of the number of parentheses that have already been opened and closed and substitute that into the formula.

$C_n = \frac{2n-(open+close)}{(n-open)!(n-close)!} - \frac{2n-(open+close)}{(n-open-1)!(n-close+1)!}$

2.13 Multiplicative Functions

In number theory, a multiplicative function is an arithmetic function $f(n)$ of a positive integer n with the property that $f(1) = 1$ and

$$f(ab) = f(a)f(b)$$

whenever a and b are coprime.

Examples of multiplicative functions:

- $gcd(n,k)$: the greatest common divisor of n and k, as a function of n, where k is a fixed integer.
- $\varphi(n)$: Euler's totient function φ , counting the positive integers coprime to (but not bigger than) n
- $\mu(n)$: the Mobius function, the parity (-1 for odd, +1 for even) of the number of prime factors of square-free numbers; 0 if n is not square-free
- $\sigma k(n)$: the divisor function, which is the sum of the k-th powers of all the positive divisors of n (where k may be any complex number). Special cases we have
  - $\sigma 0(n) = d(n)$ the number of positive divisors of n,
  - $\sigma 1(n) = \sigma(n)$, the sum of all the positive divisors of n.

The main ideia of multiplicative functions is to use the linear sieve or just the sieve to calculate them. Normaly it will lead to some principle and u will need to calculate the solution to powers of primes and products of coprimes. for example we have for $\phi(x)$ that:

- $\phi(p) = p - 1$,
- $\phi(p^k) = p^k - p^{k-1}$,
- $\phi(a * b) = \phi(a) * \phi(b)$, if a and b are coprime,
- $\phi(a * b) = \phi(a) * \phi(b) * \frac{gcd(a,b)}{\phi(gcd(a,b))}$, if a and b arent coprime,

Then we can calculate phi using sieve using the following code:

```
std::vector <int> prime;
bool is_composite[MAXN];
int phi[MAXN];

void sieve (int n) {
  std::fill (is_composite, is_composite + n, false);
  phi[1] = 1;
  for (int i = 2; i < n; ++i) {
    if (!is_composite[i]) {
      prime.push_back (i);
      phi[i] = i - 1;          //i is prime
    }
    for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
      is_composite[i * prime[j]] = true;
      if (i % prime[j] == 0) {
        phi[i * prime[j]] = phi[i] * prime[j];  //prime[j] divides i
        break;
      } else {
        phi[i * prime[j]] = phi[i] * phi[prime[j]]; //prime[j] does not divide i
      }
    }
  }
}
```

mantaining the quantity of the smallest prime can be useful to some functions, such as $f(p^k) = k$.

```cpp
vector <int> prime;
bool is_composite[MAXN];
int func[MAXN], cnt[MAXN];

void sieve (int n) {
  fill (is_composite, is_composite + n, false);
  func[1] = 1;
  for (int i = 2; i < n; ++i) {
    if (!is_composite[i]) {
      prime.push_back (i);
      func[i] = 1; cnt[i] = 1;
    }
    for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
      is_composite[i * prime[j]] = true;
      if (i % prime[j] == 0) {
        func[i * prime[j]] = func[i] / cnt[i] * (cnt[i] + 1);
        cnt[i * prime[j]] = cnt[i] + 1;
        break;
      } else {
        func[i * prime[j]] = func[i] * func[prime[j]];
        cnt[i * prime[j]] = 1;
      }
    }
  }
}
```

2.13.1 Mobius Inversion

The Mobius function which is in the list above have a really interesting property. Lets define a function f as:

$$f(x) = \begin{cases} 1, & \text{for } x = 1 \\ 0, & \text{otherwise.} \end{cases}$$

we have an interesting property which can be used to solve alot of multiplicative function related problems.

$$f(x) = \sum_{d|x} \mu(d)$$

So lets say we need to calculate the following:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j) = 1]$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d|gcd(i,j)} \mu(d)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d=1}^{n}[d|gcd(i,j)] * \mu(d)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d=1}^{n}[d|i] * [d|j] * \mu(d)$$

$$\sum_{d=1}^{n}\mu(d) * (\sum_{i=1}^{n}[d|i]) * (\sum_{j=1}^{n}[d|j])$$

$$\sum_{d=1}^{n}\mu(d) * (\left\lfloor \frac{n}{d} \right\rfloor) * (\left\lfloor \frac{n}{d} \right\rfloor)$$

So we started with a $O(N^2 * log(N))$ solution went to a $O(N^3 * log(N))$ solution, but with the power of magic(known as math in some places) we ended with a $O(N)$ solution.

Some other possible problems using this:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} gcd(i,j)$$
$$\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=k]$$
$$\sum_{k=1}^{n} k * \sum_{a=1}^{\lfloor \frac{n}{k} \rfloor}\sum_{b=1}^{\lfloor \frac{n}{k} \rfloor}[gcd(a,b)=1]$$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * (\left\lfloor \frac{n}{d*k} \right\rfloor) * (\left\lfloor \frac{n}{d*k} \right\rfloor)$$

Next:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} lcm(i,j)$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{i*j}{gcd(i,j)}$$
$$\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}[gcd(i,j)=k] * \frac{i*j}{k}$$
$$\sum_{k=1}^{n} k * \sum_{a=1}^{\lfloor \frac{n}{k} \rfloor}\sum_{b=1}^{\lfloor \frac{n}{k} \rfloor}[gcd(a,b)=1] * a * b$$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * (\sum_{a=1}^{\lfloor \frac{n}{k} \rfloor}[d|a] * a) * (\sum_{b=1}^{\lfloor \frac{n}{k} \rfloor}[d|b] * b)$$
$a = d*p,\ b = d*q$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * (\sum_{p=1}^{\lfloor \frac{n}{k*d} \rfloor} d*p) * (\sum_{q=1}^{\lfloor \frac{n}{k*d} \rfloor} d*q)$$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * (\sum_{p=1}^{\lfloor \frac{n}{k*d} \rfloor} p) * (\sum_{q=1}^{\lfloor \frac{n}{k*d} \rfloor} q)$$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * (\sum_{p=1}^{\lfloor \frac{n}{k*d} \rfloor} p) * (\sum_{q=1}^{\lfloor \frac{n}{k*d} \rfloor} q)$$
the left part can be done with an PA, which will denoted as $PA(l,r)$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * (PA(1, \left\lfloor \frac{n}{k*d} \right\rfloor)^2)$$
let $l = k*d$
$$\sum_{k=1}^{n} k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * (PA(1, \left\lfloor \frac{n}{l} \right\rfloor)^2)$$
$$\sum_{l=1}^{n} (PA(1, \left\lfloor \frac{n}{l} \right\rfloor)^2) * \sum_{d'|l} \mu(d') * d' * l$$

Now lets analyze a problem with an vector A with max element being M and we need to calculate:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} lcm(A[i], A[j])$$

$$\sum_{d=1}^{M}\sum_{(a\in A\ \&\ d|a)}\sum_{(b\in A\ \&\ d|b)}[gcd(\frac{a}{d},\frac{b}{d})=1]*\frac{a*b}{d}$$

$$\sum_{d=1}^{M}\sum_{(a\in A\ \&\ d|a)}\sum_{(b\in A\ \&\ d|b)}\sum_{l=1}^{M}\mu(l)*[l|\frac{a}{d}][l|\frac{b}{d}]*\frac{a*b}{d}$$

$$\sum_{d=1}^{M}\frac{1}{d}*\sum_{l=1}^{M}\mu(l)*(\sum_{a\in A}\sum_{b\in A}[dl|a][dl|b]*a*b)$$

let $t = dl$

$$\sum_{t=1}^{M}\frac{1}{d}*\sum_{l|t}\frac{l}{t}*\mu(l)*(\sum_{a\in A}\sum_{b\in A}[t|a][t|b]*a*b)$$

$$\sum_{t=1}^{M}(\sum_{l|t}\frac{l}{t}*\mu(l))*(\sum_{a\in A\ \&\ t|a}a)^2$$

Capítulo III

Data Structures

3.1 Ordered Set

Same as Set, but with two new queries

- s.order_of_key(k): Number of items strictly smaller than k .
- s.find_by_order(k): k-th element in a set (counting from zero).

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
   tree_order_statistics_node_update
using namespace __gnu_pbds;

template<class T> using ordset = tree<
T,
null_type,
less<T>,
rb_tree_tag,
tree_order_statistics_node_update>;

ordset<int> os;
```

3.2 Fenwick Tree

3.2.1 Simples

```
/*
    Indexado em 1, n esqueca de especificar se a BIT for de sufixo

    Alteracoes:

    Adicionar o valor do off (valor dummy)
    Atualizar a funcao da BIT f
*/
template <class TT = ll>
struct Fen{
    vector<TT> fen;
    int n; // tamanho da BIT
    int pref; // flag que indica se a Fenwick eh no prefixo(1), ou no
   sufixo(0)
    const TT off_fen = ; // valor dummy

    Fen(int n_, int pref_=1) : pref(pref_),n(n_){
        fen.resize(n+10,off_fen);
    }
    ~Fen(){fen.clear();}

    // operacao da BIT
    TT f(TT x, TT y){
        return
    }
```

```

    void update(int x, TT v){
        if(x <= 0 || x > n) return;
        if(pref){
            while(x <= n){
                fen[x]=f(fen[x],v);
                x+=(x&-x);
            }
        }else{
            while(x){
                fen[x]=f(fen[x],v);
                x-=(x&-x);
            }
        }
    }

    TT query(int x){
        if(x <= 0 || x > n) return off_fen;
        TT ans = off_fen;
        if(pref){
            while(x){
                ans=f(ans,fen[x]);
                x-=(x&-x);
            }
        }else{
            while(x <= n){
                ans=f(ans,fen[x]);
                x+=(x&-x);
            }
        }
        return ans;
    }
    // TT query(int l, int r){
    //     return query(r) OPERACAO query(l);
    // }
};
```

3.2.2 2D

You are given an n×n grid representing the map of a forest. Each square is either empty or has a tree. Your task is to process q queries of the following types: Change the state (empty/tree) of a square. How many trees are inside a rectangle in the forest?

You can use the magic of Inclusion and Exclusion to solve this one.

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

typedef long long ll;
typedef pair<int,int> pii;

const int N = 1010;

int bit[N][N], a[N][N];

void updt(int x,int y,int v)
{
    if(!v)
        return;
    for(int i = x; i < N; i+=(i&-i))
    {
        for(int j = y; j < N; j+=(j&-j))
        {
            bit[i][j] += v;
        }
    }
}
```

```
int query(int x,int y)
{
    int v = 0;
    if(!x || !y)
        return 0;
    for(int i = x; i ; i-=(i&-i))
    {
        for(int j = y; j; j-=(j&-j))
        {
            v += bit[i][j];
        }
    }
    return v;
}


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n,q;
    string s;
    cin >> n >> q;
    for(int i = 1; i <= n; i++){
        cin >> s;
        for(int j = 1; j <= n; j++)
        {
            if(s[j-1] == '*'){
                a[i][j] = 1;
                updt(i, j, 1);
            }
        }
    }

    for(int i = 0; i < q; i++)
    {
        int t;
        cin >> t;
        if(t == 1)
        {
            int x, y;
            cin >> x >> y;
            updt(x, y, -a[x][y]);
            a[x][y] ^= 1;
            updt(x, y, a[x][y]);
        }else
        {
            int x1, x2, y1, y2;
            cin >> x1 >> y1 >> x2 >> y2;
            cout << query(x2, y2) - query(x2, y1-1) - query(x1-1, y2) + query(x1-1, y1-1) << '\n';
        }
    }
    return 0;
}
```

3.2.3 Fenwick Lifting

Binary Search on a BIT in $O(log(N))$

```
// This is equivalent to calculating lower_bound on prefix sums array
// LOGN = log(N)

int bit[N]; // BIT array

int bit_search(int v)
```

```
{
  int sum = 0;
  int pos = 0;

  for(int i=LOGN; i>=0; i--)
  {
    if(pos + (1 << i) < N and sum + bit[pos + (1 << i)] < v)
    {
      sum += bit[pos + (1 << i)];
      pos += (1 << i);
    }
  }

  return pos + 1; // +1 because 'pos' will have position of largest value less than 'v'
}
```

3.3 Sparse Table

3.3.1 Simple Sparse Table

Idempotent queries in $O(1)$, others $O(\log(n))$.

```
/*
    Indexado de 1.
    Responde uma operacao num subarray, a operacao precisa ser associativa e de preferencia ter uma identidade.
    Init: O(N*log(N)).
    Query: O(log(N)) ou O(1) para operacoes idempotentes.

    Alteracoes:
    Funcao f() da tabela, valor off
*/
template <class TT = ll>
struct SparseTable{
    int n; // tamanho
    vector<vector<TT>> tab; // sparse table
    vector<int> pot2; // log de cada valor
    int TETO; // Teto do log2(n)
    const TT off = ;

    SparseTable(int n_) : n(n_){
        pot2.resize(n+10);
        pot2[1] = 0;
        for(int i=2; i<=n; i++) pot2[i] = pot2[i>>1] + 1;
        tab.resize(n+10,vector<TT>(pot2[n]+1));
        TETO = pot2[n]+1;
    }
    SparseTable(int n_, vector<TT> & a) : n(n_){
        pot2.resize(n+10);
        pot2[1] = 0;
        for(int i=2; i<=n; i++) pot2[i] = pot2[i>>1] + 1;
        tab.resize(n+10,vector<TT>(pot2[n]+1));
        TETO = pot2[n]+1;
        build(a); // build da Sparse Table
    }
    SparseTable(int n_, TT * a) : n(n_){
        pot2.resize(n+10);
        pot2[1] = 0;
        for(int i=2; i<=n; i++) pot2[i] = pot2[i>>1] + 1;
        tab.resize(n+10,vector<TT>(pot2[n]+1));
        TETO = pot2[n]+1;
        build(a); // build da Sparse Table
    }

    ~SparseTable(){tab.clear(); pot2.clear();}
```

```

    void init(int n_new,vector<TT> & a){
        n = n_new;
        TETO = pot[n] + 1;
        build(a);
    }
    void init(int n_new,TT * a){
        n = n_new;
        TETO = pot[n] + 1;
        build(a);
    }

    void build(vector<TT> & a){
        for(int i=1; i<=n; i++) tab[i][0] = a[i];
        for(int j=1; j<TETO; j++) for(int i=0; i+(1<<j)-1 <= n; i++) tab[i][j] = f(tab[i][j-1],tab[i+(1<<(j-1))][j-1]);
    }
    void build(TT * a){
        for(int i=1; i<=n; i++) tab[i][0] = a[i];
        for(int j=1; j<TETO; j++) for(int i=0; i+(1<<j)-1 <= n; i++) tab[i][j] = f(tab[i][j-1],tab[i+(1<<(j-1))][j-1]);
    }

    // operacao da sparse table
    TT f(TT x, TT y){
        return ;
    }

    TT query(int l,int r){
        if(l > r) return off;
        TT ans = off;
        for(int j=TETO-1; j>=0; j--){
            if(l+(1<<j)-1 <= r){
                ans = f(ans, tab[l][j]);
                l +=(1<<j);
            }
        }
        return ans;
    }
    // query idempotente
    TT query_id(int l,int r){
        if(l > r) return off;
        int diff = pot2[r-l+1];
        return f(tab[l][diff],tab[r-(1<<diff)+1][diff]);
    }
};
```

3.3.2 Disjoint Sparse Table

All queries in $O(1)$.

```
/*
    Indexado de 1.
    Responde uma operacao num subarray, a operacao precisa ser associativa e de preferencia ter uma identidade.
    Init: O(N*log(N)).
    Query: O(1)

    Alteracoes:
    Funcao f(), valor off
*/
template <class TT = ll>
struct DST{
    int n; // tamanho do vetor original
    vector<vector<TT>> tab; // disjoint sparse table
    const TT off = ; // identidade
```

```
    int TETO_DST; // menor potencia tal que 2^TETO_DST >= n
    vector<int> pot2; // log2 de cada numero

    DST(int n_) : n(n_){
        TETO_DST = 0;
        int lim = 1;
        while(lim < n) lim <<= 1, TETO_DST++;
        pot2.resize(lim);
        pot2[1] = 0;
        for(int i=2; i<lim; i++) pot2[i] = pot2[i>>1] + 1;
        tab.resize(TETO_DST+1,vector<TT>(lim,off));
    }
    DST(int n_, vector<TT> & a) : n(n_){
        TETO_DST = 0;
        int lim = 1;
        while(lim < n) lim <<= 1, TETO_DST++;
        pot2.resize(lim);
        pot2[1] = 0;
        for(int i=2; i<lim; i++) pot2[i] = pot2[i>>1] + 1;
        tab.resize(TETO_DST+1,vector<TT>(lim,off));
        init(a);
    }

    void init(vector<TT> & a){
        int lim = 1<<TETO_DST;
        // qtd de blocos
        for(int i=0; i<TETO_DST; i++){
            int tam = (1<<(TETO_DST-i)); // tamanho do bloco
            // bloco atual
            for(int j=0; j<(1<<i); j++){
                int init = j*tam;
                int end = (j+1)*tam-1;
                int mid = (init + end)>>1;

                // primeira metade eh um sufixo
                tab[i][mid] = (mid+1 <= n ? a[mid+1] : off); // vetor de entrada eh indexado de 1
                for(int k=mid-1; k>=init; k--) tab[i][k] = f((k+1 <= n ? a[k+1] : off),tab[i][k+1]);

                // segunda metade eh um prefixo
                tab[i][mid+1] = (mid+2 <= n ? a[mid+2] : off); // vetor de entrada eh indexado de 1
                for(int k=mid+2; k<=end; k++) tab[i][k] = f((k+1 <= n ? a[k+1] : off),tab[i][k-1]);
            }
        }
        for(int j=0; j<lim; j++) tab[TETO_DST][j] = (j+1 <= n ? a[j+1] : off);
    }

    // operacao da DST
    TT f(TT x, TT y){
        return ;
    }

    // query inclusiva nos ranges
    TT query(int l,int r){
        if(l > r) return off;
        l--, r--;
        if(l == r) return tab[TETO_DST][l];
        int level = TETO_DST - pot2[l^r] - 1;
        return f(tab[level][l],tab[level][r]);
    }
};
```

3.4 SQRT Decomposition

3.4.1 MO

Nada de novo sob o sol, lembre q na aba de DSU persistente tem uma aplicação bem hard.

```
/*
    Indexado de 1 (os l e r). O K = N/sqrt(Q), tal que N eh o maior
   range dos valores de l e r e Q o numero de queries

    Alteracoes:

    Ver como calcula a query
    Ler as queries
    Ver como imprime a resposta
    Ver as funcoes de add e rmv do mo
*/
template <class TT = int>
struct Mo{
    const int K = 450;

    struct query{
        int idx,l,r;

        bool operator <(const query &o){
            if(l/K == o.l/K){
                if((l/K)&1) return r>o.r;
                return r<o.r;
            }
            return l/K < o.l/K;
        }
    };

  vector<query> q;
  vector<TT> ans;

    // qtd de queries
    Mo(int n_q){
        q.resize(n_q);
    }


    // adiciona na posicao x
  void add(int x){

    }

    // remove na posicao x
  void rmv(int x){

    }

    // calcula a resposta pra query
  TT calc(){

    }

  void solve(){
    sort(q.begin(),q.end());
    ans.resize(q.size());
    int i,j;
    i=1;
    j=0;
        for(auto [idx,l,r] : q){
      while(j < r){
        add(++j);
      }
```

```
      while(i > l){
        add(--i);
      }

      while(j > r){
        rmv(j--);
      }
      while(i < l){
        rmv(i++);
      }

      ans[idx]=calc();
    }
  }

    // ler as queries
    void ler(){
        for(int i=0; i<q.size(); i++) {
            cin >> q[i].l >> q[i].r;
            q[i].idx=i;
        }
    }

  void show(){
        // imprime a resposta
  }
};
```

3.4.2 MO com updates

Um ponto importante é que se nós fizermos aquele esquema de decompor a query de path em arvore em uma query no range do euler tour, também podemos usar mo.

```
const int N = 100005
int n, q;
int v[N];
int vv[N];

namespace mo
{
  struct query
  {
    int idx, l, r, t;
  };
  struct update
  {
    int i, prevx, x;
  };

  int block;
  vector<query> queries;
  vector<update> updates;
  vector<int> ans;

  bool cmp(query x, query y)
  {
    if (x.l / block != y.l / block)
      return x.l / block < y.l / block;
    if (x.r / block != y.r / block)
      return x.r / block < y.r / block;
    return x.t < y.t;
  }
  void run()
  {
    block = 3153; // (2 * n) ^ 0.666
    sort(queries.begin(), queries.end(), cmp);
```

```
    ans.resize(queries.size());
    int cl = 0, cr = -1, sum = 0, t = 0;
    auto add = [&](int x)
    {
      sum += x;
    };
    auto rem = [&](int x)
    {
      sum -= x;
    };
    for (int i = 0; i < queries.size(); i++)
    {
      while (cl > queries[i].l)
      {
        cl--;
        add(v[cl]);
      }
      while (cr < queries[i].r)
      {
        cr++;
        add(v[cr]);
      }
      while (cl < queries[i].l)
      {
        rem(v[cl]);
        cl++;
      }
      while (cr > queries[i].r)
      {
        rem(v[cr]);
        cr--;
      }
      while (t > queries[i].t)
      {
        t--;
        if (queries[i].l <= updates[t].i && queries[i].r >= updates[t].i
   )
        {
          rem(updates[t].x);
          add(updates[t].prevx);
        }
        v[updates[t].i] = updates[t].prevx;
      }
      while (t < queries[i].t)
      {
        if (queries[i].l <= updates[t].i && queries[i].r >= updates[t].i
   )
        {
          rem(updates[t].prevx);
          add(updates[t].x);
        }
        v[updates[t].i] = updates[t].x;
        t++;
      }
      ans[queries[i].idx] = sum;
    }
  }
}
signed main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  cin >> n >> q;
  for (int i = 0; i < n; i++)
  {
    cin >> v[i];
```

```
    vv[i] = v[i];
  }
  for (int i = 0; i < q; i++)
  {
    int type;
    cin >> type;
    if (type == 1)
    {
      mo::update curr;
      cin >> curr.i >> curr.x;
      curr.prevx = vv[curr.i];
      vv[curr.i] = curr.x;
      mo::updates.pb(curr);
    }
    else
    {
      mo::query curr;
      cin >> curr.l >> curr.r;
      curr.r--;
      curr.idx = mo::queries.size();
      curr.t = mo::updates.size();
      mo::queries.pb(curr);
    }
  }
  mo::run();
  for (auto const &i : mo::ans)
    cout << i << endl;
}
```

3.4.3 Blocking

We partition the array into blocks of size $\texttt{block\_size} = \lceil\sqrt{N}\rceil$. Each block stores the sum of elements within it, and allows for the creation of corresponding update and query operations.

Update Queries: $\mathcal{O}(1)$

To update an element at location x, first find the corresponding block using the formula $\frac{x}{\texttt{block\_size}}$.

Then, apply the corresponding difference between the element currently stored at x and the element we want to change it to.

Sum Queries: $\mathcal{O}(\sqrt{N})$

To perform a sum query from $[0 \ldots r]$, calculate

$$\sum_{i=0}^{R-1} \texttt{blocks}[i] + \sum_{R\cdot\texttt{block\_size}}^{r} \texttt{nums}[i]$$

where $\texttt{blocks}[i]$ represents the total sum of the i-th block, the i-th block represents the sum of the elements from the range $[i \cdot \texttt{block\_size}, (i+1) \cdot \texttt{block\_size})$, and $R = \left\lceil \frac{r}{\texttt{block\_size}} \right\rceil$.

Finally, $\sum_{i=l}^{r} \texttt{nums}[i]$ is the difference between the two sums $\sum_{i=0}^{r} \texttt{nums}[i]$ and $\sum_{i=0}^{l-1} \texttt{nums}[i]$, which each are calculated in $\mathcal{O}(\sqrt{N})$.

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

struct Sqrt {
  int block_size;
  vector<int> nums;
  vector<long long> blocks;
  Sqrt(int sqrtn, vector<int> &arr) : block_size(sqrtn), blocks(sqrtn,
   0) {
    nums = arr;
    for (int i = 0; i < nums.size(); i++) {
      blocks[i / block_size] += nums[i];
    }
  }

  /** O(1) update to set nums[x] to v */
  void update(int x, int v) {
    blocks[x / block_size] -= nums[x];
```

```
    nums[x] = v;
    blocks[x / block_size] += nums[x];
  }

  /** O(sqrt(n)) query for sum of [0, r) */
  long long query(int r) {
    long long res = 0;
    for (int i = 0; i < r / block_size; i++) { res += blocks[i]; }
    for (int i = (r / block_size) * block_size; i < r; i++) {
      res += nums[i];
    }
    return res;
  }

  /** O(sqrt(n)) query for sum of [l, r) */
  long long query(int l, int r) { return query(r) - query(l - 1); }
};

int main() {
  int n, q;
  cin >> n >> q;

  vector<int> arr(n);
  for (int i = 0; i < n; i++) { cin >> arr[i]; }
  Sqrt sq((int)ceil(sqrt(n)), arr);

  for (int i = 0; i < q; i++) {
    int t, l, r;
    cin >> t >> l >> r;
    if (t == 1) {
      sq.update(l - 1, r);
    } else {
      cout << sq.query(l, r) << "\n";
    }
  }
}
```

3.4.4 Blocking com cyclic shift

Legendary Huron is a fan of beautiful fences. He defines the beauty of a sequence (l,r) is $max_{l \leq i \leq j \leq r}(a[j] - a[i])$. The queries are:

Given three integers l, r, k, he will repaint the planks in the range $[l, r]$. With $a[l] = k$, $a[l+1] = k+1$, $\ldots$, $a[r-1] = k + (r-l)$, $a[r] = k + (r-l+1)$;

Given two integers l and r, returns the beauty.

```
// Example of sqrt decomposition
const int N = 400100;
const int K = 500;

int ans[K][N/K], bmi[K][N/K], bmx[K][N/K];
int pre[N/K],lazy[N/K], ami[N/K], amx[N/K];
int a[N], b[N];

int calc(int ini,int k)
{
    ini%=K;
    k%=K;
    k -= ini;
    if(k<0)
        k+=K;
    return k;
}

void updt(int id)
{
    int j = id*K;
```

```
    int mi = a[j];
    pre[id] = 0;
    amx[id] = a[j];
    while(j/K == id)
    {
        pre[id] = max(pre[id], a[j] - mi);
        mi = min(mi, a[j]);
        ami[id] = mi;
        amx[id] = max(amx[id], a[j]);
        j++;
    }
}

void push(int id)
{
    // cout << id << ' ' << lazy[id] << '\n';
    if(lazy[id] == -1)
        return;
    int ini = id*K;
    rep(i, 0, K)
    {
        a[ini+i] = b[lazy[id]+i];
    }
    lazy[id] = -1;
}

void build(int n)
{
    int mi;
    rep(i, 0, K)
    {
        rep(j, i, n)
        {
            if((j-i)%K == 0)
            {
                mi = b[j];
                ans[i][(j-i)/K] = 0;
                bmx[i][(j-i)/K] = b[j];
            }
            ans[i][(j-i)/K] = max(ans[i][(j-i)/K], b[j] - mi);
            mi = min(mi, b[j]);
            bmi[i][(j-i)/K] = mi;
            bmx[i][(j-i)/K] = max(bmx[i][(j-i)/K], b[j]);
        }
    }
    rep(j, 0, n)
    {
        if((j)%K == 0)
        {
            mi = a[j];
            pre[(j)/K] = 0;
            lazy[j/K] = -1;
            amx[j/K] = a[j];
        }
        pre[(j)/K] = max(pre[(j)/K], a[j] - mi);
        mi = min(mi, a[j]);
        ami[j/K] = mi;
        amx[j/K] = max(amx[j/K], a[j]);
    }
    // rep(j, 0, K)
    //     cout << j <<' ' << pre[j] << '\n';
}

int ask(int l, int r)
{
    // if(l%K != 0)
```

```
    // {
        push(l/K);
        updt(l/K);
    // }
    int rs = 0, mi = a[l];
    // cout << l << ' ' << l%K << '\n';
    while(l%K != 0 && l <= r)
    {
        rs = max(rs, a[l]-mi);
        mi = min(mi, a[l++]);
    }
    while(l+K-1 <= r)
    {
        // cout << "OPA " << l << ' ' << pre[l/K] << ' ' << amx[l/K] << "\n";
        rs = max(rs, pre[l/K]);
        rs = max(rs, amx[l/K]-mi);
        mi = min(mi, ami[l/K]);
        l+=K;
    }
    // cout << l << ' ' << r << ' ' << rs << "<<<\n";
    if(l <= r){
        push(l/K);
        updt(l/K);
        // cout << rs << "<<<<\n";
        while(l <= r)
        {
            // cout << a[l] << ' ';
            rs = max(rs, a[l]-mi);
            mi = min(mi, a[l++]);
        }
        // cout << '\n';
    }
    return rs;
}

void updt(int l, int r, int k)
{
    int id = calc(l, k);
    if(l%K != 0)
    {
        push(l/K);
        while(l%K != 0 && l <= r)
        {
            a[l++] = b[k++];
        }
        updt((l-1)/K);
    }
    while(l+K-1 <= r)
    {
        // cout << l << ' ' << k << "<<<\n";
        // cout << id << ' ' << pre[l/K] << ' ' << ans[id][(k-id)/K] << '\n';
        pre[l/K] = ans[id][(k-id)/K];
        amx[l/K] = bmx[id][(k-id)/K];
        ami[l/K] = bmi[id][(k-id)/K];
        lazy[l/K] = k;
        l+=K;
        k+=K;
    }
    if(l <= r)
    {
        // cout << l << ' ' << k << "<<<<<<<\n";
        push(r/K);
        while(l <= r)
            a[l++] = b[k++];
```

```
        updt(r/K);

    }
}
```

3.4.5 Batching

Maintain a "buffer"of the latest updates (up to $\sqrt{N}$). The answer for each sum query can be calculated with prefix sums and by examining each update within the buffer. When the buffer gets too large ($\geq \sqrt{N}$), clear it and recalculate prefix sums.

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

int n, q;
vector<int> arr;
vector<long long> prefix;

/** Build the prefix array for arr */
void build() {
  prefix[0] = 0;
  for (int i = 1; i <= n; i++) { prefix[i] = prefix[i - 1] + arr[i - 1];
    }
}

int main() {
  cin >> n >> q;
  arr.resize(n);
  for (int i = 0; i < n; i++) { cin >> arr[i]; }
  prefix.assign(n + 1, 0);
  build();

  vector<pair<int, int>> updates;
  for (int i = 0; i < q; i++) {
    int type, a, b;
    cin >> type >> a >> b;
    if (type == 1) {
      a--;
      updates.push_back({a, b - arr[a]});
      arr[a] = b;
    } else {
      long long ans = prefix[b] - prefix[a - 1];
      a--, b--;
      for (const auto &[idx, val] : updates) {
        if (a <= idx && idx <= b) { ans += val; }
      }
      cout << ans << "\n";
    }

    // rebuild the prefix array once the buffer gets to sqrt(n)
    if (updates.size() * updates.size() >= n) {
      updates.clear();
      build();
    }
  }
}
```

3.5 Treap

Basicamente é uma estrutura que dá pra fazer um monte de coisa com "pouco"código. A operação reverse é a + roubada.

```
#define tipo int
mt19937 mt_rand(time(0));
struct Node{
```

```
    int p,cnt;
    tipo value;
    bool rev;
    struct Node * l;
    struct Node * r;
    node() { }
    node(tipo value) : value(value), p(mt_rand()),cnt(1),rev(false), l(
   NULL), r(NULL) { }

};
typedef Node * pnode;

int cnt (pnode t) {
    return t ? t->cnt : 0;
}

void upd_cnt (pnode t) {
    if (t){

        t->cnt = 1 + cnt(t->l) + cnt (t->r);
    }
}

void push (pnode it) {
    if (it && it->rev) {
        it->rev = false;
        swap (it->l, it->r);
        if (it->l)  it->l->rev ^= true;
        if (it->r)  it->r->rev ^= true;
        //upd_cnt(it);
    }
}

void split(pnode t, pnode &l, pnode &r,int key,int add = 0){

    if(!t){
        return void( l = r = NULL );
    }
    push(t);
    int cur_key=add+cnt(t->l);
    if(key<=cur_key){
        split(t->l,l,t->l,key,add);
        r=t;
    }else{
        split(t->r,t->r,r,key,add+1+cnt(t->l));
        l=t;
    }
    upd_cnt (t);
}

void merge(pnode &t,pnode l, pnode r){
    push(l);
    push(r);
    if (!l || !r)
        t = l ? l : r;
    else if(l->p>r->p){
        merge(l->r,l->r,r),t=l;
    }else{
        merge(r->l,l,r->l),t=r;
    }
    upd_cnt (t);
}

void insert(pnode &t,pnode n,int key){
    pnode t1,t2;
    split(t,t1,t2,key);
```

```
    // cout<<"oi\n";
    merge(t1,t1,n);
    merge(t,t1,t2);
}

void reverse (pnode t, int l, int r) {
    if(l>=r)
        return;
    pnode t1,t2,t3;
    split (t, t1, t2, l);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}
void output (pnode t) {
    if (!t)  return;
    // push (t);
    output (t->l);
    cout<<t->value<<" ";
    output (t->r);
}
void clr (pnode &t) {
    if (!t)  return;
    clr (t->l);
    clr (t->r);
    delete t;
    t=NULL;
    // cout<<t<<"\n";
}

void erase(pnode &t,int key)
{
    pnode t1,t2,t3;
    split(t,t1,t2,key-1);
    split(t2,t2,t3,key);
    merge(t,t1,t3);
}
```

3.6 DSU

There are multiples implementations here, with different focuses.

3.6.1 Persistent DSU

First one: Just an DSU such that u can ask who was my parent in time t.

```
int pai[N];
int rk[N];
int tmp[N];

int findx(int x, int t)
{
  if(tmp[x] > t || tmp[x] == 0)
    return x;
  return findx(pai[x],t);
}
int cs = 0;
void merge(int a, int b)
{
  a = findx(a,cs);
  b = findx(b,cs);
  if(a != b)
  {
    if(rk[a] < rk[b])
      swap(a,b);
```

```
    rk[a] = max(rk[a],rk[b]+1);
    pai[b] = a;
    tmp[b] = ++cs;
    tmp[a] = 0;
  }
}
```

3.6.2 DSU with rollback

Second: an DSU such that u roll back last operation.

```

int pai[N];
vector<int> rk[N];

int findx(int x)
{
  if(pai[x] == x)
    return x;
  return findx(pai[x]);
}
int cs = 0;
vector<pii> rmv;
void merge(int a, int b)
{
  a = findx(a);
  b = findx(b);
  if(a != b)
  {
    if(rk[a].back() < rk[b].back())
      swap(a,b);
    rk[a].pb(max(rk[a].back(),rk[b].back()+1));
    pai[b] = a;
    rmv.eb(a,b);
        cs++;
  }
}

void roll_back()
{
  pai[rmv.back().sd] = rmv.back().sd;
  rk[rmv.back().ft].pop_back();
  rmv.pop_back();
  cs--;
}
```

Now, some crazy applications.

3.6.3 Connected Components With Segments

First one, given some set of edges, answer multiples queries. For each query answer how many connected components are there, if u use the edges in range (l,r).

The solution does an approach with mo and dsu with rollback. The main ideia is:

First solve for those such that $(r - l + 1) < \sqrt(N)$. Just brute with DSU. After it, lets split the solution in blocks of l. For each block u sort the queries by r. Lets say u are in first block and every l lies in range $1 - -\sqrt{N}$. The solution does the following steps:

- Add every edge in range $\sqrt{N} + 1 - r$.
- Add every edge in range $l - \sqrt{N}$.
- get answer for this segment.
- rollback the edges in range $l - \sqrt{N}$.
- go to the next segment.

The first step can be slow, but for each block the amortized time will be N. The other steps, take $\sqrt{N}$ time.

```
#include<bits/stdc++.h>
using namespace std;
//template
const int N = 50100;
const int K = 250;

//DSU with Rollback

vector<pii> ed;
namespace mo
{
    struct query
    {
        int idx, l, r;
    };
    vector<query> queries;
    vector<int> ans;

    bool cmp(query x, query y)
    {
        if (x.l / K != y.l / K)
            return x.l / K < y.l / K;
        return x.r < y.r;
    }
    void run(int n)
    {
    dsu::init(n);
        sort(queries.begin(), queries.end(), cmp);
        ans.resize(queries.size());
        int lim, cr;
        int check, at = -1;
        for (int i = 0; i < queries.size(); i++)
        {
      int al = queries[i].l;
      int ar = queries[i].r;
      if(al/K != at)
      {
        while(dsu::cs)
          dsu::roll_back();
        at = al/K;
        lim = min<int>(at*K + K-1,ed.size()-1);
        cr = lim;
      }
      if(ar/K == at)
      {
        check = 0;
        rep(j, al, ar+1)
          dsu::merge(ed[j].ft,ed[j].sd);
      }else
      {
        while(cr < ar)
        {
          cr++;
          dsu::merge(ed[cr].ft,ed[cr].sd);
        }
        check = dsu::cs;
        rep(j, al,lim+1)
          dsu::merge(ed[j].ft,ed[j].sd);

      }
            ans[queries[i].idx] = n-dsu::cs;
      while(dsu::cs != check)
        dsu::roll_back();
        }
    }
```

```
}


void test()
{
  int n, m;
  cin >> n >> m;
  rep(i, 0, m)
  {
    int u, v;
    cin >> u >> v;
    ed.eb(u,v);
  }
  int k;
  cin >> k;
  rep(i, 0, k)
  {
    mo::query curr;
    cin >> curr.l >> curr.r;
    curr.l--;
    curr.r--;
    curr.idx = i;
    mo::queries.pb(curr);
  }
  mo::run(n);
  rep(i, 0, k)
    cout << mo::ans[i] << '\n';
}


int32_t main()
{
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  int t = 1;
  // cin >> t;
  while(t--)
    test();
  return 0;
}
```

3.6.4 Dynamic Connectivity Offline

Last application. solve those queries:

- add an edge $u - v$
- remove an edge $u - v$
- answer the number off connected components.

The main idea is to define an id to each query of type 3 and define ranges off "activation"suppose u add an edge $u - v$ and after it u remove. This will create a range $l - r$ which represents which queries off type 3 this edge is active.

After it we will do a "segment tree"alike solution.

The main idea is to keep the segments which are relevant to this part of the queries and just add the edge when it cover all the ids in this segment. after it u always roll back the edges added in this level. it is easy to see (maybe it is not) that each segment will have a complexity close to a query in a segment tree.

```
#include<bits/stdc++.h>
using namespace std;
//template
const int N = 300100;
//DSU with Rollback
map<pii,int> id;
struct query
```

```
{
  int l, r, u, v;
};
int ans[N], n;

vector<query> queries;
int sum = 0;
// vector<pair<pii,vector<int>>>
void solve(vector<int> &nw, int l, int r)
{
  if(l > r)
    return;
  // sum++;
  int check = dsu::cs;
  if(l == r)
  {
    // cout << l << ' ' << r << "<<<\n";

    for(int i:nw)
      dsu::merge(queries[i].u,queries[i].v);
    ans[l] = n-dsu::cs;

  }else
  {
    int mid = (l+r) >> 1;
    vector<int> lef,rig;
    // cout << l << ' ' << r << "<<<\n";
    for(int i:nw)
    {
      if(queries[i].l <=l && queries[i].r >= r){
        // cout << queries[i].l << ' ' << queries[i].r << ' ' << queries
[i].u << ' ' << queries[i].v << '\n';
        dsu::merge(queries[i].u,queries[i].v);
      }
      else {
        if(queries[i].l <= mid && queries[i].r >= l)
          lef.pb(i);
        if(queries[i].r > mid && queries[i].l <= r)
          rig.pb(i);
      }
    }
    // nw.clear();
    solve(lef,l,mid);
    solve(rig,mid+1,r);
  }
  while(dsu::cs != check)
      dsu::roll_back();
}
vector<int> nw;
void test()
{
  int m;
  cin >> n >> m;
  dsu::init(n);
  int pass = 1;
  rep(i, 1, m+1)
  {
    char ch;
    cin >> ch;
    if(ch == '?')
    {
      pass++;
    }else if(ch == '+')
    {
      int u, v;
      cin >> u >> v;
```

```
      if(u > v)
        swap(u,v);
      id[mp(u,v)] = pass;
    }else
    {
      query at;
      cin >> at.u >> at.v;
      if(at.u > at.v)
        swap(at.u,at.v);
      at.l = id[mp(at.u,at.v)];
      at.r = pass-1;
      id[mp(at.u,at.v)] = 0;
      if(at.l <= at.r)
      {
        nw.pb(queries.size());
        queries.pb(at);
      }
    }
  }
  for(auto [ed,x]:id)
  {
    if(!x)
      continue;
    query at;
    at.u = ed.ft;
    at.v = ed.sd;
    at.l = x;
    at.r = pass-1;
    if(at.l <= at.r)
    {
      nw.pb(queries.size());
      queries.pb(at);
    }
  }
  // for(auto at:queries)
  //  cout << at.l << ' ' << at.r << ' ' << at.u << ' ' << at.v << '\n';
  solve(nw,1,pass-1);
  rep(i, 1, pass)
    if(ans[i] != 0)
      cout << ans[i] << '\n';

  // cout << sum << "<<\n";
}


int32_t main()
{
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  int t = 1;
  // cin >> t;
  while(t--)
    test();
  return 0;
}
```

3.6.5 DSU with Small to Large

When you have DSU and besides the representative thing, you may have other information about the groups, useful to solve queries about unions.

```
/*
    Indexado em 1

    Alteracoes:

```

```
    Ver se precisa adicionar coisa no gp do smol
*/
struct DSU{
    // o que cada grupo carrega de informacao
    struct gp{
        // se precisa adicionar coisa a mais
        int tam = 0;

        // inicializar o smol
        void init(int x){
            // se precisa inicializar coisa a mais, passa por parametro
            tam=1;
        }
        void clear(){
            tam = 0;
        }
        ~gp(){}
    };

    vector<int> repre;
    vector<gp> smol;
    int n;

    // inicializar passando a qtd de vertices, grupos iniciais, se t = 0 entao so aloca a memoria sem inicializar
    DSU (int n_,int t = 1) : n(n_){
        repre.resize(n+10);
        smol.resize(n+10);
        if(t){
            for(int i=1; i<=n; i++){
                repre[i]=i;
                smol[i].init(i);
            }
        }
    }
    ~DSU(){
        repre.clear();
        smol.clear();
    }
    void init(int n){
        this->n = n;
        for(int i=1; i<=n; i++){
            repre[i] = i;
            smol[i].init(i);
        }
    }
    void clear(){
        for(int i=1; i<=n; i++){
            smol[i].clear();
        }
    }


    // achar o representante do u
    int rep(int u){
        return repre[u] = (repre[u] == u ? u : rep(repre[u]));
    }
    // pegar o smol do cara u
    gp & smoll(int u){
        return smol[rep(u)];
    }

    // unir u e v
    void unite(int u,int v,int t){
        u=rep(u);
        v=rep(v);
```

```
        if(u == v) return;

        auto &x=smol[u];
        auto &y=smol[v];
        if(y.tam > x.tam){
            unite(v,u,t);
            return;
        }

        // da merge nos smols
        merge(x,y,t);
        repre[v]=u;
    }

    // fazer o merge de 2 grupos
    void merge(gp &x, gp &y,int t){
        // faz o merge se precisar
        x.tam += y.tam;
        y.clear();
    }
};
```

Capítulo IV

Segment Tree

Era uma seção, mas tem tanta coisa que virou um capitulo.

4.1 Sem/Com Lazy

```
/*
    Indexado de 1.
    Responde uma operacao num subarray, suporta update em range
    Init: O(4*N).
    Query: O(4*log(N))
    Update: O(4*log(N))
*/
template <class TT = int>
struct Seg{
    // inicializar so o tamanho da seg, n fazer o build
    Seg(int n_) : n(n_){
        seg.resize(n<<2);
        lazy.resize(n<<2);
    }

    // fazer o build da estrutura
    void init(int n_new, TT * a){
        n = n_new;
        vec = a;
        build(1,1,n);
    }

    // o que vai ter dentro do no de cada seg
    struct node{

        bool operator ==(const node &ot)const{
            return true;
        }
    };
    // o que vai ter dentro do no de cada lazy
    struct sono{

        bool operator ==(const sono &ot)const{
            return true;
        }
    };

    TT * vec;
    int n;
    // no nulo
    const node off = {};
    // lazy nula
    const sono off_lazy = {};
    vector<node> seg;
    vector<sono> lazy;
    // operacao de unir dois nos
    node merge(node x, node y){
        if(x == off) return y;
```

```
        if(y == off) return x;

    }

    // coisar a lazy pra baixo
    void push(int u,int tl,int tr){
        if(tl == tr || lazy[u] == off_lazy) return;
        // atualizar os filhos

        // atualizar as lazies dos filhos

        lazy[u] = off_lazy;
    }

    // inicializar a seg
    void build(int u,int tl,int tr){
        if(tl == tr){
            // inicializar os caras bases
            seg[u] = {};
            lazy[u] = off_lazy;
            return;
        }
        int tmid = tl + tr;
        tmid >>= 1;
        build(lef(u), tl, tmid);
        build(rig(u), tmid+1, tr);
        seg[u] = merge(seg[lef(u)], seg[rig(u)]);
        lazy[u] = off_lazy;
    }

    // consulta em range
    node query_(int u,int tl,int tr,int l, int r){
        if(l > r) return off;
        if(tl == l && tr == r) return seg[u];
        push(u, tl, tr);
        int tmid= tl + tr;
        tmid >>= 1;
        return merge(query_(lef(u), tl, tmid, l, min(tmid,r)), query_(
  rig(u), tmid+1, tr, max(tmid+1,l), r));
    }
    node query(int l, int r){
        return query_(1, 1, n, l, r);
    }

    // update em range
    void update_(int u, int tl, int tr, int l, int r, TT x){
        if(l > r) return;
        if(tl == l && tr == r){
            // atualizar a seg e o lazy

            return;
        }
        push(u, tl, tr);
        int tmid = tl + tr;
        tmid >>= 1;
        update_(lef(u), tl, tmid, l, min(tmid,r), x);
        update_(rig(u), tmid+1, tr, max(tmid+1,l), r, x);
        seg[u] = merge(seg[lef(u)], seg[rig(u)]);
    }
    void update(int l, int r, TT x){
        update_(1, 1, n, l, r, x);
    }
};
```

4.2 Implicita

Essa eu roubei do tiagodfs XD

```
// SegTree Implicita O(nlogMAX)

struct node{
    int val;
    int l, r;
    node(int a=0, int b=0, int c=0){
        l=a;r=b;val=c;
    }
};

int idx=2; // 1-> root / 0-> zero element
node t[8600010];
int N;

int merge(int a, int b){
    return a + b;
}

void update(int pos, int x, int i=1, int j=N, int no=1){
    if(i==j){
        t[no].val+=x;
        return;
    }
    int meio = (i+j)/2;

    if(pos<=meio){
        if(t[no].l==0) t[no].l=idx++;
        update(pos, x, i, meio, t[no].l);
    }
    else{
        if(t[no].r==0) t[no].r=idx++;
        update(pos, x, meio+1, j, t[no].r);
    }

    t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
}

int query(int A, int B, int i=1, int j=N, int no=1){
    if(B<i or j<A)
        return 0;
    if(A<=i and j<=B)
        return t[no].val;

    int mid = (i+j)/2;

    int ansl = 0, ansr = 0;

    if(t[no].l!=0) ansl = query(A, B, i, mid, t[no].l);
    if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no].r);

    return merge(ansl, ansr);
}
```

4.3 Implicita com lazy

```
struct node{
    pll val;
    ll lazy;
    ll l, r;
    node(){
        l=-1;r=-1;val={0,0};lazy=0;
```

```
    }
};

node tree[40*MAX];
int id = 2;
ll N=1e9+10;

pll merge(pll A, pll B){
    if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
    return (A.ff<B.ff ? A:B);
}

void prop(ll l, ll r, int no){
    ll mid = (l+r)/2;
    if(l!=r){
        if(tree[no].l==-1){
            tree[no].l = id++;
            tree[tree[no].l].val = {0, mid-l+1};
        }
        if(tree[no].r==-1){
            tree[no].r = id++;
            tree[tree[no].r].val = {0, r-(mid+1)+1};
        }
        tree[tree[no].l].lazy += tree[no].lazy;
        tree[tree[no].r].lazy += tree[no].lazy;
    }
    tree[no].val.ff += tree[no].lazy;
    tree[no].lazy=0;
}

void update(int a, int b, int x, ll l=0, ll r=2*N, ll no=1){
    prop(l, r, no);
    if(a<=l and r<=b){
        tree[no].lazy += x;
        prop(l, r, no);
        return;
    }
    if(r<a or b<l) return;
    int m = (l+r)/2;
    update(a, b, x, l, m, tree[no].l);
    update(a, b, x, m+1, r, tree[no].r);

    tree[no].val = merge(tree[tree[no].l].val, tree[tree[no].r].val);
}

pll query(int a, int b, int l=0, int r=2*N, int no=1){
    prop(l, r, no);
    if(a<=l and r<=b) return tree[no].val;
    if(r<a or b<l) return {INF, 0};
    int m = (l+r)/2;
    int left = tree[no].l, right = tree[no].r;

    return tree[no].val = merge(query(a, b, l, m, left),
                                query(a, b, m+1, r, right));
}
```

4.4 Persistente

Solution for Spoj Count on a Tree, Given a tree there are Q queries where you need to find the kth minimum weigth in a path between u and v, it also needs the lca and lca's father between u and v. Main idea is to build a segment tree for each node with a dfs starting from the root.

Queries in $O(log(n))$

Pra iniciar da $build(1, 1, n)$. todo update tem que ter uma head nova. o last representa o ultimo id livre.

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

#define lef(x) (tree[x].l)
#define rig(x) (tree[x].r)

const int N = 200100;
const ll emp = 0; //NULL VALUE

struct node
{
    int l, r;
    ll v;
    node(): l(-1), r(-1), v(0){};
    node(int l, int r, ll v): l(l), r(r), v(v){};
};

ll a[N];
node tree[N << 6];
int last = 2;
int head[N];
ll f(ll a, ll b){return a+b;}//function

void build(int p, int tl, int tr)
{
    if(tl == tr)
    {
        tree[p].v = a[tl];
    return;
    }

  int tmid = (tl + tr) >> 1;
    tree[p].l = last++;
    tree[p].r = last++;
  build(lef(p), tl, tmid);
  build(rig(p), tmid+1, tr);
  tree[p].v = f(tree[lef(p)].v, tree[rig(p)].v);

}

void updt(int p,int np, int tl, int tr, int k, ll v)
{
    tree[np] = tree[p];
    if(tl == tr){
        tree[np].v = v;
        return;
    }
    int tmid = (tl+tr) >> 1;
    if (tmid < k){
        tree[np].r = last++;
        updt(rig(p), rig(np), tmid+1, tr, k, v);
    }
    else{
        tree[np].l = last++;
        updt(lef(p), lef(np), tl, tmid, k, v);
    }
    tree[np].v = f(tree[lef(np)].v, tree[rig(np)].v);

}

ll query(int p, int tl, int tr, int l, int r)
{
    if(l > r)
        return emp;
```

```
    if(tl == l && tr == r)
        return tree[p].v;
    int tmid = (tl+tr) >> 1;
    return f(query(lef(p), tl, tmid, l, min(r,tmid)),
        query(rig(p), tmid+1, tr, max(l,tmid+1), r));
}
int walk(int u,int v,int lc,int plc,int d,int tl,int tr){
  if(tl==tr)
    return tl;
  int at=0;
  int mid=(tl+tr)/2;
  at+=tree[lef(u)].v;
  at+=tree[lef(v)].v;
  at-=tree[lef(lc)].v;
  at-=tree[lef(plc)].v;
  // cout<<at<<" "<<tl<<" "<<tr<<"\n";
  if(at>=d)
    return walk(tree[u].l,tree[v].l,tree[lc].l,tree[plc].l,d,tl,mid);
  if(at<d)
    return walk(tree[u].r,tree[v].r,tree[lc].r,tree[plc].r,d-at,mid+1,tr
   );
}
```

4.5 update é uma PA

Nessa seg o update é uma PA que começa com V na posição L e vira $V+R-L+1$ na posição R. A sacada é que da pra juntar multiplas PAs só aumento o coeficiente r, lembrando que:

$an = a_1 + r \cdot (n-1)$

$sn = (a_1 + a_n) \cdot (n)/2$

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

#define lef(x) (x << 1)
#define rig(x) (lef(x) | 1)

const int N = 200100;
const ll emp = 0; //NULL VALUE

ll a[N],tree[N << 2];
pair<ll,ll> lazy[N << 2];

ll f(ll a, ll b){return a+b;}//function

pair<ll,ll> ff(pair<ll,ll> a, pair<ll,ll> b){return pair<ll,ll>{a.first+
   b.first,a.second+b.second};}//function

//node p is always ok

// PA FORMULAS
// an = a1 + r * (n-1)
// sn = (a1 + an) * (n) / 2
void push(int p, int tl, int tr)
{
    if(tl == tr || lazy[p] == pair<ll,ll>{emp,emp})
        return;
    int tmid = (tl+tr) >> 1;

    //solve propagations
    ll at = lazy[p].first + lazy[p].second * (tmid-tl);
    ll sum = (at + lazy[p].first) * (tmid-tl+1) >> 1;
    tree[lef(p)] = f(tree[lef(p)], sum);

```

```
    at += lazy[p].second;
    ll at2 = lazy[p].first + lazy[p].second * (tr - tl);
    sum = (at + at2) * (tr-tmid) >> 1;
    tree[rig(p)] = f(tree[rig(p)], sum);

    //update lazy
    lazy[lef(p)] = ff(lazy[lef(p)], lazy[p]);
    lazy[rig(p)] = ff(lazy[rig(p)], pair<ll,ll>{at,lazy[p].second});

    //clean lazy
    lazy[p] = {emp,emp};
}

void build(int p, int tl, int tr)
{
    if(tl == tr)
    {
        tree[p] = a[tl];
    return;
    }
  int tmid = (tl + tr) >> 1;
  build(lef(p), tl, tmid);
  build(rig(p), tmid+1, tr);
  tree[p] = f(tree[lef(p)], tree[rig(p)]);

}

void updt(int p, int tl, int tr, int l, int r, ll v)
{
    if(l > r)
        return ;
    if(tl == l && tr == r){
        ll at = tl-v+1;
        ll at2 = tr -v+1;
        ll sum = (at + at2) * (tr-tl+1) >> 1;
        tree[p] = f(tree[p], sum);

        lazy[p] = ff(lazy[p], {tl-v+1,1});
        return;
    }
    //push if go down
    push(p,tl,tr);

    int tmid = (tl+tr) >> 1;
    updt(lef(p), tl, tmid, l, min(r,tmid), v);
    updt(rig(p), tmid+1, tr, max(l,tmid+1), r, v);
    tree[p] = f(tree[lef(p)], tree[rig(p)]);

}



ll query(int p, int tl, int tr, int l, int r)
{
    if(l > r)
        return emp;
    if(tl == l && tr == r)
        return tree[p];
    //push if go down
    push(p,tl,tr);
    int tmid = (tl+tr) >> 1;
    return f(query(lef(p), tl, tmid, l, min(r,tmid)),
        query(rig(p), tmid+1, tr, max(l,tmid+1), r));
}
```

4.6 Sub-segmento contiguo com maior soma

Nessa a query é achar o subsegmento contíguo com maior soma numa range $[L, R]$. O update é mudar o valor de um ponto.

```
const int N = 200100;
const ll emp = -2000000000000000LL; //NULL VALUE

struct dt
{
    ll l, r, mx, tot;
    dt(){};
    dt(ll a):l(a), r(a), mx(max(a,0LL)), tot(a){};
};
ll a[N];
dt tree[N << 2];

dt f(dt a, dt b){
    dt x;
    x.mx = max(a.mx, b.mx);
    x.mx = max(x.mx, a.r + b.l);
    x.r = max(a.r + b.tot, b.r);
    x.l = max(b.l + a.tot, a.l);
    x.tot = a.tot + b.tot;
    return x;
}//function


void build(int p, int tl, int tr)
{
    if(tl == tr)
    {
        tree[p] = dt(a[tl]);
    return;
    }
  int tmid = (tl + tr) >> 1;
  build(lef(p), tl, tmid);
  build(rig(p), tmid+1, tr);
  tree[p] = f(tree[lef(p)], tree[rig(p)]);

}


void updt_point(int p, int tl, int tr, int k, ll v)
{
    if(tl == tr){
        tree[p] = dt(v);
        return;
    }

    int tmid = (tl+tr) >> 1;
    if (tmid < k)
        updt_point(rig(p), tmid+1, tr, k, v);
    else
        updt_point(lef(p), tl, tmid, k, v);
    tree[p] = f(tree[lef(p)], tree[rig(p)]);
    // cout << tl << " " << tr << ' '<< tree[p].mx << '\n';
}
```

4.7 Contar minimos (união de area de retangulos)

We would like a data structure that can efficiently handle two types of operations:

Update index i to value v Report the minimum and the number of occurences of the minimum on a range $[l, r]$

We can use a normal segment tree to handle range queries, but slightly modify each node and the merge operation. Let each node be a pair of values ($\texttt{val}, \texttt{cnt}$), where `val` is the minimum value and `cnt` is the number occurences of the minimum value.

If node c has two children a and b, then

if $a.\texttt{val} < b.\texttt{val}$, then $c = a$ if $a.\texttt{val} > b.\texttt{val}$, then $c = b$ if $a.\texttt{val} = b.\texttt{val}$, then $c = \{a.\texttt{val}, a.\texttt{cnt} + b.\texttt{cnt}\}$

O problema clássico é unir a área de vários retângulos. O codigo faz esse.

```
const int N = 1000001;

vector<pii> event[2*N + 100][2];

pii stree[8*N+ 100];
int lazy[8*N+ 100];

pii merge(pii a, pii b)
{
    if(a.ft == b.ft)
        a.sd+=b.sd;
    if(a.ft <= b.ft)
        return a;
    return b;
}

void push(int p, int tl, int tr)
{
    if(tl != tr && lazy[p] != 0)
    {
        stree[lef(p)].ft += lazy[p];
        stree[rig(p)].ft += lazy[p];
        lazy[lef(p)] += lazy[p];
        lazy[rig(p)] += lazy[p];
    }
    lazy[p] = 0;
}

void build(int p, int tl, int tr)
{
    if(tl == tr)
    {
        stree[p] = {0,1};
        return;
    }
    int tmid = (tl+tr) >> 1;
    build(lef(p), tl, tmid);
    build(rig(p), tmid+1, tr);
    stree[p] = merge(stree[lef(p)],stree[rig(p)]);
}


void updt(int p, int tl, int tr,int l, int r, int x)
{
    if(l > r)
        return;
    if(tl == l && tr == r)
    {
        stree[p].ft+=x;
        lazy[p]+=x;
        return;
    }
    int tmid = (tl+tr) >> 1;
    push(p, tl, tr);
    updt(lef(p), tl, tmid, l, min(r,tmid), x);
    updt(rig(p), tmid+1, tr, max(l, tmid+1), r, x);
    stree[p] = merge(stree[lef(p)],stree[rig(p)]);
}

```

```

void test()
{
    int n;
    cin >> n;
    pii pt;
    rep(i, 1, n+1)
    {
        // cout << i << endl;
        int x1,y1,x2,y2;
        cin >> x1 >> y1 >> x2 >> y2;
        pii at = pii{y1+N,y2+N-1};
        // cout << x1+N << ' ' << x2+N << endl;
        event[x1+N][0].pb(at);
        event[x2+N][1].pb(at);
    }
    build(1,1,2*N);
    ll rs = 0;
    rep(i, 0, 2*N)
    {
        for(auto [l,r]:event[i][0])
        {
            // cout << l << ' ' << r << '\n';
            updt(1, 1, 2*N, l, r, 1);
        }
        for(auto [l,r]:event[i][1])
        {
            // cout << l << ' ' << r << '\n';
            updt(1, 1, 2*N, l, r, -1);
        }
        // if(stree[1].sd != 2*N)
        //     cout << i << ' ' << stree[1] << '\n';
        if(stree[1].ft != 0)
            rs += 2*N;
        else
            rs += 2*N-stree[1].sd;
    }
    cout << rs << '\n';
}
```

4.8 Merge Sort Tree

Very strong Data Structure to answer queries but u can't do updates.

A problem of this kind is something like that: You have an array and want to answer some queries on it:

- Given l, r, x ($l \leq x \leq r$), how many elements in this range are greater than x?

The idea is that each node of your tree is an array of elements now, and you will maintain a sorted array in each node. To answer the query, one must do a binary search in each node of the range to find how many of the elements are greater than x. Time complexity of this is $\mathcal{O}(n \log^2(n))$

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

typedef long long ll;

#define all(x) x.begin(), x.end()
#define lef(x) (x << 1)
#define rig(x) (lef(x) | 1)
#define pb push_back

const int N = 30010;

int a[N];
vector<int> tree[N << 2];
```

```cpp

void build(int x, int tl, int tr){
    if(tl == tr){
        tree[x].pb(a[tl]);
    }
    else{
        int tm = tl + tr >> 1;
        build(lef(x), tl, tm);
        build(rig(x), tm+1, tr);
        tree[x].resize(tr-tl+1);
        merge(all(tree[lef(x)]), all(tree[rig(x)]), tree[x].begin());
    }
}

int get(int x, int tl, int tr, int l, int r, int k){
    if(tl > r || tr < l)
        return 0;
    else if(tl >= l && tr <= r)
        return (int)tree[x].size() - (upper_bound(all(tree[x]), k) - tree[x].begin());
    else{
        int tm = tl + tr >> 1;
        return get(lef(x),tl,tm,l,r,k) + get(rig(x),tm+1,tr,l,r,k);
    }
}

int32_t main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

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

    build(1,1,n);

    int q; cin >> q;
    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        cout<<get(1,1,n,l,r,x)<<"\n";
    }

    return 0;
}
```

4.9 Segment Tree Beats

Updates:
min(a[i],x) - range l - r
max(a[i],x) - range l - r
a[i]+=x - range l - r
Query
sum ai - range l-r
$O(N * log(N)^2)$

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

const int MAXN = 200001;  // 1-based
const ll llINF=1e18;
#define lef(x) (x<<1)
```

```
#define rig(x) (lef(x) | 1)
int N;
ll A[MAXN];

struct Node {
  ll sum;  // Sum tag
  ll max1;  // Max value
  ll max2;  // Second Max value
  ll maxc;  // Max value count
  ll min1;  // Min value
  ll min2;  // Second Min value
  ll minc;  // Min value count
  ll lazy;  // Lazy tag
} T[MAXN * 4];

void merge(int t) {
  // sum
  T[t].sum = T[lef(t)].sum + T[rig(t)].sum;

  // max
  if (T[lef(t)].max1 == T[rig(t)].max1) {
    T[t].max1 = T[lef(t)].max1;
    T[t].max2 = max(T[lef(t)].max2, T[rig(t)].max2);
    T[t].maxc = T[lef(t)].maxc + T[rig(t)].maxc;
  } else {
    if (T[lef(t)].max1 > T[rig(t)].max1) {
      T[t].max1 = T[lef(t)].max1;
      T[t].max2 = max(T[lef(t)].max2, T[rig(t)].max1);
      T[t].maxc = T[lef(t)].maxc;
    } else {
      T[t].max1 = T[rig(t)].max1;
      T[t].max2 = max(T[lef(t)].max1, T[rig(t)].max2);
      T[t].maxc = T[rig(t)].maxc;
    }
  }

  // min
  if (T[lef(t)].min1 == T[rig(t)].min1) {
    T[t].min1 = T[lef(t)].min1;
    T[t].min2 = min(T[lef(t)].min2, T[rig(t)].min2);
    T[t].minc = T[lef(t)].minc + T[rig(t)].minc;
  } else {
    if (T[lef(t)].min1 < T[rig(t)].min1) {
      T[t].min1 = T[lef(t)].min1;
      T[t].min2 = min(T[lef(t)].min2, T[rig(t)].min1);
      T[t].minc = T[lef(t)].minc;
    } else {
      T[t].min1 = T[rig(t)].min1;
      T[t].min2 = min(T[lef(t)].min1, T[rig(t)].min2);
      T[t].minc = T[rig(t)].minc;
    }
  }
}

void push_add(int t, int tl, int tr, ll v) {
  if (v == 0) {
    return;
  }
  T[t].sum += (tr - tl + 1) *v;
  T[t].max1 += v;
  if (T[t].max2 != -llINF) {
    T[t].max2 += v;
  }
  T[t].min1 += v;
  if (T[t].min2 != llINF) {
    T[t].min2 += v;
```

```
  }
  T[t].lazy += v;
}

// corresponds to a chmin update
void push_max(int t, ll v, bool l) {
  if (v >= T[t].max1) {
    return;
  }
  T[t].sum -= T[t].max1 * T[t].maxc;
  T[t].max1 = v;
  T[t].sum += T[t].max1 * T[t].maxc;
  if (l) {
    T[t].min1 = T[t].max1;
  } else {
    if (v <= T[t].min1) {
      T[t].min1 = v;
    } else if (v < T[t].min2) {
      T[t].min2 = v;
    }
  }
}

// corresponds to a chmax update
void push_min(int t, ll v, bool l) {
  if (v <= T[t].min1) {
    return;
  }
  T[t].sum -= T[t].min1 * T[t].minc;
  T[t].min1 = v;
  T[t].sum += T[t].min1 * T[t].minc;
  if (l) {
    T[t].max1 = T[t].min1;
  } else {
    if (v >= T[t].max1) {
      T[t].max1 = v;
    } else if (v > T[t].max2) {
      T[t].max2 = v;
    }
  }
}

void pushdown(int t, int tl, int tr) {
  if (tl == tr)
    return;
  // sum
  int tm = (tl + tr) >> 1;
  push_add(lef(t), tl, tm, T[t].lazy);
  push_add(rig(t), tm + 1, tr, T[t].lazy);
  T[t].lazy = 0;

  // max
  push_max(lef(t), T[t].max1, tl == tm);
  push_max(rig(t), T[t].max1, tm + 1 == tr);

  // min
  push_min(lef(t), T[t].min1, tl == tm);
  push_min(rig(t), T[t].min1, tm + 1 == tr);
}

void build(int t=1, int tl=0, int tr=N-1) {
  T[t].lazy = 0;
  if (tl == tr) {
    T[t].sum = T[t].max1 = T[t].min1 = A[tl];
    T[t].maxc = T[t].minc = 1;
    T[t].max2 = -llINF;
```

```
    T[t].min2 = llINF;
    return;
  }

  int tm = (tl + tr) >> 1;
  build(lef(t), tl, tm);
  build(rig(t), tm + 1, tr);
  merge(t);
}

void update_add(int l, int r, ll v, int t=1, int tl=0, int tr=N-1) {
  if (r < tl || tr < l) {
    return;
  }
  if (l <= tl && tr <= r) {
    push_add(t, tl, tr, v);
    return;
  }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  update_add(l, r, v, lef(t), tl, tm);
  update_add(l, r, v, rig(t), tm + 1, tr);
  merge(t);
}

void update_chmin(int l, int r, ll v, int t=1, int tl=0, int tr=N-1) {
  if (r < tl || tr < l || v >= T[t].max1) {
    return;
  }
  if (l <= tl && tr <= r && v > T[t].max2) {
    push_max(t, v, tl == tr);
    return;
  }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  update_chmin(l, r, v, lef(t), tl, tm);
  update_chmin(l, r, v, rig(t), tm + 1, tr);
  merge(t);
}

void update_chmax(int l, int r, ll v, int t=1, int tl=0, int tr=N-1) {
  if (r < tl || tr < l || v <= T[t].min1) {
    return;
  }
  if (l <= tl && tr <= r && v < T[t].min2) {
    push_min(t, v, tl == tr);
    return;
  }
  pushdown(t, tl, tr);

  int tm = (tl + tr) >> 1;
  update_chmax(l, r, v, lef(t), tl, tm);
  update_chmax(l, r, v, rig(t), tm + 1, tr);
  merge(t);
}

ll query_sum(int l, int r, int t=1, int tl=0, int tr=N-1) {
  if (r < tl || tr < l) {
    return 0;
  }
  if (l <= tl && tr <= r) {
    return T[t].sum;
  }
  pushdown(t, tl, tr);
```

```
  int tm = (tl + tr) >> 1;
  return query_sum(l, r, lef(t), tl, tm) + query_sum(l, r, rig(t), tm +
   1, tr);
}

int main() {
  int Q;

  cin >> N >> Q;
  for (int i = 0; i < N; i++) {
    cin >> A[i];
  }
  build();
  for (int q = 0; q < Q; q++) {
    int t; cin >> t;
    if (t == 0) {
      int l, r;
      ll x;
      cin >> l >> r >> x;
      update_chmin(l, r - 1, x);
    } else if (t == 1) {
      int l, r;
      ll x;
      cin >> l >> r >> x;
      update_chmax(l, r - 1, x);
    } else if (t == 2) {
      int l, r;
      ll x;
      cin >> l >> r >> x;
      update_add(l, r - 1, x);
    } else if (t == 3) {
      int l, r;
      cin >> l >> r;
      cout << query_sum(l, r - 1) << '\n';
    }
  }
}
```

4.10 Wavelet Tree

Encontra o kth menor valor num range [L,R] com queries em O(log n).
Espaço e pre-processamento é em O(n log n).
Isso é do kataki e tecnicamente é uma seg tree.
p é o L e **q** é o R na query o resto aceita que funciona

```
/*
WAVELET TREE, encontra o kth menor valor numa range [L,R] com queries em
    O(log n)ç
Espaco e pre-processamento é em O(n log n)
referencia: https://www.quora.com/How-can-you-build-a-data-structure-on-
   an-array-that-returns-kth-order-statistics-on-subarrays-in-
   logarithmic-time

artigo : https://users.dcc.uchile.cl/~jperez/papers/ioiconf16.pdf

*/
#include<bits/stdc++.h>
using namespace std;
#define MAXN 10000
int N;
struct elem {
    int val, pos;
    bool operator< (elem b) const {
        return val<b.val;
```

```
    }
};
int* tree[4*MAXN+10];
elem temp[MAXN], arr[MAXN], sorted[MAXN];
int* merge(int e, int d) {
    int* num_left = (int*) malloc(sizeof(int) * (d - e + 1));
    int left = e, right = (e+d)/2+1;
    int i = 0, cnt = 0;
    while (left <= (e+d)/2 && right <= d) {
        if (arr[left].pos <= arr[right].pos) {
            num_left[i] = ++cnt;
            temp[i] = arr[left++];
        }
        else {
            num_left[i] = cnt;
            temp[i] = arr[right++];
        }
        i++;
    }
    while (left <= (e+d)/2) {
        num_left[i] = ++cnt;
        temp[i] = arr[left++];
        i++;
    }
    while (right <= d) {
        num_left[i] = cnt;
        temp[i] = arr[right++];
        i++;
    }
    for (int j = 0; j < (d-e+1); j++) {
        arr[e+j]=temp[j];
    }
    return num_left;
}
void create_tree (int i=1, int e=1, int d=N) {
  if (e == d) return;
  else {
    create_tree(2*i, e, (e+d)/2);
    create_tree(2*i+1, (e+d)/2 + 1, d);
    tree[i] = merge(e-1, d-1);
  }
}
int query(int p, int q, int k, int i=1, int st=1, int end=N) {
    if (st == end) return sorted[st-1].val;
    int left = (p!=1 ? tree[i][p-2] : 0);
    int right = tree[i][q-1];
    int diff = right - left;
    if (diff >= k)
        return query(left+1,right,k,2*i,st,st+(end-st)/2);
    else
        return query(p-left,q-right,k-diff,2*i+1,st+(end-st)/2+1,end);
}
int main() {
  scanf("%d",&N);
  for (int i = 0; i < N; i++) {
    scanf("%d", &sorted[i].val);
    sorted[i].pos = i;
  }
    sort(sorted, sorted+N);
    memcpy(arr,sorted,sizeof(sorted));
    create_tree();
    printf("%d\n",query(1,N,3));
}
```

Capítulo V

Dynammic Programming

5.1 Fast Knapsack 3k trick

Suponha que você tem um conjunto A com N valores e $\sum_{i=0}^{N-1} a[i] = M$. Queremos calcular quais possiveis somas podemos obter.

Primeiro, se todos os valores são distintos $N < \sqrt(M)$.

Se não são, podemos representar os K caras iguais a x como sendo $x, 2*x, 4*x$ até o maior $2^i < K$ assim esses elementos representaram a mesma soma que os anteriores. Chegando numa solução que tem complexidade $O(M * \sqrt(M)$. se for só um subset sum, podemos ainda usar um bitset.

Exemplo 1:

You are in a book shop which sells n different books. You know the price $h[i]$, the number of pages $s[i]$ and the number of copies of each book $k[i]$.

You have decided that the total price of your purchases will be at most x. What is the maximum number of pages you can buy? You can buy several copies of the same book.

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

//template

const int N = 100010;

vector<pii> obj;
int dp[N], vh[N], vs[N], vk[N];
void test()
{
    int n, x;
    cin >> n >> x;

    rep(i, 0, n)
        cin >> vh[i];
    rep(i, 0, n)
        cin >> vs[i];
    rep(i, 0, n)
        cin >> vk[i];
    rep(i, 0, n)
    {
        int h = vh[i], s = vs[i], k = vk[i];
        int sum = 1, pg = s, cost = h;
        while(sum <= k)
        {
            k-=sum;
            obj.eb(pg,cost);
            pg+=pg;
            cost+=cost;
            sum+=sum;
        }
        if(k != 0)
        {
            obj.eb(s*k,h*k);
        }
    }
    int mx = 0;
    dp[0] = 0;
```

```
    for(auto [p,c]:obj)
    {
        // cout << p << ' ' << c << '\n';
        rep(i, mx+1, 0)
        {
            if(i+c <= x)
                dp[i+c] = max(dp[i]+p,dp[i+c]);
        }
        mx+=c;
        mx = min(mx, x);
    }
    int rs = 0;
    rep(i, 0, x+1)
        rs = max(rs, dp[i]);
    cout << rs << '\n';
}


int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int t = 1;
    // cin >> t;
    while(t--)
        test();
    return 0;
}
```

Exemplo 2:

A group of n children are coming to Helsinki. There are two possible attractions: a child can visit either (zoo) or (amusement park).

There are m pairs of children who want to visit the same attraction. Your task is to find all possible alternatives for the number of children that will visit zoo. The children's wishes have to be taken into account.

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

//template

const int N = 100100;

vector<int> adj[N];
bitset<N> bt;
int vis[N], cont[N];
int dfs(int v)
{
    vis[v] = 1;
    int qt = 1;
    for(int u: adj[v])
    {
        if(vis[u])
            continue;
        qt+=dfs(u);
    }
    return qt;
}
vector<int> movs;
void test()
{
    int n, m;
    cin >> n >> m;
    while(m--)
    {
        int a, b;
        cin >> a >> b;
```

```cpp
            adj[a].pb(b);
            adj[b].pb(a);
        }
        rep(i, 1, n+1)
            if(!vis[i])
                cont[dfs(i)]++;

        rep(i, 1, n+1)
        {
            int s = i, tot = cont[i]*i;
            while(tot > 0)
            {
                movs.pb(min(tot,s));
                tot-=s;
                s+=s;
            }
        }
        for(int x:movs)
        {
            bt |= bt<<x;
            bt[x] = 1;
        }
        rep(i, 1, n+1)
        {
            cout << bt[i];
        }
        cout << '\n';

}

int32_t main()
{
        ios::sync_with_stdio(false);
        cin.tie(NULL);
        int t = 1;
        // cin >> t;
        while(t--)
            test();
        return 0;
}
```

5.2 SOS DP

Sum over subsets (SOS) DP is a trick that allows you to efficiently compute the sum of all the subsets of an array.

The naive solution would be to iterate through every pair of masks and check if one of them is a subset of the other. We can speed this up if we iterate over only subsets of the current mask and add up all of the those values to get the sum over subsets for a particular mask.

The difference comes from the fact that in the first example we iterate over every pair of subsets which takes $(2^n)^2$ time and the second we iterate directly over the subsets for each mask. This means each mask is only visited by 2^{n-k} other masks where k is the number of elements of the mask.

This means that the total time complexity is $\mathrm{O}(\sum_0^n \binom{n}{k} \cdot 2^{n-k} = 3^n)$.

Notice how in both of these examples we don't seem to be saving much information between different subsets which is the essence of DP. Define $\texttt{SOS}(\texttt{mask}, x)$ to be the sum of subsets of mask such that the first x bits of the subset are identical to the first x bits of mask. For example, $\texttt{SOS}(1001001, 3)$ includes the subsets 1001001, 1000001, 1001000, 1000000 which all have the same common prefix of 100.

$$\texttt{SOS}(mask, x) = \begin{cases} \texttt{SOS}(\texttt{mask}, x-1) + \texttt{SOS}(\texttt{mask} - 2^i, x-1) & \text{if } |2^i \wedge \texttt{mask}| > 0 \\ \texttt{SOS}(\texttt{mask}, x-1) & \text{otherwise} \end{cases}$$

Example: Given a list of n integers, your task is to calculate for each element x:

- the number of elements y such that $x|y = x$, can be seen as $y \subseteq x$.
- the number of elements y such that $x\&y = x$, that is equal to $!x|!y = !x$, can be seen as $!y \subseteq !x$.
- the number of elements y such that $x\&y \neq 0$, that is the complement of to $!x|y = !x$, can be seen as $y \subseteq !x$.

solution of the problem is $O(X)$, where X is the greates element in n.

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

const int MAXN = 2e5+100;
const int MAXMASK = (1<<20)+100;
const int MAXB = 21;

int X[MAXN];

int dp[MAXMASK][MAXB];
int custo[MAXMASK];
int resp[MAXN][3];

void sosdp(){
    rep(mask, 0, MAXMASK){
        dp[mask][MAXB-1] = custo[mask];
        for (int i = MAXB-2; i >= 0; i--){
        // rep (i, MAXB-1, 0){
            dp[mask][i] = dp[mask][i+1];
            if (mask&(1 << i)){
                dp[mask][i] += dp[mask - (1 << i)][i+1];
            }
        }
    }
}

void test()
{
    int n;
    cin >> n;
    rep(i, 0, n){
        cin >> X[i];
        custo[X[i]]++;
    }
    sosdp();
    rep(i, 0, n){
        resp[i][0] = dp[X[i]][0];
        int mask = X[i]^((1<<(MAXB-1))-1);
        resp[i][2] = dp[mask][0];
        custo[X[i]]--;
        custo[mask]++;
        // cout << mask << endl;
    }
    sosdp();
    rep(i, 0, n){
        int mask = X[i]^((1<<(MAXB-1))-1);
        resp[i][1] = dp[mask][0];
    }
    rep(i, 0, n){
        cout << resp[i][0] << " " << resp[i][1] << " " << n-resp[i][2]
   << "\n";
    }
}

int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int t = 1;
    // cin >> t;
    while(t--)
        test();
    return 0;
```

```
}
```

5.3 Permutation Trick

suppose u have to count something over permutations. It can be modelled as:

$DP[i]$ counting over permutations with i elements. The main idea is to iterate j over 1 to i, and we say that element i was positioned at j, and move every other to the right, dealing with the counting in the process.

Example:

Your task is to count the number of permutations of $1, 2, \ldots, n$ that have exactly k inversions.

$DP[i][j] =$ permutations with i elements and j inversions.

$DP[i][j] = \sum_{k=0}^{i} dp[i-1][j-(i-k)]$

In the problem u need to do some optimizations with prefix sum.

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

//template

const int N = 510;
const ll mod = 1000000007LL;

ll dp[2][N*N];

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n, k;

    cin >> n >> k;

    dp[0][0] = 1;
    rep(j, 1, k + 1)
        dp[0][j] = 1;

    rep(i, 1, n+1)
    {
        rep(j, 0, k + 1)
        {
            int l = j-i;
            dp[1][j] = dp[0][j];
            if(l >= 0)
                dp[1][j] -= dp[0][l];
            if(dp[1][j] < 0)
                dp[1][j] += mod;
        }
        // error(i);
        // error(dp[1][k]);
        dp[0][0] = dp[1][0];
        rep(j, 1, k + 1){
            dp[0][j] = dp[1][j] + dp[0][j-1];
            // cout << dp[0][j] << '\n';
            if(dp[0][j] >= mod)
                dp[0][j] -= mod;
        }
    }
    cout << dp[1][k] << '\n';
    return 0;
}
// 3 2 1 -> 1
```

5.4 Open Interval Trick

Suppose u have to count something over multiple intervals, and each element need to be added to some interval. Suppose they are sorted.

$DP[i][k]$ counting with the first i elements added and there are k intervals "open". the main idea is to iterate j over 1 to i, and we say that element i was positioned at j, and move every other to the right, dealing with the counting in the process.

Example:

Your company has n coders, and each of them has a skill level between 0 and 100. Your task is to divide the coders into teams that work together.

The penalty for creating a team is the skill level difference between the best and the worst coder.

In how many ways can you divide the coders into teams such that the sum of the penalties is at most x?

We just need another state to x, and then do the combinations.

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

//template

const int N = 510;
const ll mod = 1000000007LL;

ll dp[2][N*N];

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n, k;

    cin >> n >> k;

    dp[0][0] = 1;
    rep(j, 1, k + 1)
        dp[0][j] = 1;

    rep(i, 1, n+1)
    {
        rep(j, 0, k + 1)
        {
            int l = j-i;
            dp[1][j] = dp[0][j];
            if(l >= 0)
                dp[1][j] -= dp[0][l];
            if(dp[1][j] < 0)
                dp[1][j] += mod;
        }
        // error(i);
        // error(dp[1][k]);
        dp[0][0] = dp[1][0];
        rep(j, 1, k + 1){
            dp[0][j] = dp[1][j] + dp[0][j-1];
            // cout << dp[0][j] << '\n';
            if(dp[0][j] >= mod)
                dp[0][j] -= mod;
        }
    }
    cout << dp[1][k] << '\n';
    return 0;
}
// 3 2 1 -> 1
```

5.5 Dp Optimizations

- $A[i][j]$ — the smallest k that gives optimal answer, for example in $dp[i][j] = dp[i-1][k] + C[k][j]$.

Name	**Original Recurrence**	**Sufficient Condition of Applicability**	**Original Complexity**	**Optimized Complexity**
Convex Hull Optimization1	$dp[i] = min_{j<i}\{F[j] + b[j] * a[i]\}$	$b[j] \geq b[j+1]$ and $a[i] \leq a[i+1]$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$ or $\mathcal{O}(nlog(n))$ w/ line container
Divide and Conquer Optimization	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$\mathcal{O}(kn^2)$	$\mathcal{O}(knlog(n))$
Knuth Optimization	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$

Tabela V.1

- $C[i][j]$ — some given cost function.
- $F[j]$ - Value computed in constant time, frequently will be equal to $dp[j]$.

5.5.1 D&C

```
 //Optimization dp[i][j] = min{dp[i-1][k]+C[k][j]}
 //O(n * k^2) -> O(n * k * log k)


int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

5.5.2 Knuth Optimization

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 4005;
int dp[MAXN][MAXN];
int opt[MAXN][MAXN];
int sum[MAXN][MAXN];

int cost(int i, int j)
{
  return sum[j][j] - sum[i][j] - sum[j][i] + sum[i][i];
}
int main()
{
  int n,k;
  n = get();
  k = get();
  for(int i = 1; i <= n ; i++)
  {
    for(int j = 1; j <= n ; j++)
    {
      sum[i][j] = get();
      sum[i][j] += sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1];
    }
  }
  for(int i  = 1; i <= n ; i++)
  {
    dp[i][1] = cost(0, i);
    opt[1][i] = 1;
  }
  int aux = 0;
  for(int i = 2; i <= k ; i++)
  {
    for(int j = n; j >= 1; j--)
    {
      dp[j][i] = INT_MAX;
      opt[n+1][i] = n;
      for(int l = opt[j][i-1]; l <= opt[j+1][i]; l++)
      {
        aux = cost(l,j);
        if( dp[j][i] > dp[l][i-1] + aux)
        {
          dp[j][i] = dp[l][i-1] + aux;
          opt[j][i] = l;
        }
      }
    }
  }
  printf("%d\n", dp[n][k] >> 1);

  return 0;
}
```

5.5.3 Convex Hull Trick

Convex Hull Trick 1

This solutions only works when the coefficients are all increasing / decreasing, and the queries decreasing / increasing. Improve $O(n^2)$ to $O(n)$. (Original Problem: Codeforces - Round 189 (Div. 1) C)

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

using namespace std;

struct line {
```

```
    long long m, c;
    long long eval(long long x) { return m * x + c; }
    long double intersectX(line l) { return (long double)
    (c - l.c) / (l.m - m); }
};

const int N = int(2e5);
typedef long long    ll;
ll a[N], b[N];

int main(){
    int n;

    scanf("%d", &n);

    for(int i = 1 ; i <= n ; i++){
        scanf("%lld", &a[i]);
        // a[i] *= -1;
    }

    for(int i = 1 ; i <= n ; i++){
        scanf("%lld", &b[i]);
    }

    deque<line> dq;
    dq.push_front({b[1], 0LL});//caso base

    ll ans = b[1];

    for (int i = 2; i <= n; i++) {

        while (dq.size() >= 2 && dq.back().eval(a[i]) > dq[dq.size() -
    2].eval(a[i])) //Inverta caso queira o ámximo
            dq.pop_back();

        ll f = dq.back().eval(a[i]);

        if(i == n){
            ans = f;
        }

        line cur = {b[i], f};
        while (dq.size() >= 2 && cur.intersectX(dq[0]) <= dq[0].
    intersectX(dq[1]))
            dq.pop_front();
        dq.push_front(cur);
    }

    printf("%lld\n", ans);

    return 0;
}
```

Convex Hull Trick 2

This solutions works under the same conditions for Convex Hull Trick 1, but it's used to solve problems in a 2D dynamic programming, like problem NKLEAVES from spoj. In resume we need to group N leaves in K groups, for each coordinate between 1 and N there is a leaf with weight w_i, and the leaves can only be moved to the left, and the cost is $w_i * d$ where d is the distance that leaf i was moved. The problem asks for minimum cost to group the N leaves in K groups. Lets reverse the leaves weights, now the leafs can only be moved to the right.

So, the recurrence is:

$dp_{i,j} = \min_{k \le i}((\sum_{p=k}^{i}(i-p) * w_p) + dp_{k-1,j-1})$, clearly $O(n^2k)$

So we will keep our lines in such way, that we can solve this problem, see the code below.

This solutions only works when the coefficients are all increasing / decreasing, and the queries decreasing / increasing. Improve $O(n^2k)$ to $O(nk)$. (Original Problem: SPOJ - NKLEAVES)

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

using namespace std;

struct line {
    long long m, c;
    long long eval(long long x) { return m * x + c; }
    long double intersectX(line l) { return (long double) (c - l.c) / (l
   .m - m); }
};

const int N = int(1e5 + 100);
const int K = 20;
typedef long long    ll;
ll x[N], w[N];
ll dp[N][K];
ll pref[N], g[N];

int main(){
    int n, k;

    while(scanf("%d %d", &n, &k) != EOF){
      for(int i = 1 ; i <= n ; i++){
          scanf("%lld", &w[i]);
          x[i] = i;

          // a[i] *= -1;
      }

      reverse(w + 1, w + 1 + n);

      for(int i = 1 ; i <= n ; i++){
        pref[i] = pref[i - 1] + w[i];
        g[i] = w[i] * x[i] + g[i - 1];
      }

      for(int i = 1 ; i <= n ; i++){
        dp[i][1] = x[i] * pref[i] - g[i];
      }

      deque<line> dq;

      for(int j = 2 ; j <= k ; j++){
        dq.push_front({0LL, 0LL});

        for (int i = 1; i <= n; i++) {
            while (dq.size() >= 2 && dq.back().eval(x[i]) > dq[dq.size()
    - 2].eval(x[i])){ //Inverta caso queira o maximo
                dq.pop_back();
            }

            ll f = dq.back().eval(x[i]);
            dp[i][j] = x[i] * pref[i] - g[i] + f;
            line cur = {-pref[i], dp[i][j - 1] + g[i]};

            while (dq.size() >= 2 && cur.intersectX(dq[0]) <= dq[0].
   intersectX(dq[1]))
                dq.pop_front();

            dq.push_front(cur);
        }

        dq.clear();
```

```
    }

      printf("%lld\n", dp[n][k]);
    }

    return 0;
}
```

Convex Hull Trick 3 (Online Queries)

This solutions works without any assumption about the coeficients, in this case we will query is answered in $O(logn)$.

Improve $O(n^2)$ to $O(nlogn)$. (Original Problem: Codeforces - Round 463 div1+div2 F. Escape Through Leaf)

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

using namespace std;

typedef long long ll;

const int N = int(1e5 + 10);
int subtree[N];
ll a[N], b[N];
ll dp[N];
bool is_leaf[N];
vector< int > adj[N];

template <class TT = ll>
struct Line{
    // coef angular, linear, criterio de comparacao
    mutable TT k,m,p;
    // aqui eu quero sempre deixar os com menor coef angular mais pra
      frente, conc pra baixo
    bool operator <(const Line & o) const{
        return k>o.k;
    }
    bool operator <(const TT o) const{
        return p<o;
    }
};
struct CHT : multiset<Line,less<>>{
    static const TT inf = std::numerical_limit<TT>::max();
    TT div(TT a, TT b){
        return a/b-((a^b) < 0 && a%b);
    }

    // x eh melhor que y?
    bool isect(iterator x, iterator y){
        if(y == end()){
            x->p=inf;
            return false;
        }
        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(TT k, TT m){
        auto z= insert({k,m,0});
        auto y=z++;
        auto 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)
            insect(x,erase(y));
    }
    TT query(TT x){
    assert(!empty());
        auto ans=lower_bound(x);
        return ans->k*x+ans->m;
    }
};

int dfs(int u, int ft){
  is_leaf[u] = 1;

  for(auto v: adj[u]){
    if(v == ft){
      continue;
    }

    is_leaf[u] = 0;
  }

  if(is_leaf[u]){
    return subtree[u] = 1;
  }

  subtree[u] = 1;

  for(auto v: adj[u]){
    if(v == ft){
      continue;
    }

    subtree[u] += dfs(v, u);
  }

  return subtree[u];
}

void dfs1(int u, int ft, CHT &cur){
  if(is_leaf[u]){
    dp[u] = 0LL;
    cur.add(-b[u], -dp[u]);

    return;
  }

  int mx = 0;
  int big_son = 0;

  for(auto v: adj[u]){
    if(v == ft){
      continue;
    }

    if(subtree[v] > mx){
      big_son = v;
      mx = subtree[v];
    }
  }

  dfs1(big_son, u, cur);

  for(auto v: adj[u]){
    if(v == big_son || v == ft){
      continue;
    }
```

```

    LineContainer tmp;
    dfs1(v, u, tmp);

    for(auto d: tmp){
      //printf("aqui!\n");
      cur.add(d.k, d.m);
    }
  }

  dp[u] = -cur.query(a[u]);
  cur.add(-b[u], -dp[u]);
}

int main(){
  int n;

  scanf("%d", &n);

  for(int i = 1 ; i <= n ; i++){
    scanf("%lld", &a[i]);
  }

  for(int i = 1 ; i <= n ; i++){
    scanf("%lld", &b[i]);
  }

  for(int i = 1 ; i < n ; i++){
    int ui, vi;

    scanf("%d %d", &ui, &vi);

    adj[ui].push_back(vi);
    adj[vi].push_back(ui);
  }

  CHT cur;
  dfs(1, 1);
  dfs1(1, 1, cur);

  for(int i = 1 ; i <= n ; i++){
    printf("%lld ", dp[i]);
  }

  printf("\n");

  return 0;
}
```

5.6 Steiner Tree DP

MST for a subset

```
#include <iostream>
using namespace std;
//5717 - Peach Blossom Spring
#define INF 1000000000

int dp[1<<10][64];

bool check(int m, int K) {
    int i, tmp1=0, tmp2=0;
    for (i=0; i<K; i++) tmp1 += ((m>>i)&1);
    for (i=0; i<K; i++) tmp2 += ((m>>(K+i))&1);
    return tmp1 == tmp2;
```

```
}

int main() {
    int N, M, K, i, j, k, t, T, a, b, c, d[64][64], RES, tmp;

    cin >> T;
    for (t=0; t<T; t++) {
        cin >> N >> M >> K;

        for (i=0; i<N; i++) {
            d[i][i] = 0;
            for (j=i+1; j<N; j++) d[i][j] = d[j][i] = INF;
        }

        for (i=0; i<M; i++) {
            cin >> a >> b >> c; a--; b--;
            d[a][b] = d[b][a] = min(c, min(d[a][b], d[b][a]));
        }

        for (k=0; k<N; k++) for (i=0; i<N; i++) for (j=0; j<N; j++) d[i][j] = min(d[i][j], d[i][k]+d[k][j]);

        for (i=0; i<(1<<(2*K)); i++) for (j=0; j<N; j++) dp[i][j] = INF;
        for (i=0; i<K; i++) for (j=0; j<N; j++) dp[1<<i][j] = d[i][j];
        for (i=0; i<K; i++) for (j=0; j<N; j++) dp[1<<(K+i)][j] = d[N-1-i][j];

        for (i=1; i<(1<<(2*K)); i++) {
            for (j=0; j<N; j++) {
                if (dp[i][j] != INF) continue;

                for (k=(i-1)&i; k; k=(k-1)&i) dp[i][j] = min(dp[i][j], dp[k][j]+dp[i^k][j]);
            }

            if (check(i, K)) {
                tmp = INF;
                for (j=0; j<N; j++) tmp = min(tmp, dp[i][j]);
                for (j=0; j<N; j++) dp[i][j] = tmp;
            } else {
                for (j=0; j<N; j++) for (k=0; k<N; k++) dp[i][j] = min(dp[i][j], dp[i][k] + d[k][j]);
            }
        }

        RES = INF;
        for (i=0; i<N; i++) RES = min(RES, dp[(1<<(2*K))-1][i]);
        if (RES < INF) cout << RES << endl;
        else cout << "No solution" << endl;
    }

    return 0;
}
```

Capítulo VI

String

6.1 String Hash

Compare two strings in $O(1)$ time, with $O(N)$ time preprocessing.

6.1.1 Simple Hash

```cpp
// String Hash - Polynomial rolling hash
#include <bits/stdc++.h>
using namespace std;

//template

const int N = 200100;
const uint64_t p=33, mod=1000000007, p2=73, mod2=1000000009;
uint64_t ppow[N], ppow2[N];

void build()
{
    ppow[0] = 1;
    ppow2[0] = 1;
    for (int i = 1; i < N; i++) {
        ppow[i] = (ppow[i-1] * p) % mod;
        ppow2[i] = (ppow2[i-1] * p2) % mod2;
    }
}
struct Hash {
    vector<pair<uint64_t, uint64_t>> h;

    Hash () {}

    void init(string &s) {
        h.resize(s.size()+2);
        h[0] = {5389ULL, 5389ULL};
        for (size_t i = 0; i < s.size(); i++) {
            int code = s[i];
            h[i+1].first = (h[i].first * p + code) % mod;
            h[i+1].second = (h[i].second * p2 + code) % mod2;
        }
    }

    pair<uint64_t, uint64_t> get_hash(int i, int j) {
        pair<uint64_t, uint64_t> r;
        r.first = (h[j+1].first - (h[i].first * ppow[j-i+1]) % mod + mod
    ) % mod;
        r.second = (h[j+1].second - (h[i].second * ppow2[j-i+1]) % mod2
    + mod2) % mod2;
        return r;
    }
};

uint64_t fastExp(uint64_t a, uint64_t p, uint64_t mod) {
```

```
    return p == 0 ? 1 : ((fastExp((a * a) % mod, p / 2, mod) % mod) * (p & 1 ? a%mod : 1)) % mod;
}

pair<uint64_t, uint64_t> repeatHash(pair<uint64_t, uint64_t> r, int size, int qtd){
    uint64_t q = ppow[size], q2 = ppow2[size];
    uint64_t num = fastExp(q, qtd, mod) - 1, num2 = fastExp(q2, qtd, mod2) - 1;
    uint64_t den = fastExp((q  - 1) , mod - 2, mod), den2 = fastExp((q2 - 1), mod2 - 2, mod2);
    return mp((num * den % mod) * r.ft % mod, (num2 * den2 % mod2) * r.sd % mod2);
}

pair<uint64_t, uint64_t> getQtdCharsInACiclicSubstring(int l, int r, int qtd, Hash &hat){
    int size = r - l + 1;
    if (qtd <= size){
        return hat.get_hash(l, l+qtd-1);
    }
    pair<uint64_t, uint64_t> h = hat.get_hash(l, r);
    int integerPart = qtd / size;
    int rest = qtd - size * integerPart;
    pair<uint64_t, uint64_t> repeatedPart = repeatHash(h, size, integerPart, hat);
    pair<uint64_t, uint64_t> restPart = (rest == 0 ? mp((uint64_t)0, (uint64_t)0) : hat.get_hash(l, l+rest-1));
    return mp(
        (restPart.ft + repeatedPart.ft * ppow[rest] % mod) % mod,
        (restPart.sd + repeatedPart.sd * ppow2[rest] % mod2) % mod2
    );
}

*/
```

6.1.2 Hash 2D$_{\text{ayllon}}$

Cuidado, a hash é dupla, pode dar TLE! As string são indexados de 0! Apenas dentro da hash que setamos para ser idexado de 1 (igual string hash normal).

Problema resolvido: Dado duas matrizes de char, achar as posições das ocorrências da primeira matriz na segunda.

```
const int N = 2010;
const uint64_t p[4] = {33, 73, 37, 93}, mod[2] = {1000000007, 1000000009};
uint64_t ppow[4][N];

void build()
{
    for (int j = 0; j < 4; j++){
        ppow[j][0] = 1;
        for (int i = 1; i < N; i++) {
            ppow[j][i] = (ppow[j][i-1] * p[j]) % mod[j>>1];
        }
    }
}
struct Hash {
    int n, m;
    vector<vector<pair<uint64_t, uint64_t>>>h;

    Hash () {}

    void init(vector<string> &word2d) {
        n = word2d.size();
        m = word2d[0].size();
```

```
        h.resize(n+1, vector(m+1, mp((uint64_t)0,(uint64_t)0)));

        pair<uint64_t, uint64_t> sv;
        for (int i = 1; i <= n; i++) {
            sv = {0, 0};
            for (int j = 1; j <= m; j++){
                int code = word2d[i-1][j-1];
                sv = {(sv.ft * p[0] + code)%mod[0],
                      (sv.sd * p[2] + code)%mod[1]};
                h[i][j] = {(h[i-1][j].ft * p[1])%mod[0] + sv.ft,
                           (h[i-1][j].sd * p[3])%mod[1] + sv.sd};
                if (h[i][j].ft >= mod[0]) h[i][j].ft -= mod[0];
                if (h[i][j].sd >= mod[1]) h[i][j].sd -= mod[1];
            }
        }
    }

    pair<uint64_t, uint64_t> get_hash(int i, int j, int height, int width) {
        int i2 = i + height, j2 = j + width;
        pair<uint64_t, uint64_t> r = {
            (h[i2][j2].ft + mod[0] + mod[0]
                - (h[i][j2].ft * ppow[1][i2-i])%mod[0]
                - (h[i2][j].ft * ppow[0][j2-j])%mod[0]
                + ((ppow[1][i2-i] * ppow[0][j2-j])%mod[0] * h[i][j].ft)%mod[0]
            )%mod[0],
            (h[i2][j2].sd + mod[1] + mod[1]
                - (h[i][j2].sd * ppow[3][i2-i])%mod[1]
                - (h[i2][j].sd * ppow[2][j2-j])%mod[1]
                + ((ppow[3][i2-i] * ppow[2][j2-j])%mod[1] * h[i][j].sd)%mod[1]
            )%mod[1]
        };
        return r;
    }
};

Hash h1, h2;
vector<string> ss1, ss2;

void test()
{
    int n,m,n2,m2;
    string s;
    cin >> n >> m;
    rep(i, 0, n){
        cin >> s;
        ss1.pb(s);
    }
    cin >> n2 >> m2;
    rep(i, 0, n2){
        cin >> s;
        ss2.pb(s);
    }
    build();
    h1.init(ss1);
    h2.init(ss2);

    auto target = h1.get_hash(0, 0, n, m);
    for(int i = 0; i < n2 - n + 1; i++){
        for(int j = 0; j < m2 - m + 1; j++){
            if (target == h2.get_hash(i, j, n, m)){
                cout << "MATCH " << i << " " << j << "\n";
            }
        }
```

```
    }
}
```

6.1.3 Hash with Updates

For this problem, the idea is that you can compare the hashes of the string and it's reversed one to discover if it is a palindrome. To do the updates, just use some data structure that allows point update and range queries. The one below uses a BIT.

```
// Palindrome Queries - CSES Solution
// link: https://cses.fi/problemset/task/2420

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

typedef long long ll;

#define all(x) x.begin(), x.end()
#define ft first
#define sd second
#define mp make_pair

const int N = 2e5+3;
const ll mod = 1e9+7, mod2 = 1e9+9;
const ll p = 33, p2 = 73;

struct updtHash{
    vector<ll> ppow, ppow2;
    vector<pair<ll,ll>> h;
    updtHash(){
        ppow.resize(N);
        ppow2.resize(N);
        ppow[0] = 1;
        ppow2[0] = 1;
        for (int i = 1; i < N; i++) {
            ppow[i] = (ppow[i-1] * p) % mod;
            ppow2[i] = (ppow2[i-1] * p2) % mod2;
        }
    }

    ll inv(ll x, int w){
        ll md = (w ? mod2 : mod);
        ll b = md-2;
        ll rs = 1;

        while(b){
            if(1&b) rs = rs*x%md;
            x = x*x%md;
            b >>= 1;
        }
        return rs;
    }

    void updt(int x, pair<ll,ll> v){
        while(x < N){
            h[x].ft += v.ft;
            h[x].sd += v.sd;
            if(h[x].ft > mod) h[x].ft -= mod;
            if(h[x].sd > mod2) h[x].sd -= mod2;
            x += (x&-x);
        }
    }

    pair<ll,ll> get(int x){
        pair<ll,ll> r = {0ll, 0ll};
        while(x){
```

```
            r.ft += h[x].ft;
            r.sd += h[x].sd;
            if(r.ft > mod) r.ft -= mod;
            if(r.sd > mod2) r.sd -= mod2;
            x -= (x&-x);
        }

        return r;
    }

    void init(string &s) {
        h.assign(N, mp(0, 0));
        for (int i = 0; i < s.size(); i++) {
            int code = s[i];
            pair<ll, ll> v;
            v.first = code * ppow[i+1] % mod;
            v.second = code * ppow2[i+1] % mod2;
            updt(i+1, v);
        }
    }

    pair<ll,ll> get_hash(int i, int j){
        pair<ll,ll> r, l;
        r = get(j+1);
        l = get(i);
        r.ft = (r.ft - l.ft + mod) * inv(ppow[i], 0) % mod;
        r.sd = (r.sd - l.sd + mod2) * inv(ppow2[i], 1) % mod2;

        return r;
    }

    void updt_char(int i, int code){
        pair<ll,ll> l, r, v;
        r = get(i+1);
        l = get(i);
        v.ft = (code * ppow[i+1] - r.ft + l.ft + mod) % mod;
        v.sd = (code * ppow2[i+1] - r.sd + l.sd + mod2) % mod2;
        updt(i+1, v);
    }
};

updtHash h1, h2;

void test(){
    int n, q;
    cin >> n >> q;
    string s;
    cin >> s;

    h1.init(s);
    reverse(all(s));
    h2.init(s);

    while(q--){
        int t;
        cin >> t;
        if(t == 1){
            int x;
            char c;
            cin >> x >> c;
            h1.updt_char(x-1, c);
            h2.updt_char(n-x, c);
        }
        else{
            int l, r;
            cin >> l >> r;
```

```
            pair<ll,ll> v1, v2;
            v1 = h1.get_hash(l-1, r-1);
            v2 = h2.get_hash(n-r, n-l);
            if(v1 == v2){
                cout<<"YES\n";
            }
            else{
                cout<<"NO\n";
            }
        }
    }
}

int32_t main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int teste = 1;
    // cin >> teste;

    while(teste--){
        test();
    }

    return 0;
}
```

6.2 Suffix Array

The key idea is to make a substring problem, into a prefix problem. Remember to add a '#' at the end of the string. Another important fact is that $lcp(i,j) = min_{k=i}^{j-1} lcp[k]$.

```
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
```

```
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n
]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}



vector<int> lcp_construction(string const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}


void test()
{
    string s;
    cin >> s;
    s.pb('#');
    vector<int> sa = sort_cyclic_shifts(s);
    vector<int> lcp = lcp_construction(s,sa);

}
```

6.3 Suffix Automaton

The key idea is to make a substring problem, into a DAG problem.

link points to the greatest suffix

Most of harder problems, you can just sort the states by len, and walk from greater to smaller making some updates in a dp like style.

```
const int N = 200100;
struct state {
    int len, link, qt = 0;
    map<char, int> next;
};
state st[N * 2];
int sz, last;
void sa_init() {
    st[0].len = 0;
```

```
    st[0].link = -1;
    sz++;
    last = 0;
}
void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    st[cur].qt = 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

Example 1:

You are given a string of length n. If all of its substrings (not necessarily distinct) are ordered lexicographically, what is the k-th smallest of them?

DP to count Paths(remember its a DAG) then just do a walk. Easier version doesnt need to count repetitions, so just a path.

```
ll dp[N];
string bfs(ll k)
{
    vector<pair<int,int>> ids;
    for(int pos = 1; pos < sz; pos++)
        ids.push_back({st[pos].len,pos});
    sort(ids.begin(), ids.end());
    reverse(ids.begin(), ids.end());
    for(auto [l,x]:ids){
        dp[x] = st[x].qt;
        // cout << x << ' ' << st[x].qt << '\n';
        for(auto [c,y]:st[x].next)
        {
            dp[x] += dp[y];
        }
        if(st[x].link != -1){
            // cout << x << " " << st[x].link << " << link\n";
            st[st[x].link].qt += st[x].qt;
        }
    }
    reverse(ids.begin(), ids.end());
    int pos = 0;
    string resp;
    while(k>0){
        for(auto [c,y]:st[pos].next)
        {
            if(dp[y] < k){
```

```
                // cout << c << ' ' << dp[y] << ' ' << y <<"\n";
                k-= dp[y];
            }
            else{
                k-=st[y].qt;
                // cout << c << ' ' << dp[y] << " " << y <<" <<go\n";
                resp.push_back(c);
                pos = y;
                break;
            }
        }
    }
    return resp;
}
```

Example 2:

You are given a string of length n. For every integer between 1...n you need to print the number of distinct substrings of that length.

```
int getlen(int x)
{
    if(x<=0)
        return 0;
    return st[x].len;
}
int rs[N];
void bfs(int n)
{
    vector<pair<int,int>> ids;
    for(int pos = 0; pos < sz; pos++){
        rs[getlen(st[pos].link)+1]++;
        rs[getlen(pos)+1]--;
    }
    for(int i = 1; i <= n; i++){
        rs[i] += rs[i-1];
        cout << rs[i] << ' ';
    }
    cout << '\n';
}
```

Example 3:

A repeating substring is a substring that occurs in two (or more) locations in the string. Your task is to find the longest repeating substring in a given string.

```
int best = 0, dep[N], pai[N];
string bfs()
{
    vector<pair<int,int>> ids;
    for(int pos = 1; pos < sz; pos++)
        ids.push_back({st[pos].len,pos});
    sort(ids.begin(), ids.end());
    reverse(ids.begin(), ids.end());
    for(auto [l,x]:ids){
        if(st[x].link != -1){
            // cout << x << " " << st[x].link << " << link\n";
            st[st[x].link].qt += st[x].qt;
        }
    }
    reverse(ids.begin(), ids.end());
    int at = -1;
    for(auto [l,pos]:ids){
        if(st[pos].qt<2)
            continue;
        if(dep[pos] == 0){
            dep[pos] = 1;
        }
        if(dep[pos] > best){
            best = dep[pos];
```

```
                at = pos;
            }
        for(auto [x,y]:st[pos].next)
        {
            if(st[y].qt>1){
                if(dep[y] < dep[pos]+1)
                {
                    dep[y] = dep[pos]+1;
                    pai[y] = pos;
                }
                dep[y] = max(dep[y],dep[pos]+1);
            }
        }
    }
    if(at == -1)
        return "-1";
    string resp;
    while(at)
    {
        for(auto [x,y]:st[pai[at]].next)
        {
            if(y == at){
                resp.push_back(x);
            }
        }
        at = pai[at];
    }
    reverse(all(resp));
    return resp;

}
```

6.4 Suffix Tree

The idea is that you transform a string problem into a tree problem, so you can use tree techniques to solve it, like LCA, Euler Tour, DP on Trees, and many others.

(Original Problem SPOJ - TOP 10).

```
const int NS =200100 ;
const int N =  NS*2;

int cn, cd, ns, en = 1, lst;
vector<int>/*string*/ S[NS]; int si = -1;


vector<int> sufn[N];

struct node {
  int l, r, si;
  int p, suf;
  //map<char, int> adj;
  map<int, int> adj;
  node() : l(0), r(-1), suf(0), p(0) {}
  node(int L, int R, int S, int P) : l(L), r(R), si(S), p(P) {}
  inline int len() { return r - l + 1; }
  inline int operator[](int i) { return S[si][l + i]; }
  inline int& operator()(int c) { return adj[c]; }
} t[N];

inline int new_node(int L, int R, int S, int P) {
  t[en] = node(L, R, S, P);
  return en++;
}

void add_string(vector<int>/*string*/ &s) {
```

```
  //s += '$';
  S[++si] = s;
  sufn[si].resize(s.size() + 1);
  cn = cd = 0;
  int i = 0; const int n = s.size();
  for(int j = 0; j < n; j++)
    for(; i <= j; i++) {
      if(cd == t[cn].len() && t[cn](s[j]))
        cn = t[cn](s[j]), cd = 0;
      if(cd < t[cn].len() && t[cn][cd] == s[j]) {
        cd++;
        if(j < s.size() - 1) break;
        else {
          if(i) t[lst].suf = cn;
          for(; i <= j; i++) {
            sufn[si][i] = cn;
            cn = t[cn].suf;
          }
        }
      } else if(cd == t[cn].len()) {
        sufn[si][i] = en;
        if(i) t[lst].suf = en; lst = en;
        t[cn](s[j]) = new_node(j, n - 1, si, cn);
        cn = t[cn].suf;
        cd = t[cn].len();
      } else {
        int mid = new_node(t[cn].l, t[cn].l + cd - 1, t[cn].si, t[cn].p)
  ;
        t[t[cn].p](t[cn][0]) = mid;
        if(ns) t[ns].suf = mid;
        if(i) t[lst].suf = en; lst = en;
        sufn[si][i] = en;
        t[mid](s[j]) = new_node(j, n - 1, si, mid);
        t[mid](t[cn][cd]) = cn;
        t[cn].p = mid; t[cn].l += cd;
        cn = t[mid].p;
        int g = cn? j - cd : i + 1;
        cn = t[cn].suf;
        while(g < j && g + t[t[cn](S[si][g])].len() <= j)
          cn = t[cn](S[si][g]), g += t[cn].len();
        if(g == j)
          ns = 0, t[mid].suf = cn, cd = t[cn].len();
        else
          ns = mid, cn = t[cn](S[si][g]), cd = j - g;
      }
    }
}
```

6.5 Palindromic Tree

```
//UVALive - 6072
//http://adilet.org/blog/palindromic-tree/
#include <bits/stdc++.h>
using namespace std;

const int N=200010;
typedef long long ll;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
    int lazy;
```

```
};

string s;
node tree[N];
int num;            // node 1 - root with len -1, node 2 - root with len
    0
int suff;           // max suffix palindrome

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        tree[suff].lazy++;
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;
    tree[num].lazy=1;
    // cout<<num<<" "<<let<<"\n";
    if (tree[num].len == 1) {
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}

void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}
ll value[26],pot[N];
vector<pair<ll,ll>> pl;
const ll mod=777777777LL;
void dfs(int t)
{
    for (int i = 0; i < 26; ++i)
    {
        // cout<<t<<" "<<tree[t].next[i]<<'\n';
        if(tree[t].next[i])
        {
            dfs(tree[t].next[i]);
```

```
        }
    }
}
void dfs1(int t,ll at,int pt)
{
    // cout<<t<<' '<<at<<" "<<tree[t].lazy<<'\n';
    if(t>2)
        pl.push_back({at,tree[t].lazy});
    for (int i = 0; i < 26; ++i)
    {
        if(tree[t].next[i])
        {
            // cout<<t<<" --- "<<(char)(i+'a')<<" --> "<<tree[t].next[i
    ]<<"\n";
            dfs1(tree[t].next[i],(at+value[i]*pot[pt])%mod,pt+1);
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    pot[0]=1;
    for (int i = 1; i < N; ++i)
    {
        pot[i] = (pot[i-1]*26LL)%mod;
    }
    int q;
    cin>>q;
    while(q--)
    {

        int n,m;
        cin>>n>>m;
        cin>>s;

        initTree();

        for (int i = 0; i < n; i++) {
            addLetter(i);
        }
        for (int i = num; i >0 ; --i)
        {
            if(tree[i].sufflink)
                tree[tree[i].sufflink].lazy+=tree[i].lazy;
        }
        // cout<<num<<" "<<s<<'\n';
        // dfs(1);
        for (int i = 0; i < m; ++i)
        {
            ll k;
            cin>>k;
            for (int j = 0; j < 26; ++j)
            {
                cin>>value[j];
                value[j]%=mod;
            }
            pl.clear();
            dfs1(1,0,0);
            dfs1(2,0,0);
            sort(pl.begin(),pl.end());
            ll now=0;
            int j=0;
            while(j<pl.size() && now+pl[j].second<k)
            {
                now+=pl[j++].second;
```

```
            }
            cout<<pl[j].first<<"\n";
        }
        cout<<"\n";
        for (int i = 0; i <= num; ++i)
        {
            tree[i].len=tree[i].sufflink=tree[i].num=tree[i].lazy=0;
            for (int j = 0; j < 26; ++j)
            {
                tree[i].next[j]=0;
            }
        }
    }

    return 0;
}
```

6.6 Minimal Rotation

Find the minimal cyclic shift of an string in O(N).

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define all(x) x.begin(), x.end()
#define lef(x) (x<<1)
#define rig(x) (lef(x)|1)
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    string s;
    cin >> s;
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k)
            i += j - k;
    }
    cout << s.substr(ans, n / 2) << '\n';
    return 0;
}
```

6.7 Aho Corasick

This aho has the shape of the Suffix Automaton, mos of things that can be done in there, can be done here as well(sort by len and do something using links).

```
const int K = 26;
struct node {
    int fim = 0,p,link = 0,elink = -1,len;
    int go[K] = {0},nxt[K] = {0};
    node(int p=0,int len = 0) : p(p),len(len) {};
};
vector<node> t(1);
```

```
void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        ch-='a';
        if (!t[v].nxt[ch]) {
            t[v].nxt[ch] = t.size();
            t.eb(v,t[v].len+1);
        }
        v = t[v].nxt[ch];
    }
    // maybe change this to a vector with some ids
    t[v].fim = 1;
}

void calc_link()
{
    queue<int> q;
    q.push(0);
    while(!q.empty())
    {
        int v = q.front();
        q.pop();
        int lk = t[v].link;
        rep(i, 0, K)
        {
            if(!t[v].nxt[i])
                t[v].go[i] = t[lk].go[i];
            else{
                t[v].go[i] = t[v].nxt[i];
                t[t[v].go[i]].link = (!v?0:t[lk].go[i]);
                q.push(t[v].nxt[i]);
            }
        }
        //exit link is the first link with fim != 0
        if(t[lk].fim)
            t[v].elink = lk;
        else
            t[v].elink = t[lk].elink;
    }
}
```

6.8 Trie

Tá safe

```
const int N = 1000500;
const int mod=1e9+7;
typedef long long ll;
struct node{
    bool leaf;
    map<int,int> prox;
    node(){
        leaf = false;
    }
};
int id=0;
node nodes[N];
void insert_trie(string s){
    int curr=0;
    for(char c: s){
        if(nodes[curr].prox[c-'a']==0)
            nodes[curr].prox[c-'a']=++id;
        curr=nodes[curr].prox[c-'a'];
    }
    nodes[curr].leaf=true;
```

```
}
bool find_trie(string s){
    int curr=0;
    for(char c: s){
        if(!nodes[curr].prox[c-'a'])
            return false;
        curr=nodes[curr].prox[c-'a'];
    }
    if(nodes[curr].leaf) return true;
    return false;
}
```

Capítulo VII

Graph

7.1 Flow

7.1.1 Teoremas de Fluxo

- **Corte mínimo:**
  - Valor do fluxo máximo é igual ao valor do corte mínimo.
- **Emparelhamento:**
  - O emparelhamento máximo em um grafo bipartido é igual ao fluxo máximo.
- **Teorema Berge (1957):**
  - Seja $G = (V, E)$ um grafo. Um emparelhamento $M \subseteq E$ é máximo se não há caminhos aumentantes.
- **Teorema Hall (1935):**
  - Seja G um grafo bipartido com bipartição (X, Y). Então G admite um emparelhamento que satura todos os vértices de X se e somente se $|N(S)| \geq |S|$ para todo $S \subseteq X$.
  - $|N(x)|$, para algum $x \in V$, é o conjunto vizinhança do vértice x, isto é, $|N(x)| = \{y$, para todo y tal que existe $(x, y) \in E\}$.
  - $N(S) = \underset{s \in S}{\cup} N(s)$.
- **Teorema Tutte**
  - G tem um emparelhamento perfeito se e somente se $i(G - S) \leq |S|$, para todo $S \subseteq V(G)$.
  - Uma componente de um grafo é ímpar/par se possui um número ímpar/par de vértices. Denotaremos por $i(G)$ o número de componentes ímpares de um grafo G.
- **Teorema Kőnig-Egerváry (1931):**
  - Seja $G = (V, E)$ um grafo bipartido. Então cobertura(G) = emparelhamento(G).
  - Let (S, T) be a minimum cut. Let $A = A_S \cup A_T$ and $B = B_S \cup B_T$, such that $A_S, B_S \subset S$ and $A_T, B_T \subset T$. Then the minimum cut is composed only of edges going from s to A_T or from B_S to t, as any edge from A_S to B_T would make the size of the cut infinite.

    Therefore, the size of the minimum cut is equal to $|A_T| + |B_S|$. On the other hand, $A_T \cup B_S$ is a vertex cover, as any edge that is not incident to vertices from A_T and B_S must be incident to a pair of vertices from A_S and B_T, which would contradict the fact that there are no edges between A_S and B_T. Thus, $A_T \cup B_S$ is a minimum vertex cover of G.
  - basically just check if the cut was on a edge related to S or to T.
- **Teorema Independência:**
  - Seja $G = (V, E)$ um grafo. Então independência$(G) = n -$ cobertura(G).
- **Teorema Menge (1927):**
  - Seja $D = (V, E)$ um grafo direcionado e $s, t \in V$ dois vértices distintos. Então o número máximo de st-caminhos disjuntos nas arestas é igual ao número mínimo de arestas cuja remoção impossibilita a existência de st-caminhos.
  - Seja $D = (V, E)$ um grafo direcionado e $s, t \in V$ dois vértices distintos. Então o número máximo de st-caminhos disjuntos nos vértices é igual ao número mínimo de vértices cuja remoção impossibilita a existência de st-caminhos.

  - Seja $D = (V, E)$ um grafo direcionado, onde toda aresta tem capacidade 1, e $s, t \in V$ dois vértices distintos. Seja f^* o fluxo máximo e seja K^* um corte mínimo separador. Então:
    * val(f^*) = número máximo de caminho disjuntos nas arestas.
    * cal(K^*) = número mínimo de arestas em D cuja remoção impossibilita a existência de st-caminhos.

- **Observações:**
  - Dá pra usar um monte de gambiarra junto com Fluxo, tipo Dijkstra, Busca Binária, perceber que a vizinhança é igual e juntar elas de algum jeito, entre outras.

7.1.2 Dinic

```cpp
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
```

```
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].
    flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].
    flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long f = 0;
    long long flow() {
        //long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
    vector<pair<int, int>> mincut()
    {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
        vector<pair<int, int>> cut;
        bfs();
        for (auto & e: edges) {
            if (e.flow == e.cap && level[e.v] != -1 && level[e.u] == -1
    && e.cap > 0) {
                cut.eb(e.v, e.u);
            }
        }
        return cut;
    }
};
```

7.1.3 MinCostMaxFlow

The problem is given a Graph with a source and sink vertices, each edge has a capacity c_i and cost l_i per unit of flow, compute the minimum cost to transport a max flow in this network.

```
template <class T = int>
class MCMF {
public:
    struct Edge {
        Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
        int to;
        T cap, cost;
    };

    MCMF(int size) {
        n = size;
        edges.resize(n);
        pot.assign(n, 0);
        dist.resize(n);
```

```
        visit.assign(n, false);
    }

    pair<T, T> mcmf(int src, int sink) {
        pair<T, T> ans(0, 0);
        if(!SPFA(src, sink)) return ans;
        fixPot();
        // can use dijkstra to speed up depending on the graph
        while(SPFA(src, sink)) {
            auto flow = augment(src, sink);
            ans.first += flow.first;
            ans.second += flow.first * flow.second;
            fixPot();
        }
        return ans;
    }

    void addEdge(int from, int to, T cap, T cost) {
        edges[from].push_back(list.size());
        list.push_back(Edge(to, cap, cost));
        edges[to].push_back(list.size());
        list.push_back(Edge(from, 0, -cost));
    }
private:
    int n;
    vector<vector<int>> edges;
    vector<Edge> list;
    vector<int> from;
    vector<T> dist, pot;
    vector<bool> visit;

    /*bool dij(int src, int sink) {
        T INF = numeric_limits<T>::max();
        dist.assign(n, INF);
        from.assign(n, -1);
        visit.assign(n, false);
        dist[src] = 0;
        for(int i = 0; i < n; i++) {
            int best = -1;
            for(int j = 0; j < n; j++) {
                if(visit[j]) continue;
                if(best == -1 || dist[best] > dist[j]) best = j;
            }
            if(dist[best] >= INF) break;
            visit[best] = true;
            for(auto e : edges[best]) {
                auto ed = list[e];
                if(ed.cap == 0) continue;
                T toDist = dist[best] + ed.cost + pot[best] - pot[ed.to
  ];
                assert(toDist >= dist[best]);
                if(toDist < dist[ed.to]) {
                    dist[ed.to] = toDist;
                    from[ed.to] = e;
                }
            }
        }
        return dist[sink] < INF;
    }*/

    pair<T, T> augment(int src, int sink) {
        pair<T, T> flow = {list[from[sink]].cap, 0};
        for(int v = sink; v != src; v = list[from[v]^1].to) {
            flow.first = min(flow.first, list[from[v]].cap);
            flow.second += list[from[v]].cost;
        }
```

```
        for(int v = sink; v != src; v = list[from[v]^1].to) {
            list[from[v]].cap -= flow.first;
            list[from[v]^1].cap += flow.first;
        }
        return flow;
    }

    queue<int> q;
    bool SPFA(int src, int sink) {
        T INF = numeric_limits<T>::max();
        dist.assign(n, INF);
        from.assign(n, -1);
        q.push(src);
        dist[src] = 0;
        while(!q.empty()) {
            int on = q.front();
            q.pop();
            visit[on] = false;
            for(auto e : edges[on]) {
                auto ed = list[e];
                if(ed.cap == 0) continue;
                T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
                if(toDist < dist[ed.to]) {
                    dist[ed.to] = toDist;
                    from[ed.to] = e;
                    if(!visit[ed.to]) {
                        visit[ed.to] = true;
                        q.push(ed.to);
                    }
                }
            }
        }
        return dist[sink] < INF;
    }

    void fixPot() {
        T INF = numeric_limits<T>::max();
        for(int i = 0; i < n; i++) {
            if(dist[i] < INF) pot[i] += dist[i];
        }
    }
};
```

7.2 Matching

7.2.1 HopCroft Karp

Complexity for sparse graphs $O(Elog(V))$ Complexity for dense graphs $O(Esqrt(V))$

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

#define MAX_V1 1000000
#define MAX_V2 1000000
#define MAX_E 8000000

int V1 = 0,V2 = 0,l[MAX_V2],r[MAX_V1];
int E,to[MAX_E],nex[MAX_E],last[MAX_V1];

void hk_init(){
    memset(last,-1,sizeof last);
    E=0;
}

void hk_add_edge(int u, int v){
    to[E] = v; nex[E] = last[u]; last[u] = E++;
```

```
}

bool visited[MAX_V1];

bool hk_dfs(int u){
    if(visited[u]) return false;
    visited[u] = true;

    for(int e = last[u],v;e != -1;e = nex[e]){
        v = to[e];

        if(l[v] == -1 || hk_dfs(l[v])){
            r[u] = v;
            l[v] = u;
            return true;
        }
    }

    return false;
}

int hk_match(){
    memset(l,-1,sizeof l);
    memset(r,-1,sizeof r);
    bool change = true;

    while(change){
        change = false;
        memset(visited,false,sizeof visited);

        for(int i = 0;i < V1;++i)
            if(r[i] == -1)
                change |= hk_dfs(i);
    }

    int ret = 0;

    for(int i = 0;i < V1;++i)
        if(r[i] != -1) ++ret;

    return ret;
}

int n,m;
bool valid(int x, int y)
{
  return (x > 0  && x <= n && y > 0 && y <= m);
}
int id[1005][1005];
int dx[] = {-1,-1,1,1,2,2,-2,-2};
int dy[] = {-2,2,-2,2,-1,1,-1,1};
int main()
{
    int e;
  scanf("%d %d %d",&V1,&V2,&e);
  hk_init();
  for (int i = 0; i < e; ++i)
    {
        int a,b;
        cin>>a>>b;
        hk_add_edge(a,b);
    }

  printf("%d\n",hk_match());
    for (int i = 0; i < V1; ++i)
    {
```

```
        if(r[i]!=-1)
            cout<<i<<' '<<r[i]<<"\n";
    }
  return 0;
}
```

7.2.2 Bipartide Weighted Hungarian Method

Complexity $O(V^3)$

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

using namespace std;

typedef long long  ll;

template<typename T>
class hungarian {
public:
    int n;
    int m;
    vector< vector< T > >a;
    vector< T > u;
    vector< T > v;
    vector< int > pa;
    vector< int > pb;
    vector< int > way;
    vector< T > minv;
    vector< bool > used;
    T inf;

    hungarian(int _n, int _m) : n(_n), m(_m) {
        assert(n <= m);
        a = vector< vector< T > >(n, vector< T >(m));
        u = vector< T >(n + 1);
        v = vector< T >(m + 1);
        pa = vector< int >(n + 1, -1);
        pb = vector< int >(m + 1, -1);
        way = vector< int >(m, -1);
        minv = vector< T >(m);
        used = vector< bool >(m + 1);
        inf = numeric_limits< T >::max();
    }

    inline void add_row(int i){
        fill(minv.begin(), minv.end(), inf);
        fill(used.begin(), used.end(), false);
        pb[m] = i;
        pa[i] = m;
        int j0 = m;
        do {
            used[j0] = true;
            int i0 = pb[j0];
            T delta = inf;
            int j1 = -1;
            for(int j = 0 ; j < m ; j++){
                if(!used[j]){
                    T 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[pb[j]] += delta;
                    v[j] -= delta;
                }else{
                    minv[j] -= delta;
                }
            }
            j0 = j1;
        }while(pb[j0] != -1);

        do{
            int j1 = way[j0];
            pb[j0] = pb[j1];
            pa[pb[j0]] = j0;
            j0 = j1;
        }while(j0 != m);
    }

    inline T current_score(){
        return -v[m];
    }

    inline pair< long long, vector< int > > solve() {
        vector< int > ans(n, 0);
        for(int i = 0 ; i < n ; i++){
            add_row(i);
        }
        for(int i = 0 ; i < n ; i++){
            ans[pb[i]] = i;
        }

        return make_pair(current_score(), ans);
    }
};

const int N = 510;
const ll mx = ll(1e4) + 10;

ll prefx[N], sufx[N];
ll prefy[N], sufy[N];
ll px[N], py[N];
ll vx[N], vy[N];
vector< pair< ll, int > > x, y;

int main(){

    x.push_back({-mx, -1});
    y.push_back({-mx, -1});

    int n;

    scanf("%d", &n);

    for(int i = 0 ; i < n ; i++){
        scanf("%lld %lld", &px[i], &py[i]);
        x.push_back({px[i], i});
        y.push_back({py[i], i});
    }

    for(int i = 0 ; i < n ; i++){
        scanf("%lld %lld", &vx[i], &vy[i]);
    }

```

```
    sort(x.begin(), x.end());
    sort(y.begin(), y.end());

    for(int i = 1 ; i < x.size() ; i++){
        prefx[i] = x[i].first + prefx[i - 1];
        prefy[i] = y[i].first + prefy[i - 1];
    }

    for(int i = int(x.size()) - 1 ; i >= 1 ; i--){
        sufx[i] = sufx[i + 1] + x[i].first;
        sufy[i] = sufy[i + 1] + y[i].first;
    }

    hungarian< ll > hg(n, n);
    const ll INF = ll(1e14);

    for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < n ; j++){
            int idx = lower_bound(x.begin(), x.end(), make_pair(px[i], i
)) - x.begin();
            int idy = lower_bound(y.begin(), y.end(), make_pair(py[i], i
)) - y.begin();

            ll wx = 2LL * px[i] * vx[j] + vx[j] * vx[j] * ll(n);
            ll wy = 2LL * py[i] * vy[j] + vy[j] * vy[j] * ll(n);

            hg.a[i][j] = INF -(wx + wy);
        }
    }

    auto u = hg.solve();

    printf("Yes\n");

    for(int i = 0 ; i < n ; i++){
        printf("%d ", u.second[i] + 1);
    }
    printf("\n");

    return 0;
}
```

7.2.3 General Graph (Edmonds Blossom)

Complexity $O(VE^2)$

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

using namespace std;

const int M=500;
struct struct_edge{int v;struct_edge* n;};
typedef struct_edge* edge;
struct_edge pool[M*M*2];
edge top=pool,adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
int a[M][M];
int n;

void add_edge(int u,int v){
  top->v=v,top->n=adj[u],adj[u]=top++;
  top->v=u,top->n=adj[v],adj[v]=top++;
}

int LCA(int root,int u,int v){
```

```
  static bool inp[M];
  memset(inp,0,sizeof(inp));

  while(1){
    inp[u=base[u]]=true;
    if (u==root) break;
    u=father[match[u]];
  }
  while(1){
    if (inp[v=base[v]]) return v;
    else v=father[match[v]];
  }
}

void mark_blossom(int lca,int u)
{
  while (base[u]!=lca)
    {
      int v=match[u];
      inb[base[u]]=inb[base[v]]=true;
      u=father[v];
      if (base[u]!=lca) father[u]=v;
    }
}
void blossom_contraction(int s,int u,int v){
  int lca=LCA(s,u,v);
  memset(inb,0,sizeof(inb));
  mark_blossom(lca,u);
  mark_blossom(lca,v);
  if (base[u]!=lca)
    father[u]=v;
  if (base[v]!=lca)
    father[v]=u;
  for (int u=0;u<V;u++){
    if (inb[base[u]]){
      base[u]=lca;
      if (!inq[u])
        inq[q[++qt]=u] = true;
    }
  }
}
int find_augmenting_path(int s)
{
  memset(inq,0,sizeof(inq));
  memset(father,-1,sizeof(father));

  for (int i=0;i<V;i++) base[i]=i;
  inq[q[qh=qt=0]=s]=true;

  while (qh<=qt){
    int u = q[qh++];
    for (edge e=adj[u];e;e=e->n){
      int v = e->v;
      if (base[u]!=base[v]&&match[u]!=v){
        if ((v==s) || (match[v]!=-1 && father[match[v]]!=-1)){
          blossom_contraction(s,u,v);
        }else if (father[v]==-1){
          father[v] = u;
          if (match[v]==-1){
            return v;
          }
          else if (!inq[match[v]]){
            inq[q[++qt] = match[v]]=true;
          }
        }
      }
```

```
    }
  }
  return -1;
}

int augment_path(int s,int t)
{
  int u=t,v,w;
  while (u!=-1){
    v=father[u];
    w=match[v];
    match[v]=u;
    match[u]=v;
    u=w;
  }
  return t!=-1;
}

int edmonds()
{
    int matchc=0;
    memset(match,-1,sizeof(match));
    for (int u=0;u<V;u++){
      if (match[u]==-1){
        matchc+=augment_path(u,find_augmenting_path(u));
      }
    }

    top = pool;
    memset(adj, 0, sizeof adj);

    return matchc;
}

bool can(int mid){
    for(int i = 0 ; i < n ; i++){
        for(int j = i + 1 ; j < n ; j++){
            if(a[i][j] >= mid){
                add_edge(i, j);
                // printf("edge(%d, %d)\n", i, j);
            }
        }
    }

    int r = edmonds();

    // printf("para %d temos %d\n", mid, r);

    return r == (n / 2);
}

int main(){
    int t;
    int cs = 1;

    scanf("%d", &t);

    while(t--){
      scanf("%d", &n);
      n = 1 << n;
      V = n;

      for(int i = 0 ; i < n ; i++){
          for(int j = i + 1 ; j < n ; j++){
              scanf("%d", &a[i][j]);
              a[j][i] = a[i][j];
```

```
            }
        }

        int lo = 0, hi = 1e9;
        int r = -1;

        while(lo <= hi){
            int mid = (lo + hi) >> 1;

            if(can(mid)){
                lo = mid + 1;
                r = mid;
            }else{
                // printf("nao consegue!\n");
                hi = mid - 1;
            }
        }

        printf("Case %d: %d\n", cs++, r);
    }

    return 0;
}
```

7.3 Bellman Ford

Algorithm used to calculate minimal distance with negative edges. In this Problem you need to find the minimal path(actually it is maximal) from 1 to n and if it is infinite you need to print -1". There is a important point, that is, a infinite path starting at 1, can actually stop in a sub-graph and don't go to N. A smart way to deal with it is using bellman ford again.

```
#include<bits/stdc++.h>
typedef long long ll;
using namespace std;
typedef pair<int,int> pii;
const int N = 3100;
const int M = 5100;
const ll inf = -1e16;

pair<pii,ll> edges[M];
ll dist[N];
int mark[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n,m;
    cin>>n>>m;
    for (int i = 1; i <= n; ++i)
    {
        dist[i] = inf;
    }
    for (int i = 0; i < m; ++i)
    {
        int a,b;
        cin>>a>>b>>edges[i].second;
        edges[i].first = {a,b};
    }
    dist[1] = 0;
    mark[1] = 0;
    for (int i = 0; i <= n; ++i)
    {
        for (int j = 0; j < m ; ++j)
        {
            int a,b;
```

```
                tie(a,b) = edges[j].first;
                if(dist[a] > inf)
                {
                    if(dist[b] < dist[a] + edges[j].second)
                    {
                        mark[b] = i;
                        dist[b] = dist[a] + edges[j].second;
                    }
                }
            }
        }
    for (int i = 0; i <= n; ++i)
    {
        for (int j = 0; j < m ; ++j)
        {
            int a,b;
            tie(a,b) = edges[j].first;
            mark[b] = max(mark[b],mark[a]);
        }
    }
    if(mark[n]==n)
        cout<<"-1\n";
    else
        cout<<dist[n]<<"\n";
    return 0;
}
```

7.4 2-SAT

In a 2-SAT, all states should be a in conjunctive normal form, so we can see as a and of or's.

- $a \vee b = a \vee b$
- $a \wedge b = (a \vee a) \wedge (b \vee b)$
- $\neg a = \neg a \vee \neg a$
- $a = a \vee a$
- $a \oplus b = (a \vee b) \wedge (\neg a \vee \neg b)$
- $\neg(a \oplus b) = (\neg a \vee b) \wedge (\neg b \vee a)$
- $a \Rightarrow b = \neg a \vee b$

```
int n;
vector<vector<int>> adj, adj_t;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    used[v] = true;
    for (int u : adj[v]) {
        if (!used[u])
            dfs1(u);
    }
    order.push_back(v);
}

void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : adj_t[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
}
```

```

bool solve_2SAT() {
    order.clear();
    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!used[i])
            dfs1(i);
    }

    comp.assign(n, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1)
            dfs2(v, j++);
    }

    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
}

#define neg(x) (x^1)

void add_disjunction(int a, bool na, int b, bool nb) {
    // na and nb signify whether a and b are to be negated
    a = 2*a ^ na;
    b = 2*b ^ nb;
    int neg_a = neg(a);
    int neg_b = neg(b);
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adj_t[b].push_back(neg_a);
    adj_t[a].push_back(neg_b);
}
```

7.5 Componentes Fortemente Conexas

Uma componente é dita fortemente conexa se podemos sair de u para v e de v para u. Um grafo das componentes fortemente conexas é a condensação de cada componente em um vértice, e isso sempre vira uma DAG. Dá pra usar algoritmos clássicos de resolver problemas em uma DAG dps de condensar o grafo.

Aqui tem um algoritmo que faz isso. De autoria do grande Wevton Santana, o inconfiável.

```
struct SCC {
    int n; // tamanho do grafo.
    int n_cond; // tamanho do grafo condensado.

    vector<vector<int>> edges, rEdges; // arestas e arestas reversas.
    vector<vector<int>> cond_graph; // grafo condensado em uma SCC.
    vector<int> order, comp;
    vector<int> root_nodes, roots; // representantes de cada componente
   e raizes
    vector<bool> vis; // marcar os caras como visitado ou ãno.
    vector<int> in, out; // indegree e outdegree do grafo condensado.

    SCC () {}

    // áj inicializar todos os caras.
    SCC(int n) {
        this->n = n;
        edges.resize(n);
```

```
        rEdges.resize(n);
        roots.resize(n);
        in.resize(n);
        out.resize(n);
    }

    // ou se ãno quiser pode usar init. Fica a gosto do êfregus.
    void init(int n) {
        this->n = n;
        edges.resize(n);
        rEdges.resize(n);
        roots.resize(n);
        in.resize(n);
        out.resize(n);
    }

    // dfs para andar nas arestas e arestas reversas do grafo, coletando o que é ánecessrio.
    void dfs(int u, vector<int> &vet, vector<vector<int>> &graph) {
        vis[u] = true;
        for(auto v: graph[u]) {
            if(!vis[v]) dfs(v, vet, graph);
        }
        vet.push_back(u);
    }

    // adicionar arestas u->v e v->u.
    void add_edge(int u, int v) {
        edges[u].push_back(v);
        rEdges[v].push_back(u);
    }

    // çãfuno para criar a Componente Fortemente Conexa.
    void initSCC() {
        vis.assign(n, false);
        for(int i=0; i<n; i++){
            if(!vis[i]) dfs(i, order, edges);
        }

        reverse(order.begin(), order.end());

        vis.assign(n, false);
        int color = 0;

        for(auto v: order) {
            if(!vis[v]) {
                dfs(v, comp, rEdges);
                root_nodes.push_back(color);
                for(auto u: comp){
                    roots[u] = color;
                }
                color++;
                comp.clear();
            }
        }
        this->n_cond = color;

        cond_graph.resize(n_cond);

        for(int v=0; v<n; v++) {
            for(auto u: edges[v]) {
                if(roots[u] != roots[v]) {
                    cond_graph[roots[v]].push_back(roots[u]);
                    in[roots[u]]++;
                    out[roots[v]]++;
                }
```

```
            }
        }
    }

    // çãfuno de solve para resolver algo usando a scc.
    void solve() {}
};
```

7.6 Stable Marriage

Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. When there are no such pairs of people, the set of marriages is deemed stable.

Someone way before my father was born proved that it is always possible to match everyone if there are equal numbers of men and women, and that all marriages in the end are stable. The algorithm presented below works in $\mathcal{O}(n^2)$.

```
const int MAXN = 510;

queue<int> menPref[MAXN], Q;
int womenPref[MAXN][MAXN];//womenPref[a][b] = c, men b is the c-th
    preference of women a
int womenMarryWith[MAXN];

void test()
{
    int n, in;
    int men, women, other;
    cin >> n;
    rep(i, 0, n)
        womenMarryWith[i] = -1;

    rep(i, 0, n){
        cin >> men;
        men--;
        rep(j, 0, n){
            cin >> in;
            in--;
            menPref[men].push(in);
        }
    }
    rep(i, 0, n){
        cin >> women;
        women--;
        rep(j, 0, n){
            cin >> in;
            in--;
            womenPref[women][in] = j;
        }
    }

    rep(i, 0, n){
        Q.push(i);
    }

    while(!Q.empty()){
        men = Q.front();
        Q.pop();

        if (menPref[men].empty())
            continue;

        women = menPref[men].front();
        menPref[men].pop();
```

```

        if (womenMarryWith[women] == -1){
            womenMarryWith[women] = men;
        }
        else if (womenPref[women][womenMarryWith[women]] > womenPref[women][men]){
            other = womenMarryWith[women];
            womenMarryWith[women] = men;
            Q.push(other);
        }
        else{
            Q.push(men);
        }
    }

    rep(i, 0, n){
        cout << i+1 << "  " << womenMarryWith[i]+1 << "\n";
    }

    rep(i, 0, n){
        while(!menPref[i].empty())
            menPref[i].pop();
    }
}
```

7.7 Planar Graphs

Some theorems related to planar graphs. Most of them constraints conditions between the quantity of vertices v, the quantity of edges e and the quantity of faces f.

- $v - e + f = 2$
- $e \leq 3 * v - 6$
- $D = \frac{f-1}{2*v-5}$
- every planar graph is 4-colourable.
- a really dense(maximal) planar graph has $3 * v - 6$ edges and $2 * v - 4$ faces.

7.8 Dominator Tree

- Dominator : Dominators are defined in a directed graph with respect to a source vertex S. Formally, A node u is said to dominate a node w when related to a source vertex S if all the paths from S to w in the graph must pass through node u.
- Immediate Dominator : A node u is said to be an immediate dominator of a node w (denoted as $idom(w)$) if u dominates w and every other dominator of w dominates u (every vertex have only one immediate dominator).
- Dominator Tree : The edges $\{(idom(w), w) \mid w \epsilon V - \{S\}\}$ forms a directed tree with S being the root of the tree.

Note that only the vertices that are reachable from source vertex in the directed graph are considered here. It is assumed that every vertex in the graph is reachable from the source.

The problem using the structure is Critical Cities, CSES. It is related to find all dominators from n, when S = 1.

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

const int N = 200100;

vector<int> adj[N],tree[N],r_adj[N],bucket[N];
int sdom[N],par[N],dom[N],dsu[N],rot[N];
int in[N],rev[N],cs = 1;
//1-Based directed graph input
```

```
int findx(int u,int x=0)
{
    if(u==dsu[u])return x?-1:u;
    int v = findx(dsu[u],x+1);
    if(v<0)return u;
    if(sdom[rot[dsu[u]]]<sdom[rot[u]])
        rot[u] = rot[dsu[u]];
    dsu[u] = v;
    return x?v:rot[u];
}
void merge(int u,int v){ //Add an edge u-->v
    dsu[v]=u;
}
void dfs0(int u)
{
    rev[cs]=u;
    in[u] = rot[cs] = sdom[cs] = dsu[cs] = cs;
    cs++;
    for(int v:adj[u])
    {
        if(!in[v]){
            dfs0(v);
            par[in[v]]=in[u];
        }
        r_adj[in[v]].pb(in[u]);
    }
}

void build_dominator_tree(int n, int s)
{
    dfs0(s);
    rep(u, n+1, 1)
    {
        for(int v:r_adj[u])
            sdom[u] = min(sdom[u], sdom[findx(v)]);
        if(u > 1)
            bucket[sdom[u]].pb(u);

        for(int v:bucket[u]){
            int w = findx(v);
            if(sdom[v] == sdom[w])
                dom[v] = sdom[v];
            else
                dom[v] = w;
        }
        if(u > 1)
            merge(par[u],u);
    }
    rep(i, 2, n+1)
    {
        if(dom[i]!=sdom[i])
            dom[i]=dom[dom[i]];
        tree[rev[i]].pb(rev[dom[i]]);
        tree[rev[dom[i]]].pb(rev[i]);
    }
}

int pai[N];

void dfs(int u, int p)
{
    for(int v:tree[u])
    {
        if(v == p)
            continue;
        pai[v] = u;
```

```
        dfs(v,u);
    }
}

void test(){
    int n, m;
    cin >> n >> m;
    while(m--)
    {
        int u, v;
        cin >> u >> v;
        adj[u].pb(v);
    }
    build_dominator_tree(n,1);
    dfs(1,1);
    int at = n;
    vector<int> resp;
    while(at)
    {
        resp.pb(at);
        at = pai[at];
    }
    cout << resp.size() << '\n';
    sort(all(resp));
    for(int x:resp)
        cout << x << ' ';
    cout << '\n';
}

int32_t main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t = 1;
    // cin >> t;
    while(t--) test();
    return 0;
}
```

7.9 Chinese Postman Problem

The problem is, given a undirected weighted graph, find a tour that visits each edge at least one time, and the sum of weights from visited edges is minimum as possible. $O(2^n.n)$ where n is the number of vertices. (Original Problem - UVa 10296 - Jogging Trails)

```
#include <stdio.h>
#include <string.h>
#define min(x, y) ((x) < (y) ? (x) : (y))
int map[16][16], odd[16];
void floyd(int n) {
    int i, j, k;
    for(k = 1; k <= n; k++)
        for(i = 1; i <= n; i++)
            for(j = 1; j <= n; j++)
                map[i][j] = min(map[i][j], map[i][k]+map[k][j]);
}
int dp[1<<16];
int build(int pN, int ot) {
    if(pN == 0)
        return 0;
    if(dp[pN] != -1)
        return dp[pN];
    int i, j, tmp;
    dp[pN] = 0xffffff;
    for(i = 0; i < ot; i++) {
```

```
        if(pN&(1<<i)) {
            for(j = i+1; j < ot; j++) {
                if(pN&(1<<j)) {
                    tmp = build(pN-(1<<i)-(1<<j), ot);
                    dp[pN] = min(dp[pN], tmp+map[odd[i]][odd[j]]);
                }
            }
            break;
        }
    }
    return dp[pN];
}
int main() {
    int n, m, x, y, w;
    while(scanf("%d", &n) == 1 && n) {
        scanf("%d", &m);
        memset(map, 63, sizeof(map));
        memset(dp, -1, sizeof(dp));
        int sum = 0, deg[16] = {};
        while(m--) {
            scanf("%d %d %d", &x, &y, &w);
            map[x][y] = min(map[x][y], w);
            map[y][x] = min(map[y][x], w);
            deg[x]++, deg[y]++;
            sum += w;
        }
        floyd(n);
        int ot = 0;
        for(int i = 1; i <= n; i++)
            if(deg[i]&1)
                odd[ot++] = i;
        printf("%d\n", sum+build((1<<ot)-1, ot));
    }
    return 0;
}
```

7.10 Some Bridge related topics

7.10.1 Finding Bridges

just check if the lowest back edge of the bottom most vertex off an edge in a dfs tree entered in a time later than the time the top most vertex entered the dfs.

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

//template

const int N = 200100;

int cs = 1;
int id[N],low[N];
int vis[N];
vector<int> adj[N];
vector<pii> bridges;
void dfs(int u, int p) {
  vis[u] = 1;
  id[u] = low[u] = cs++;

  int sz = 0;
  for (int v : adj[u]) {
    if (v == p)
            continue;

    if (!vis[v]) {
      dfs(v, u);
```

```
      low[u] = min(low[u], low[v]);
      sz++;

      // Check if the current u-v is an bridge
      if (low[v] > id[u])
                bridges.eb(u,v);
    } else {
      low[u] = min(low[u], id[v]);
    }
  }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
  int n,m;
    cin >> n >> m;
    rep(i, 1, n+1){
        adj[i].clear();
        vis[i] = 0;
    }
    bridges.clear();
    cs = 1;
    rep(i, 0, m)
    {
        int u, v;
        cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }

    dfs(1, 1);

    cout << bridges.size() << "\n";
    for(auto [u,v]:bridges)
        cout << u << ' ' << v << '\n';
}
```

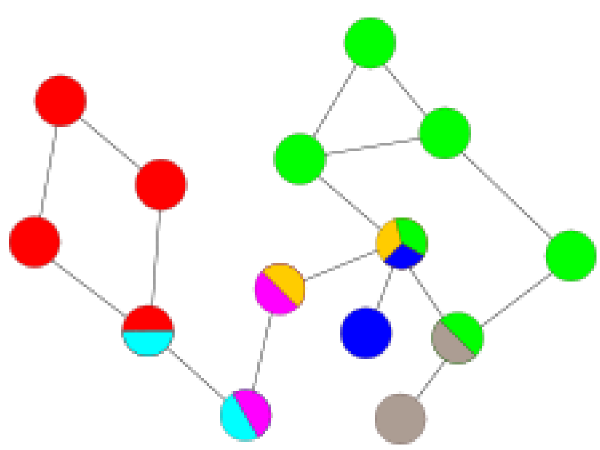
7.10.2 Articulation Points

Points which when removed create multiple graphs. The main idea is the same of finding bridges, utilize the low to check somethings in the dfs tree.

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

//template

const int N = 200100;

int cs = 1;
int id[N],low[N];
int vis[N],ap[N];
vector<int> adj[N];

void dfs(int u, int p) {
  vis[u] = 1;
  id[u] = low[u] = cs++;

  int sz = 0;
  for (int v : adj[u]) {
    if (v == p)
            continue;

    if (!vis[v]) {
      dfs(v, u);
```

```
   low[u] = min(low[u], low[v]);
   sz++;

   // Check if the current u is an articualation point
   if (low[v] >= id[u] && p != u)
        ap[u] = 1;
  } else {
   low[u] = min(low[u], id[v]);
  }
 }
 if (p == u && sz > 1)
    ap[u] = 1;
}

int main() {
 int n,m;
  cin >> n >> m;
  rep(i, 1, n+1){
    adj[i].clear();
    ap[i] = vis[i] = 0;
  }
  cs = 1;
  rep(i, 0, m)
  {
    int u, v;
    cin >> u >> v;
    adj[u].pb(v);
    adj[v].pb(u);
  }

  dfs(1, 1);

  int rs = 0;
  rep(i, 1, n+1)
    rs += ap[i];
  cout << rs << '\n';
}
```

7.10.3 Block Cut Tree



Construct a tree of bi-connected components of the graph. Some of the approaches related to this structure needs you to do some updates in the parent of the node.

The application is a problem with a lot of queries each query will give 3 vertices a b and c and ask if there is a path that goes from a to b, without passing in c. The answer is no if a ==b, a == c or c is an articulation point and is in the path of the block cut tree going from a to b.

```
#include <bits/stdc++.h>
using namespace std;
//template
const int N = 200100;
const int K = 20;
int cs = 1, qt_comp = 0;
int id[N],low[N], rot[N];
```

```
int vis[N],ap[N];
vector<int> adj[N], tree[N];
vector<int> comp[N], stk;
void dfs0(int u, int p) {
  vis[u] = 1;
  id[u] = low[u] = cs++;
    stk.pb(u);
  for (int v : adj[u]) {
    if (v == p)
            continue;

    if (!vis[v]) {
      dfs0(v, u);
      low[u] = min(low[u], low[v]);
      // Check if the current u is an articualation point
      if (low[v] >= id[u] && p != u)
            {
                ap[u] = (id[u] > 1 || id[v] > 2);
                comp[qt_comp].pb(u);
                while (comp[qt_comp].back() != v) {
                    comp[qt_comp].pb(stk.back());
                    stk.pop_back();
                }
                qt_comp++;
            }
    } else {
      low[u] = min(low[u], id[v]);
    }
  }
}
int n_tree;
// return the size of the tree
int build_block(int n)
{
    dfs0(1, -1);
  // Build the block-cut tree
    int node_id = 1;
    rep(i, 1, n+1)
        if (ap[i])
            rot[i] = node_id++;

    rep(j, 0, qt_comp) {
      int node = node_id++;
      for (int u : comp[j])
        if (!ap[u]) {
          rot[u] = node;
        } else {
          tree[node].pb(rot[u]);
          tree[rot[u]].pb(node);
        }
    }
  return node_id-1;
}

int up[N][K], in[N], out[N],tmp = 0;;
void dfs(int u, int p)
{
    in[u] = tmp++;
    up[u][0] = p;
    rep(j, 1, K) {
        up[u][j] = up[up[u][j - 1]][j - 1];
    }
    for (auto v : tree[u]) {
        if (v == p) continue;
        dfs(v, u);
    }
```

```
    out[u] = tmp-1;
}
bool is_ancestor(int u, int v)
{
    return (in[u] <= in[v] && out[v] <= out[u]);
}
int calc_lca(int u, int v)
{
    if(is_ancestor(u,v))
        return u;

    if(is_ancestor(v,u))
        return v;
    rep(j, (int)K, 0)
    {
        if(!is_ancestor(up[u][j],v))
            u = up[u][j];
    }
    return up[u][0];
}
int main() {
  int n, m, q;
  cin >> n >> m >> q;

  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].pb(v);
    adj[v].pb(u);
  }

  n_tree = build_block(n);

  // Check whether the node z is on path (x, y)
  auto on_path = [&](int x, int y, int z) {
    int lca = calc_lca(x, y);
    int lca1 = calc_lca(x, z);
    int lca2 = calc_lca(y, z);

    if (lca == z || (lca1 == lca && lca2 == z) ||
        (lca2 == lca && lca1 == z)) {
      return true;
    }
    return false;
  };

  dfs(1, 1);

  for (int i = 0; i < q; i++) {
    int a, b, c;
    cin >> a >> b >> c;
    // The path does not exist in two cases:
    // 1) a == c or b == c
    // 2) c is a cutpoint and it lies on path between a and b in block-cut
    // tree
    if (a == c || b == c ||
        (ap[c] && on_path(rot[a], rot[b], rot[c]))) {
      cout << "NO\n";
    } else {
      cout << "YES" << '\n';
    }
  }
}
```

7.10.4 Two Edge Connected Components

Decompose the graph in Components connected by at least two edges. Can be used to create a tree as well. a "block cut"decomposition, but with edges instead of vertices.

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

// template

typedef long long ll;

const int N = 200100;

int cs = 1;  // Time of entry in u
int qt_comp;    // Number of strongly connected components
int id[N];
int low[N],rot[N];  // Lowest ID in u's subtree in DFS tree
vector<int> adj[N];
vector<int> comp[N], stk;
void dfs(int u, int p = -1) {
  id[u] = low[u] = cs++;
  stk.pb(u);
    // careful with multiple edges
    int did = 0;

  for (int v : adj[u]) {
    if (v == p && !did) {
      did = 1;
      continue;
    }
    if (!id[v]) {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
    } else {
      low[u] = min(low[u], id[v]);
    }
  }

  if (low[u] == id[u]) {
    while (stk.back() != u) {
            rot[stk.back()] = qt_comp;
      comp[qt_comp].pb(stk.back());
      stk.pop_back();
    }
        rot[stk.back()] = qt_comp;
    comp[qt_comp++].pb(stk.back());
    stk.pop_back();
  }
}

int main() {
  int n, m;
  cin >> n >> m;
    rep(i, 0, m) {
    int x, y;
    cin >> x >> y;
    adj[x].push_back(y);
    adj[y].push_back(x);
  }

    rep(u, 0, n)
    if (!id[u])
            dfs(u);


  cout << qt_comp << '\n';
```

```
    rep(i, 0, qt_comp) {
    cout << comp[i].size() << ' ';
    for (int u : comp[i]) { cout << u << ' '; }
    cout << '\n';
  }
}
```

Capítulo VIII

Trees

8.1 Diameter

Finding the diameter of the tree. Just do two DFS's, where in the first one u find the longest guy u can reach, and then find the other guy starting from the one you found before.

```
const int N = 200100;

vector<int> adj[N];

pair<int, int> dfs_diameter(int u, int p, int dep)
{
    pair<int, int> best;
    best = {dep, u};
    for(int v : adj[u])
        if(v != p)
            best = max(best, dfs_diameter(v, u, dep+1));
    return best;
}


void test()
{
    int n;
    cin >> n;
    for(int i = 0, u, v, t; i < n-1; i++)
    {
        cin >> u >> v;
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    int at = dfs_diameter(1, 1, 1).second;
    cout << dfs_diameter(at, at, 1).first << '\n';
}
```

8.2 LCA

A way to find the lowest common ancestor of two vertex (a, b) in a tree. Needs a Sparse Table.

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

const int N = 300100;
const int K = 20;
vector<int> adj[N];
int pai[N][K], in[N], out[N], cs = 0;

void pre_dfs(int u, int p)
{
    in[u] = cs++;
    pai[u][0] = p;
    for(int i = 1; i < K; i++){
        pai[u][i] = pai[pai[u][i-1]][i-1];
    }
```

```
    for(int v : adj[u])
        if(v != p)
            pre_dfs(v, u);
    out[u] = cs++;
}

// u is ancestor of v ?
bool is_ancestor(int u, int v)
{
    return in[u] <= in[v] && out[v] <= out[u];
}

int calc_lca(int a, int b)
{
    if(is_ancestor(a, b))
        return a;
    if(is_ancestor(b, a))
        return b;
    for(int i = K-1; i >=0; i--)
    {
        if(!is_ancestor(pai[b][i], a))
            b = pai[b][i];
    }
    return pai[b][0];
}



void test()
{
    int n, q;
    cin >> n;
    for(int i = 0, u, v; i < n-1; i++)
    {
        cin >> u >> v;
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    pre_dfs(1, 1);
    cin >> q;
    while(q--)
    {
        int a, b;
        cin >> a >> b;
        cout << calc_lca(a, b) << '\n';
    }
}

int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    test();

    return 0;
}
```

Version that Sabino likes

```
/*
    Indexado de 1

    Alteracoes:

    As vezes mudar como calcula o calc do lca
*/
```

```
struct LCA{
    vector<vector<int>> graph;
    int n;
    // entrada, saida, e euler tour
    int timer=0;
    vector<int> tin,tout,euler;
    // se vai diferenciar ou n o tin do tout
    int flag;
    // distancia de cada cara pra raiz
    vector<int> dist;
    // tabela de ancestrais
    vector<vector<int>> pai;
    // raiz
    int r;
    int TETO;

    LCA(vector<vector<int>> &graph,int n,int flag=0,int r=1){
        this->n = n;
        this->graph = graph;
        tin.resize(n+10);
        tout.resize(n+10);
        this->flag = flag;
        euler.resize((flag ? 2*n+10 : n+10));
        dist.resize(n+10);
        int t=0;
        int d=1;
        while(d <= n){
            t++;
            d<<=1;
        }
        TETO = t;
        pai.resize(n+10,vector<int>(TETO));
        this->r = r;

        build(r,r);
    }

    void build(int u,int ant){
        if(u == ant) dist[u]=0;
        else dist[u]=dist[ant]+1;
        tin[u]=++timer;
        pai[u][0]=ant;
        // construir tabela de ancestrais
        for(int i=1; i < TETO; i++) {
            pai[u][i]=pai[pai[u][i-1]][i-1];
        }
        for(auto v: graph[u]){
            if(v == ant) continue;
            build(v,u);
        }
        tout[u]=(flag ? ++timer : timer);
    }

    // x eh lca de y?
    bool lca(int x,int y){
        return tin[x] <= tin[y] && tout[x] >= tout[y];
    }

    // calcula alguma coisa, deixei como se fosse dist, entre x e y
    int calc(int x,int y){
        if(lca(x,y)) return dist[y]-dist[x];
        else if(lca(y,x)) return dist[x]-dist[y];
        int z=x;
        for(int i=TETO-1; i>=0; i--){
            if(!lca(pai[z][i],y)) z=pai[z][i];
        }
```

```
        z=pai[z][0];
        return dist[x]+dist[y]-2*dist[z];
    }
};
```

8.3 Euler tour tree

No code here, use lca code. remember u can calc paths using the ett. There are two types.

Incrementing *out*, and not incrementing *out*. We can use the first one as a kind of bit with $f(x)$ in $in[x]$ and $f(x)^{-1}$ in $out[x]$

8.4 Small to Large

Not much to say, the code is always different, so lets just keep the USACO text:

Naive Solution

Suppose that we want merge two sets a and b of sizes n and m, respectively. One possibility is the following:

`for (int x : b) a.insert(x);`

which runs in $\mathcal{O}(m \log(n+m))$ time, yielding a runtime of $\mathcal{O}(N^2 \log N)$ in the worst case. If we instead maintain a and b as sorted vectors, we can merge them in $\mathcal{O}(n+m)$ time, but $\mathcal{O}(N^2)$ is also too slow.

Better Solution

With just one additional line of code, we can significantly speed this up.

`if (a.size() < b.size()) swap(a, b);for (int x : b) a.insert(x);`

Note that swap exchanges two sets in $\mathcal{O}(1)$ time. Thus, merging a smaller set of size m into the larger one of size n takes $\mathcal{O}(m \log n)$ time.

Claim: The solution runs in $\mathcal{O}(N \log^2 N)$ time.

Proof: When merging two sets, you move from the smaller set to the larger set. If the size of the smaller set is X, then the size of the resulting set is at least $2X$. Thus, an element that has been moved Y times will be in a set of size at least 2^Y, and since the maximum size of a set is N (the root), each element will be moved at most $\mathcal{O}(\log N)$ times.

Generalizing

We can also merge other standard library data structures such as std::map or std:unordered_map in the same way. However, std::swap does not always run in $\mathcal{O}(1)$ time. For example, swapping std::arrays takes time linear in the sum of the sizes of the arrays, and the same goes for GCC policy-based data structures such as __gnu_pbds::tree or __gnu_pbds::gp_hash_table.

To swap two policy-based data structures a and b in $\mathcal{O}(1)$ time, use a.swap(b) instead. Note that for standard library data structures, swap(a,b) is equivalent to a.swap(b).

8.5 HLD

You are given a tree consisting of n nodes. The nodes are numbered 1,2,...,n. Each node has a value.

Your task is to process following types of queries:

- Change the value of node s to x
- Find the maximum value on the path between nodes a and b.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> pii;
#define lef(x) (x << 1)
#define rig(x) (lef(x) | 1)
const int N = 200100;

class Segtree {
    public:
    const ll emp = 0; //NULL VALUE
    vector<ll> tree;
    Segtree(){}
    ll f(ll a, ll b){return max(a,b);}//function

    void updt(int p, int tl, int tr, int k, ll v)
    {
```

```
        if(tl == tr){
            tree[p] = v;
            return;
        }
        int tmid = (tl+tr) >> 1;
        if (tmid < k)
            updt(rig(p), tmid+1, tr, k, v);
        else
            updt(lef(p), tl, tmid, k, v);
        tree[p] = f(tree[lef(p)], tree[rig(p)]);

    }

    ll query(int p, int tl, int tr, int l, int r)
    {
        if(l > r)
            return emp;
        if(tl == l && tr == r)
            return tree[p];
        int tmid = (tl+tr) >> 1;
        return f(query(lef(p), tl, tmid, l, min(r,tmid)),
            query(rig(p), tmid+1, tr, max(l,tmid+1), r));
    }

};
Segtree tree[N];

vector<int> adj[N];
int depth[N], hd[N], sz[N], pai[N], sub_sz[N], pos[N];
ll a[N];

void dfs(int u,int p = -1)
{
    pai[u] = p;
    sub_sz[u] = 1;
    if(p!=-1)
        depth[u] = depth[p] + 1;

    for(int v: adj[u])
    {
        if(v != p){
            dfs(v, u);
            sub_sz[u] += sub_sz[v];
        }
    }
}
void dfs2(int u,int p = 1)
{
    for(int v: adj[u])
    {
        if(v == p)
            continue;
        if(sub_sz[u] <= 2*sub_sz[v]){
            hd[v] = hd[u];
            pos[v] = ++sz[hd[u]];
            dfs2(v, u);
        }else
        {
            hd[v] = v;
            sz[v] = 1;
            pos[v] = 1;
            dfs2(v, u);
        }
    }
    if(u == hd[u])
        tree[u].tree.resize(sz[u] << 2,0);
```

```
}
void build_hld(int n)
{
    dfs(1);
    hd[1] = sz[1] = pos[1] =  1;
    dfs2(1);
    for(int i = 1; i <= n; i++)
    {
        tree[hd[i]].updt(1,1,sz[hd[i]],pos[i],a[i]);
    }
}
ll query(int a,int b)
{
    ll ans = 0;
    while(hd[a] != hd[b])
    {
        // cout << a << ' '<< b << '\n';
        if(depth[hd[a]] > depth[hd[b]])
            swap(a,b);
        ans = max(ans, tree[hd[b]].query(1,1,sz[hd[b]],pos[hd[b]],pos[b
   ]));
        b = pai[hd[b]];
    }
    if(pos[a] > pos[b])
        swap(a,b);
    ans = max(ans, tree[hd[b]].query(1,1,sz[hd[b]],pos[a],pos[b]));
    return ans;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n, q;
    cin >> n >> q;

    for(int i = 1; i <= n; i++)
        cin >> a[i];
    for(int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    build_hld(n);
    while(q-- )
    {
        int t;
        cin >> t;
        if(t == 2)
        {
            int u, v;
            cin >> u >> v;
            cout << query(u, v) << "\n";
        }else
        {
            int u;
            ll x;
            cin >> u >> x;
            tree[hd[u]].updt(1,1,sz[hd[u]],pos[u],x);
            a[u] = x;
        }
    }
    return 0;
}
```

8.5.1 Can you answer these queries VII

Given are a tree with n vertices, each vertice has a weight x_i, and two type of operations. Operation 1 asks for the maximum contiguous sum between two vertices a and b. Operation 2 update the values of vertices between a and b to c. We have to process Q operations.

The problem can be solved with HLD in $O(Q(log(n))^2)$. Original Problem (SPOJ - GSS7).

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

using namespace std;

const int N = int(2e5 + 10);
typedef long long ll;

vector< vector< int > > adj;
vector< int > parent, depth, heavy, head, pos;
int cur_pos;
int n;
int carry[N];
int idx[N];
ll arr[N];

int dfs(int v){
  int size = 1;
  int max_c_size = 0;

  for(int c: adj[v]){
    if(c != parent[v]){
      parent[c] = v, depth[c] = depth[v] + 1;
      int c_size = dfs(c);
      size += c_size;

      if(c_size > max_c_size){
        max_c_size = c_size, heavy[v] = c;
      }
    }
  }

  return size;
}

void decompose(int v, int h){
    idx[cur_pos] = v;
    head[v] = h, pos[v] = cur_pos++;

    if(heavy[v] != -1){
        decompose(heavy[v], h);
    }

    for(int c: adj[v]){
        if(c != parent[v] && c != heavy[v]){
            decompose(c, c);
        }
    }
}

class Node{
public:
  ll pref, suf, best, total;
};

const ll inf = ll(1e10);
ll lazzy[N << 2];
int lazzy1[N << 2];
```

```
Node segtree[N << 2];

void shift(int node, ll el){
  int lf = node << 1;
  int rg = lf + 1;

  ll val = max(0LL, lazzy[node] * el);
  segtree[node].pref = val;
  segtree[node].suf = val;
  segtree[node].best = val;
  segtree[node].total = lazzy[node] * el;

  lazzy[lf] = lazzy[node];
  lazzy[rg] = lazzy[node];
  lazzy1[lf] = lazzy1[rg] = 1;

  lazzy1[node] = 0;
}

Node merge(Node lf, Node rg){
  Node r;

  if(lf.total == -inf){
    return rg;
  }

  if(rg.total == -inf){
    return lf;
  }

  r.pref = max(lf.pref, lf.total + rg.pref);
  r.suf = max(rg.suf, rg.total + lf.suf);
  r.best = max({r.pref, r.suf, lf.best, rg.best, lf.suf + rg.pref});
  r.total = lf.total + rg.total;

  return r;
}

void build(int node, int i, int j){
  if(i == j){
    segtree[node].total = arr[idx[i] + 1];
    segtree[node].suf = max(0LL, arr[idx[i] + 1]);
    segtree[node].pref = max(0LL, arr[idx[i] + 1]);
    segtree[node].best = max(0LL, arr[idx[i] + 1]);
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    build(lf, i, mid);
    build(rg, mid + 1, j);

    segtree[node] = merge(segtree[lf], segtree[rg]);
  }
}

void update(int node, int i, int j, int l, int r, ll val){
  if(lazzy1[node]){
    shift(node, j - i + 1);
  }

  if(i > r || j < l) return;

  if(l <= i && j <= r){
    lazzy[node] = val;
    lazzy1[node] = 1;
```

```
      shift(node, j - i + 1);
    }else{
      int mid = (i + j) >> 1;
      int lf = node << 1;
      int rg = lf + 1;

      update(lf, i, mid, l, r, val);
      update(rg, mid + 1, j, l, r, val);

      segtree[node] = merge(segtree[lf], segtree[rg]);
    }
}

Node segment_tree_query(int node, int i, int j, int l, int r){
  if(lazzy1[node]){
    shift(node, j - i + 1);
  }

  if(l > r) return {-inf, -inf, -inf, -inf};

  if(i > r || j < l){
    return {-inf, -inf, -inf, -inf};
  }

  if(l <= i && j <= r){
    return segtree[node];
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    return merge(segment_tree_query(lf, i, mid, l, r),
      segment_tree_query(rg, mid + 1, j, l, r));
  }
}

void init(){
  int n = adj.size();
  parent = vector< int >(n);
  depth = vector< int >(n);
  heavy = vector< int >(n, -1);
  head = vector< int >(n);
  pos = vector< int >(n);

  cur_pos = 1;
  dfs(0);
  decompose(0, 0);
  build(1, 1, n);
}

Node merge2(Node lf, Node rg){
  Node r;
  r.best = max({lf.best, rg.best, rg.pref + lf.pref});

  return r;
}

ll query(int a, int b){
  Node res[2] = {{0LL, 0LL, 0LL, 0LL}, {0LL, 0LL, 0LL, 0LL}};
  int c = 0;

  for(; head[a] != head[b] ; b = parent[head[b]]){
    if(depth[head[a]] > depth[head[b]]){
      swap(a, b);
      c = !c;
    }
```

```
    res[c] = merge(segment_tree_query(1, 1, n, pos[head[b]], pos[b]),
   res[c]);
  }

  if(pos[a] > pos[b]){
    c = !c;
    swap(a, b);
  }

  res[c] = merge(segment_tree_query(1, 1, n, pos[a], pos[b]), res[c]);
  res[c] = merge2(res[c], res[!c]);

  return res[c].best;
}

void updatet(int a, int b, ll val){
  for(; head[a] != head[b] ; b = parent[head[b]]){
    if(depth[head[a]] > depth[head[b]]){
      swap(a, b);
    }
    update(1, 1, n, pos[head[b]], pos[b], val);
  }

  if(pos[a] > pos[b]){
    swap(a, b);
  }
  update(1, 1, n, pos[a], pos[b], val);
}

int main(){
  scanf("%d", &n);
  for(int i = 1 ; i <= n ; i++){
    scanf("%lld", &arr[i]);
  }

  adj.resize(n);

  for(int i = 1 ; i < n ; i++){
    int ui, vi;
    scanf("%d %d", &ui, &vi);
    ui--, vi--;
    adj[ui].push_back(vi);
    adj[vi].push_back(ui);
  }

  int q;
  init();
  scanf("%d", &q);

  while(q--){
    int t;
    scanf("%d", &t);

    if(t == 1){
      int a, b;
      scanf("%d %d", &a, &b);
      a--, b--;
      printf("%lld\n", query(a, b));
    }else{
      int a, b;
      ll c;
      scanf("%d %d %lld", &a, &b, &c);
      a--, b--;
      updatet(a, b, c);
    }
  }
```

```
  return 0;
}
```

8.5.2 Can you answer these queries VI

Given a tree with n vertices, each vertice has a color (0 or 1), and two type of operations. Operation 0 asks for how many vertices are connected to a vertice u, when one vertice v is connected to u only if the path between u and v just contain vertices with same color of u inclusive. Operation 1 is to change the color of vertice u (0 to 1 and 1 to 0).

The problem can be solved wih HLD in $O(Q(log(n))^2)$. Original Problem (SPOJ - Can You Answer These Queries VI).

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

using namespace std;

typedef pair< int, int >  pii;
const int N = int(1e5 + 10);
vector< int > adj[N];
vector< int > parent, depth, heavy, head, pos;
int sz[N], idx[N];
pii lazzy[N << 3];
int cur_pos;
pii segtree1[N << 3];
pii segtree2[N << 3];
int n;
int pivot = N - 5;
int cor[N];

int dfs(int v){
  int size = 1;
  int max_c_size = 0;

  for(int c: adj[v]){
    if(c != parent[v]){
      parent[c] = v, depth[c] = depth[v] + 1;
      int c_size = dfs(c);
      size += c_size;

      if(c_size > max_c_size){
        max_c_size = c_size, heavy[v] = c;
      }
    }
  }

  sz[v] = size;

  return size;
}

void decompose(int v, int h){
  idx[cur_pos] = v;
  head[v] = h, pos[v] = cur_pos++;

  // printf("head[%d] = %d, %d\n", v, h, heavy[v]);

  if(heavy[v] != -1){
    decompose(heavy[v], h);
  }

  for(int c: adj[v]){
    if(c != parent[v] && c != heavy[v]){
      decompose(c, c);
    }
  }
```

```
}

pii merge(pii lf, pii rg){
  pii r = {max(lf.first, rg.first), max(lf.second, rg.second)};

  return r;
}

pii merge2(pii lf, pii rg){
  pii r = {max(lf.first, rg.first), max(lf.second, rg.second)};

  return r;
}

void build1(int node, int i, int j){
  if(i == j){
    if(i != 1){
      segtree1[node] = {i, -1};
    }else{
      segtree1[node] = {i, i};//pivot is black and white
    }
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    build1(lf, i, mid);
    build1(rg, mid + 1, j);

    segtree1[node] = merge(segtree1[lf], segtree1[rg]);
  }
}

pii query1(int node, int i, int j, int l, int r){
  if(i > r || j < l){
    return {-1, -1};
  }

  if(l <= i && j <= r){
    return segtree1[node];
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    return merge(
      query1(lf, i, mid, l, r),
      query1(rg, mid + 1, j, l, r)
    );
  }
}

void update1(int node, int i, int j, int pos){
  if(i > pos || j < pos) return;

  if(i == pos && j == pos){
    swap(segtree1[node].first, segtree1[node].second);
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    update1(lf, i, mid, pos);
    update1(rg, mid + 1, j, pos);

    segtree1[node] = merge(segtree1[lf], segtree1[rg]);
```

```
  }
}

void build2(int node, int i, int j){
  if(i == j){
    if(i != 1){
      segtree2[node] = {sz[idx[i]], 0};
    }else{
      segtree2[node] = {sz[0], 0};//pivot
    }
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    build2(lf, i, mid);
    build2(rg, mid + 1, j);

    segtree2[node] = merge2(segtree2[lf], segtree2[rg]);
  }
}

pii add(pii a, pii b){
  return {a.first + b.first, a.second + b.second};
}

void shift(int node){
  int lf = node << 1;
  int rg = lf + 1;

  segtree2[node].first += lazy[node].first;
  segtree2[node].second += lazy[node].second;

  lazy[lf] = add(lazy[lf], lazy[node]);
  lazy[rg] = add(lazy[rg], lazy[node]);

  lazy[node] = {0, 0};
}

pii query2(int node, int i, int j, int l, int r){
  if(lazy[node].first || lazy[node].second){
    shift(node);
  }

  if(j < l || i > r){
    return {0, 0};
  }

  if(l <= i && j <= r){
    return segtree2[node];
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    return merge2(query2(lf, i, mid, l, r), query2(rg, mid + 1, j, l, r)
    );
  }
}

void update2(int node, int i, int j, int l, int r, pii updt){
  if(lazy[node].first || lazy[node].second){
    shift(node);
  }

  if(j < l || i > r){
```

```
    return;
  }

  if(l <= i && j <= r){
    lazzy[node] = updt;

    shift(node);
  }else{
    int mid = (i + j) >> 1;
    int lf = node << 1;
    int rg = lf + 1;

    update2(lf, i, mid, l, r, updt);
    update2(rg, mid + 1, j, l, r, updt);

    segtree2[node] = merge2(segtree2[lf], segtree2[rg]);
  }
}

void init(){
  parent = vector< int >(N, -1);
  depth = vector< int >(N);
  heavy = vector< int >(N, -1);
  head = vector< int >(N);
  pos = vector< int >(N);

  cur_pos = 1;
  dfs(pivot);

  decompose(pivot, pivot);
  build1(1, 1, n);
  build2(1, 1, n);
}

int query(int a){
  int st = cor[a];
  int best = 0;

  while(true){
      if(cor[a] != st) return best;

    pii r = query1(1, 1, n, pos[head[a]], pos[a]);
    // printf("%d, %d\n", r.first, r.second);
    if(st == 0){
      if(r.second != -1){
        // printf("s: %d\n", idx[r.second]);
        // printf("1.%d->%d\n", idx[r.second], idx[r.second]);
        while(best > query2(1, 1, n, r.second + 1, r.second + 1).first);
        return query2(1, 1, n, r.second + 1, r.second + 1).first;
      }else{
          // printf("2.%d->%d %d\n", head[a], a, query2(1, 1, n, pos[a], pos[head[a]]).first);
        while(query2(1, 1, n, pos[head[a]], pos[a]).first < best);
        best = max(best, query2(1, 1, n, pos[head[a]], pos[a]).first);
      }
    }else{
      if(r.first != -1){
          // printf("3.%d->%d %d\n", idx[r.first], idx[r.first + 1], best);
        while(best > query2(1, 1, n, r.first + 1, r.first + 1).second);
        return query2(1, 1, n, r.first + 1, r.first + 1).second;
      }else{
          // printf("4.%d->%d\n", head[a], a);
          while(best > query2(1, 1, n, pos[head[a]], pos[a]).second);
        best = max(best, query2(1, 1, n, pos[head[a]], pos[a]).second);
      }
```

```
    }

    a = parent[head[a]];
  }

  return best;
}

void update(int a){
  int st = cor[a];
  pii insz;

  update1(1, 1, n, pos[a]);

  insz = query2(1, 1, n, pos[a], pos[a]);

  if(st == 0){
    update2(1, 1, n, pos[a], pos[a], {-1, 1});
    // printf("updt: %d, %d (%d)\n", insz.first - 1, insz.second + 1, a)
     ;
  }else{
    update2(1, 1, n, pos[a], pos[a], {1, -1});
  }

  cor[a] = !cor[a];

  a = parent[a];
  int tmp = a;

  while(true){
    pii r = query1(1, 1, n, pos[head[a]], pos[a]);

    if(st == 1){
      if(r.second != -1){
        update2(1, 1, n, r.second, pos[a], {insz.first + 1, 0});
        break;
      }else{
        update2(1, 1, n, pos[head[a]], pos[a], {insz.first + 1, 0});
      }
    }else{
      if(r.first != -1){
        update2(1, 1, n, r.first, pos[a], {0, insz.second + 1});
        break;
      }else{
        update2(1, 1, n, pos[head[a]], pos[a], {0, insz.second + 1});
      }
    }

    a = parent[head[a]];
  }

  a = tmp;

  while(true){
    pii r = query1(1, 1, n, pos[head[a]], pos[a]);

    if(st == 0){
      if(r.second != -1){
        update2(1, 1, n, r.second, pos[a], {-insz.first, 0});
        break;
      }else{
        update2(1, 1, n, pos[head[a]], pos[a], {-insz.first, 0});
      }
    }else{
      if(r.first != -1){
        update2(1, 1, n, r.first, pos[a], {0, -insz.second});
```

```
        break;
      }else{
        update2(1, 1, n, pos[head[a]], pos[a], {0, -insz.second});
      }
    }

    a = parent[head[a]];
  }
}

int main(){
  scanf("%d", &n);

  for(int i = 1 ; i < n ; i++){
    int ui, vi;

    scanf("%d %d", &ui, &vi);

    ui--, vi--;
    adj[ui].push_back(vi);
    adj[vi].push_back(ui);
  }

  adj[pivot].push_back(0);
  n++;

  init();

  int q;

  scanf("%d", &q);

  while(q--){
    int op, u;

    scanf("%d %d", &op, &u);

    u--;

    if(op == 0){
      printf("%d\n", query(u));
    }else{
      update(u);
    }
  }

  return 0;
}
```

8.5.3 HLD with subtree queries

Slower version with subtree queries, uses lazy segtree and template

```
int VALS_IN_EDGES = 0;
// namespace HLD {
  int n;
    vector<int> adj[N];
  int par[N], root[N], dep[N], sz[N], ti;
  int pos[N];
    vector<int> rpos; // rpos not used but could be useful
  void add_edge(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
  void dfsSz(int x) {
    sz[x] = 1;
    for(auto y:adj[x]) {
      par[y] = x; dep[y] = dep[x]+1;
      adj[y].erase(find(all(adj[y]),x)); /// remove parent from adj list
```

```
      dfsSz(y); sz[x] += sz[y];
      if (sz[y] > sz[adj[x][0]]) swap(y,adj[x][0]);
    }
  }
  void dfsHld(int x) {
    pos[x] = ti++; rpos.pb(x);
    for(auto y:adj[x]) {
      root[y] = (y == adj[x][0] ? root[x] : y);
      dfsHld(y);
        }
  }
  void init(int nn, int r = 1) {
        n = nn;
    par[r] = dep[r] = ti = 0;
        dfsSz(r);
    root[r] = r;
        dfsHld(r);
        // build(1, 1, n);
  }
  int lca(int x, int y) {
    for (; root[x] != root[y]; y = par[root[y]])
      if (dep[root[x]] > dep[root[y]]) swap(x,y);
    return dep[x] < dep[y] ? x : y;
  }
  int dist(int x, int y) { // # edges on path
    return dep[x]+dep[y]-2*dep[lca(x,y)];
    }
  template <class BinaryOp>
  void processPath(int x, int y, BinaryOp op) {
    for (; root[x] != root[y]; y = par[root[y]]) {
      if (dep[root[x]] > dep[root[y]])
                swap(x,y);
      op(pos[root[y]],pos[y]);
        }
    if (dep[x] > dep[y])
            swap(x,y);
    op(pos[x]+VALS_IN_EDGES,pos[y]);
  }
  void modifyPath(int x, int y, int v) {
    processPath(x,y,[&v](int l, int r) {
      updt(1, 1, n, l, r, v); });
    }
    void modifyVert(int x, int v) {
    updt(1,1,n,pos[x],pos[x],v);
    }
  ll queryPath(int x, int y) {
    ll res = 0; processPath(x,y,[&res](int l, int r) {
      res = f(res,query(1, 1, n, l, r)); });
    return res;
    }
  void modifySubtree(int x, int v) {
    updt(1,1,n,pos[x]+VALS_IN_EDGES,pos[x]+sz[x]-1,v);
    }
```

8.6 Centroid Decomposition

Always about path problems.

There are two types of centroid solving. Do some computation during centroid and use it. Or keep some data for each node and make updates in $O(log(N))$.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> pii;
const int N = 200100;
```

```
// KEEP
vector<int> adj[N];
int sz[N], did[N];
void dfs(int u,int p )
{
    sz[u] = 1;
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        dfs(v, u);
        sz[u] += sz[v];
    }
}

int centroid(int u,int p, int sub_sz)
{
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        if(2*sz[v] >= sub_sz)
            return centroid(v, u, sub_sz);
    }
    return u;
}
// SOLVE PROBLEM
int k, cs = 1;
ll mp[N], vis[N];
void push(int x)
{
    if(vis[x] != cs)
    {
        vis[x] = cs;
        mp[x] = 0;
    }
}
ll calc(int u, int p,int dep)
{
    ll rs = 0;
    if(dep <= k){
        push(k-dep);
        rs = mp[k-dep];
    }
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        rs += calc(v, u, dep+1);
    }
    return rs;
}
void put(int u, int p,int dep)
{
    push(dep);
    mp[dep]++;
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        put(v, u, dep+1);
    }
}

//DECOMPOSITION
ll decompose(int u)
```

```
{
    dfs(u, u);
    u = centroid(u, u, sz[u]);

  //SOLVE PROBLEM
    cs++;
    ll rs = 0;
    for(int v:adj[u])
    {
        if(did[v])
            continue;
        rs += calc(v, u, 1);
        put(v, u, 1);
    }
    push(k);
    rs += mp[k];

  //PROPAGATE DECOMPOSITION
    did[u] = 1;

    for(int v:adj[u])
    {
        if(did[v])
            continue;
        rs += decompose(v);
    }
    return rs;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n;
    cin >> n >> k;

    for(int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    cout << decompose(1) << '\n';
    return 0;
}
```

Version that Sabino likes

```
/*
    Indexado de 1
*/
struct Centroid{
    // o pai de cada cara no centroide
    vector<int> p;
    // visitado para construir o centroide
    vector<bool> vis;
    // tam da subarvore
    vector<int> tam;
    // grafo original
    vector<vector<int>> graph;
    int n;
    // raiz do centroide
    int r;

    // passa com o grafo ja lido
    Centroid(vector<vector<int>> &graph,int n){
```

```
        this->graph = graph;
        tam.resize(n+10);
        vis.resize(n+10);
        p.resize(n+10);
        this->n = n;
        r=build(1);
    }

    // constroi o centroide
    int build(int u){
        set_tam(u,u);
        while(1){
            int flag=1;
            for(auto v: graph[u]){
                if(vis[v]) continue;
                if(tam[v]*2 > tam[u]){
                    tam[u]-=tam[v];
                    tam[v]+=tam[u];
                    u=v;
                    flag=0;
                    break;
                }
            }
            if(flag) break;
        }
        vis[u]=1;
        p[u]=u;
        for(auto v: graph[u]){
            if(vis[v]) continue;
            int x=build(v);
            p[x]=u;
        }
        return u;
    }

    void set_tam(int u,int ant){
        tam[u]=1;
        for(auto v: graph[u]){
            if(v == ant || vis[v]) continue;
            set_tam(v,u);
            tam[u]+=tam[v];
        }
    }
};
```

Example 1: Given a tree of n nodes, your task is to count the number of distinct paths that consist of exactly k edges.

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

const int N = 400100;
vector<int> adj[N];
int sz[N], did[N];
int pass = 0;
void dfs(int u,int p )
{
    sz[u] = 1;
    // cout << pass++ << '\n';
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        dfs(v, u);
        sz[u] += sz[v];
    }
}
```

```

int centroid(int u,int p, int sub_sz)
{
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        if(2*sz[v] >= sub_sz)
            return centroid(v, u, sub_sz);
    }
    return u;
}
int k, cs = 1;
int cont[N], vis[N];
void push(int x)
{
    if(vis[x] != cs)
    {
        vis[x] = cs;
        cont[x] = 0;
    }
}
int calc(int u, int p,int dep)
{
    ll rs = 0;
    if(dep <= k){
        push(k-dep);
        rs = cont[k-dep];
    }else
        return 0;
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        rs += calc(v, u, dep+1);
    }
    return rs;
}
void put(int u, int p,int dep)
{
    push(dep);
    cont[dep]++;
    if(dep > k)
        return;
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        put(v, u, dep+1);
    }
}
ll decomp(int u)
{
    // cout << u << " <<< \n";
    dfs(u, u);

    // cout << u << " <<< \n";
    u = centroid(u, u, sz[u]);
    cs++;
    ll rs = 0;
    for(int v:adj[u])
    {
        if(did[v])
            continue;
        rs += calc(v, u, 1);
        put(v, u, 1);
```

```
    }
    push(k);
    rs += cont[k];
    if(rs == 0)
        return 0;
    // cout << u << ' ' << rs << '\n';
    did[u] = 1;
    for(int v:adj[u])
    {
        if(did[v])
            continue;
        rs += decomp(v);
    }
    return rs;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n;
    cin >> n >> k;

    for(int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    cout << decomp(1) << '\n';
    return 0;
}
```

Example 2: Xenia has a tree consisting of n nodes. We will consider the tree nodes indexed from 1 to n. We will also consider the first node to be initially painted red, and the other nodes — to be painted blue.

execute queries of two types:

- paint a specified blue node in red;
- calculate which red node is the closest to the given one and print the shortest distance to the closest red node.

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

const int N = 400100;
vector<int> adj[N];
int sz[N], did[N];
int pass = 0;
void dfs(int u,int p )
{
    sz[u] = 1;
    // cout << pass++ << '\n';
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        dfs(v, u);
        sz[u] += sz[v];
    }
}

int centroid(int u,int p, int sub_sz)
{
    for(int v:adj[u])
```

```
    {
        if(v == p || did[v])
            continue;
        if(2*sz[v] >= sub_sz)
            return centroid(v, u, sub_sz);
    }
    return u;
}
int k, cs = 1;
map<pair<int,int>, int> dist;
int best[N], pai[N];
void dfs_calc(int u,int p, int orig, int dep)
{
    dist[{orig, u}] = dep;
    for(int v:adj[u])
    {
        if(v == p || did[v])
            continue;
        dfs_calc(v, u, orig, dep+1);
    }
}

void decomp(int u, int p)
{
    dfs(u, u);
    int at = centroid(u, u, sz[u]);
    pai[at] = p;
    dfs_calc(at, at, at, 0);
    if(u == p)
        pai[at] = at;
    // cout << at << ' ' << pai[at] << "<<<\n";
    did[at] = 1;
    for(int v:adj[at])
    {
        if(did[v])
            continue;
        decomp(v, at);
    }
}


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
        best[i] = n+1;
    for(int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    decomp(1, 1);
    int u = 1;
    best[u] = 0;
    int x = u;
    while(pai[u] != u)
    {
        u = pai[u];
        best[u] = min(best[u], dist[{u, x}]);
    }
    while(m--)
```

```
    {
        int t, u;
        cin >> t >> u;
        // cout << t << ' ' << u << '\n';
        if(t == 2)
        {
            int rs = best[u];
            int x = u;
            while(pai[u] != u)
            {
                u = pai[u];
                rs = min(rs, best[u] + dist[{u,x}]);
            }
            cout << rs << '\n';
        }else
        {
            best[u] = 0;
            int x = u;
            while(pai[u] != u)
            {
                u = pai[u];
                best[u] = min(best[u], dist[{u, x}]);
            }
        }
    }
    return 0;
}
```

8.7 Tree Isomorphism

In this code, we generate an id for each tree. It is also possible to do with subtrees.

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

map<vector<int>, int> hasher;
int hashify(vector<int> x) {
    sort(all(x));
    if(!hasher[x]) {
        hasher[x] = hasher.size();
    }
    return hasher[x];
}
class Tree
{
    public:
    vector<vector<int>> adj;
    vector<int> sub, pai;
    int r[2], hs[2], n, bt = 0;
    Tree(int x): n(x)
    {
        adj.resize(n+1);
        sub.resize(n+1, 0);
        pai.resize(n+1);
    }
    void add_edge(int u, int v)
    {
        adj[u].pb(v);
        adj[v].pb(u);
    }
    int dfs(int u, int p = -1)
    {
        vector<int> at;
        for(int v:adj[u])
        {
```

```
            if(v == p)
                continue;
            at.pb(dfs(v, u));
        }
        return hashify(at);
    }
    void dfs_sub(int u, int p = -1)
    {
        pai[u] = p;
        sub[u] = 1;
        for(int v:adj[u])
        {
            if(v == p)
                continue;
            dfs_sub(v, u);
            sub[u]+=sub[v];
        }
    }
    int find_centroid(int u, int p = -1)
    {
        // cout << u << '\n';
        for(int v:adj[u])
        {
            if(v == p)
                continue;
            if(2*sub[v] > n)
                return find_centroid(v, u);
        }
        return u;
    }
    void build()
    {
        bt = 1;
        dfs_sub(1);
        r[1] = r[0] = find_centroid(1);
        for(int v:adj[r[0]])
        {
            if(v == pai[r[0]])
            {
                if(2*(n-sub[r[0]]) >= n)
                    r[1] = v;
            }else
            {
                if(2*sub[v] >= n)
                    r[1] = v;
            }
        }
        hs[0] = dfs(r[0]);
        hs[1] = dfs(r[1]);
    }


    bool isIsomorphic(Tree t2)
    {
        if(!bt)
            build();
        if(!t2.bt)
            t2.build();
        rep(i, 0, 2)
        {
            rep(j, 0, 2)
            {
                if(hs[i] == t2.hs[j]){
                    // cout << r[i] << ' ' << t2.r[j] << "<<<\n";
                    return true;
                }
```

```
            }
        }
        return false;
    }
};







void test()
{
    int n;
    cin >> n;
    Tree t1(n), t2(n);
    rep(i, 0, n-1)
    {
        int u, v;
        cin >> u >> v;
        t1.add_edge(u, v);
    }
    rep(i, 0, n-1)
    {
        int u, v;
        cin >> u >> v;
        t2.add_edge(u, v);
    }

    if(t1.isIsomorphic(t2))
        cout << "YES\n";
    else
        cout << "NO\n";
    // hasher.clear();
}


int32_t main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int t = 1;
    cin >> t;
    while(t--)
        test();
    return 0;
}
```

8.8 Virtual Tree

A different way to approach trees problems. Main idea here is to create a tree with the ”relevant nodes”. Normally related to colors and problems which we solve looking for each color at a time.

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

using namespace std;

//template

const int N = 400100;
const int K = 20;
vector<int> adj[N];
int pai[N][K], in[N], out[N], cs = 0;

```

```
void pre_dfs(int u, int p)


// u is ancestor of v ?
bool is_ancestor(int u, int v)

int calc_lca(int a, int b)


int cor[N];
vector<int> c[N];
vector<int> nadj[N];

int lc[N];

ll dp[N][3];

ll calc(int u,int col, int p)
{
    dp[u][1] = (cor[u] == col);
    dp[u][2] = 0;
    for(int v:nadj[u])
    {
        if(v == p)
            continue;
        calc(v,col,u);
        dp[u][2] += dp[u][1]*dp[v][1];
        if(cor[u] != col){
            dp[u][1] += dp[v][1];
        }
        dp[u][2] += dp[v][2];
    }
    // cout << u << ' ' << dp[u][2] << '\n';
    return dp[u][2];
}


ll solve(vector<int> &at, int j)
{
    // solve for tree with one vertex
    if(at.size() <= 1)
        return 0;
    sort(all(at),[&](int a, int b){
            return in[a] < in[b];
        });
    int m = at.size();
    rep(i, 0, m)
        lc[i] = at[i];
    rep(i, 0, m-1){
        lc[i+m] = calc_lca(at[i],at[i+1]);
    }
    sort(lc,lc+m+m-1,[&](int a, int b){
            return in[a] < in[b];
        });
    m = unique(lc,lc+m+m-1)-lc;
    rep(i, 0, m)
        nadj[lc[i]].clear();

    vector<int> st;
    st.pb(lc[0]);
    rep(i, 1, m)
    {
        while(!is_ancestor(st.back(),lc[i]))
            st.pop_back();
        nadj[st.back()].pb(lc[i]);
        nadj[lc[i]].pb(st.back());
```

```
        st.pb(lc[i]);
    }
    // solve for virtual tree
    return calc(lc[0],j,lc[0]);
}

void test()
{
    int n;
    cin >> n;

    rep(i, 1, n+1){
        adj[i].clear();
        c[i].clear();
    }
    cs = 0;
    rep(i, 1, n+1)
    {
        cin >> cor[i];
        c[cor[i]].pb(i);
    }
    rep(i, 0, n-1)
    {
        int u, v;
        cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }
    pre_dfs(1,1);
    ll rs = 0;
    rep(i, 1, n+1)
    {
        rs += solve(c[i],i);
    }
    cout << rs << '\n';
}

int32_t main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t = 1;
    cin >> t;
    while(t--){
        test();
    }
    return 0;
}
```

8.9 Link Cut Tree

A link cut tree is a data structure that uses splay trees to represent a forest of rooted trees and can perform the following operations with an amortized upper bound time complexity of $\mathcal{O}(\log N)$:

- Linking a tree with a node by making the root of the tree a child of any node of another tree
- Deleting the edge between a node and its parent, detaching the node's subtree to make a new tree
- Find the root of the tree that a node belongs to

These operations all use the `access`(v) subroutine, which creates a preferred path from the root of the represented tree to vertex v, making a corresponding auxiliary splay tree with v as the root.

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

struct Node {
```

```
  int x;
  Node *l = 0;
  Node *r = 0;
  Node *p = 0;
  bool rev = false;

  Node() = default;

  Node(int v) { x = v; }

  void push() {
    if (rev) {
      rev = false;
      swap(l, r);
      if (l) l->rev ^= true;
      if (r) r->rev ^= true;
    }
  }

  bool is_root() { return p == 0 || (p->l != this && this != p->r); }
};

struct LCT {
  vector<Node> a;

  LCT(int n) {
    a.resize(n + 1);
    for (int i = 1; i <= n; ++i) a[i].x = i;
  }

  void rot(Node *c) {
    auto p = c->p;
    auto g = p->p;

    if (!p->is_root()) (g->r == p ? g->r : g->l) = c;

    p->push();
    c->push();

    if (p->l == c) {  // rtr
      p->l = c->r;
      c->r = p;
      if (p->l) p->l->p = p;
    } else {  // rtl
      p->r = c->l;
      c->l = p;
      if (p->r) p->r->p = p;
    }

    p->p = c;
    c->p = g;
  }

  void splay(Node *c) {
    while (!c->is_root()) {
      auto p = c->p;
      auto g = p->p;
      if (!p->is_root()) rot((g->r == p) == (p->r == c) ? p : c);
      rot(c);
    }
    c->push();
  }

  Node *access(int v) {
    Node *last = 0;
    Node *c = &a[v];
```

```
    for (Node *p = c; p; p = p->p) {
      splay(p);
      p->r = last;
      last = p;
    }
    splay(c);
    return last;
  }

  void make_root(int v) {
    access(v);
    auto *c = &a[v];
    if (c->l) c->l->rev ^= true, c->l = 0;
  }

  void link(int u, int v) {
    make_root(v);
    Node *c = &a[v];
    c->p = &a[u];
  }

  void cut(int u, int v) {
    make_root(u);
    access(v);
    if (a[v].l) {
      a[v].l->p = 0;
      a[v].l = 0;
    }
  }

  bool connected(int u, int v) {
    access(u);
    access(v);
    return a[u].p;
  }
};

int main() {
  int n;
  int m;
  cin >> n >> m;
  LCT lc(n);

  for (int i = 0; i < m; i++) {
    string a;
    cin >> a;
    int b, c;
    cin >> b >> c;
    if (a == "add") { lc.link(b, c); }
    if (a == "rem") { lc.cut(b, c); }
    if (a == "conn") { cout << (lc.connected(b, c) ? "YES" : "NO") << "\n"; }
  }
}
```

Capítulo IX

Geometry

9.1 Simple Geometry

This section will have alot of functions used in geometry problems. They are related to points, lines, circles and polygons.

9.1.1 Points and Lines

This is the base of all codes in geometry. u can change the type from ldb to ll and also change the EPS. The rest is pretty much fixed.

```
const ld EPS = 1e-9;
const ld PI = acosl(-1);

using T = ll;
// using T = ld;
typedef pair<T,T> pt;
typedef pair<pt,pt> line;
typedef vector<pt> vpt;
typedef pair<pt, T> circle;

pt operator-(pt a) { return mp(-a.ft,-a.sd); }
pt &operator+=(pt &a, pt b) {
  a.ft += b.ft;
    a.sd += b.sd;
  return a;
}
pt &operator-=(pt &a, pt b) {
  a.ft -= b.ft;
    a.sd -= b.sd;
  return a;
}
pt &operator*=(pt &a, T r) {
  a.ft *= r;
    a.sd *= r;
  return a;
}
pt &operator/=(pt &a, T r) {
    a.ft /= r;
    a.sd /= r;
  return a;
}

pt operator+(pt a, pt b) { return a += b; }
pt operator-(pt a, pt b) { return a -= b; }
pt operator*(pt a, T r) {return a*=r;}
pt operator*(T r, pt a) {return a*=r;}
pt operator/(pt a, T r) {return a/=r;}
int sgn(T a) { return (a>EPS)-(a<-EPS); }
T sq(T a) { return a*a; }
T abs2(pt p) { return sq(p.ft)+sq(p.sd); }
ld abs(pt p) { return sqrtl(abs2(p)); }
pt unit(pt p) { return p/abs(p); }
ld arg(pt p) { return atan2(p.sd,p.ft); }
```

```
pt conj(pt p) { return mp(p.ft,-p.sd); }
pt perp(pt p) { return mp(-p.sd,p.ft); }
pt dir(T ang) { return mp(cos(ang),sin(ang)); }


pt operator*(pt a, pt b) {return pt(a.ft*b.ft-a.sd*b.sd,a.sd*b.ft+a.ft*b
    .sd);}

pt operator/(pt a, pt b) {return a*conj(b)/abs2(b);}

pt reflect(const pt& p, const line& l) {
  pt a = l.ft, d = l.sd-l.ft;
  return a+conj((p-a)/d)*d; }
pt foot(const pt& p, const line& l) {
  return (p+reflect(p,l))/(T)2; }

T dot(pt p1, pt p2)
{
    return p1.ft*p2.ft+p1.sd*p2.sd;
}

T dot(pt p1, pt p2, pt p3)
{
    return dot(p2-p1, p3-p1);
}

T cross(pt p1, pt p2)
{
    return p1.ft*p2.sd-p1.sd*p2.ft;
}

T cross(pt p1, pt p2, pt p3)
{
    return cross(p2-p1, p3-p1);
}

// rotate a 90 degrees centered in c
pt rotate90(pt a, pt c = mp(0,0))
{
    a-=c;
    swap(a.ft,a.sd);
    a.ft = -a.ft;
    a+=c;
    return a;
}

// rotate a d degrees centered in c
pt rotate(pt a, long double d, pt c = mp(0,0))
{
    a-=c;
    pt b;
    b.ft = a.ft*cos(d) - a.sd*sin(d);
    b.sd = a.sd*cos(d) + a.ft*sin(d);
    b+=c;
    return b;
}

bool onseg(pt p, line l) {
  return sgn(cross(l.ft,l.sd,p)) == 0 && sgn(dot(p,l.ft,l.sd)) <= 0;}

// distance betweem a line and a point, needs two points which are in
    the line
ld linedist(line lin, pt at)
{
    ld rs = abs(cross(at, lin.ft, lin.sd));
    rs/=abs(lin.ft-lin.sd);
```

```
    return rs;
}

// distance betweem a segment and a point
ld point_seg(line seg, pt at)
{
    if(dot(seg.ft, at, seg.sd) <= 0)
        return abs(at-seg.ft);
    if(dot(seg.sd, at, seg.ft) <= 0)
        return abs(at-seg.sd);
    return linedist(seg, at);
}

//  Usage: vpt v; sort(all(v),angleCmp);
// used to sort points int ccw around the origin
int half(pt x) { return x.sd != 0 ? sgn(x.sd) : -sgn(x.ft); }
bool angleCmp(pt a, pt b) { int A = half(a), B = half(b);
  return A == B ? cross(a,b) > 0 : A < B; }

// equivalent to: sort(all(v),[](pt a, pt b) {
//    return atan2(a.sd,a.ft) < atan2(b.sd,b.ft); });

// {unique intersection point} if it exists
// {b.f,b.s} if input lines are the same
// empty if lines do not intersect
vpt lineIsect(const line& a, const line& b) {
  T a0 = cross(a.ft,a.sd,b.ft), a1 = cross(a.ft,a.sd,b.sd);
  if (a0 == a1) return a0 == 0 ? vpt{b.ft,b.sd} : vpt{};
  return {(b.sd*a0-b.ft*a1)/(a0-a1)};
}

// point in interior of both segments a and b, if it exists
vpt strictIsect(const line& a, const line& b) {
  T a0 = cross(a.ft,a.sd,b.ft), a1 = cross(a.ft,a.sd,b.sd);
  T b0 = cross(b.ft,b.sd,a.ft), b1 = cross(b.ft,b.sd,a.sd);
  if (sgn(a0)*sgn(a1) < 0 && sgn(b0)*sgn(b1) < 0)
    return {(b.sd*a0-b.ft*a1)/(a0-a1)};
  return {};
}

// intersection of segments, a and b may be degenerate
vpt segIsect(const line& a, const line& b) {
  vpt v = strictIsect(a,b); if (!v.empty()) return v;
  set<pt> s;
  #define i(x,y) if (onseg(x,y)) s.insert(x)
  i(a.ft,b); i(a.sd,b); i(b.ft,a); i(b.sd,a);
  return {all(s)};
}
```

9.1.2 Circles

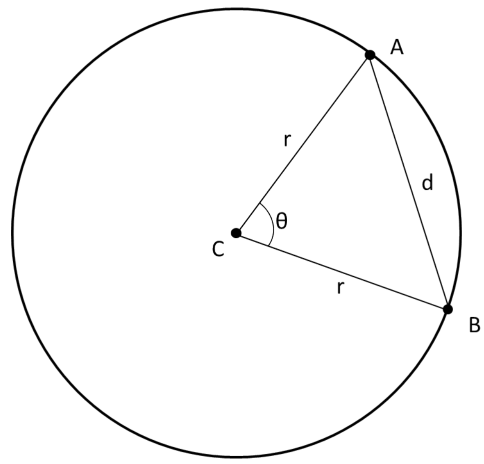
This section will have alot of algorithms related to circles and their description A circle will have the following format:

```
    typedef pair<pt, T> circle;

```

To see if a point is in/on/out a circle we can do:

```
// -1 if inside, 0 if in border, 1 if outside
int in(const circle& x, const pt& y) {
    return sgn(abs(y-x.ft)-x.sd);
 }

```

Arc length of two points:



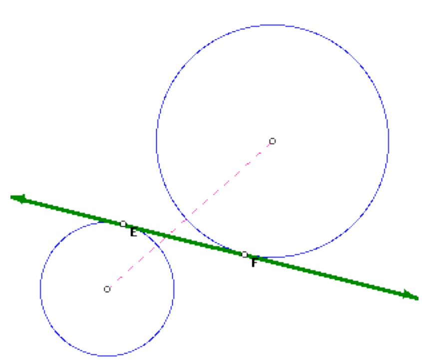

```
T arcLength(const circle& x, pt a, pt b) {
    // precondition: a and b on x
    pt d = (a-x.ft)/(b-x.ft); return x.sd*acos(d.ft);
}
```

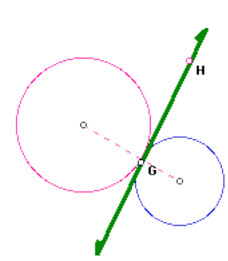
Intersections related to circles:

```
vpt isect(const circle& x, const circle& y) { // precondition: x!=y
    T d = abs(x.ft-y.ft), a = x.sd, b = y.sd;
    if (sgn(d) == 0) { assert(a != b); return {}; }
    T C = (a*a+d*d-b*b)/(2*a*d);
    if (abs(C) > 1+EPS) return {};
    T S = sqrt(max(1-C*C,(T)0));
    pt tmp = (y.ft-x.ft)/d*x.sd;
    return {x.ft+tmp*pt(C,S),x.ft+tmp*pt(C,-S)};
}

vpt isect(const circle& x, const line& y) {
    pt c = foot(x.ft,y);
    T sq_dist = sq(x.sd)-abs2(x.ft-c);
    if (sgn(sq_dist) < 0) return {};
    pt offset = unit(y.sd-y.ft)*sqrt(max(sq_dist,T(0)));
    return {c+offset,c-offset};
}

T isect_area(circle x, circle y) { // not thoroughly tested
    T d = abs(x.ft-y.ft), a = x.sd, b = y.sd;
    if (a < b) swap(a,b);
    if (d >= a+b) return 0;
    if (d <= a-b) return PI*b*b;
    T ca = (a*a+d*d-b*b)/(2*a*d), cb = (b*b+d*d-a*a)/(2*b*d);
    T s = (a+b+d)/2, h = 2*sqrt(s*(s-a)*(s-b)*(s-d))/d;
    return a*a*acos(ca)+b*b*acos(cb)-d*h;
}
```

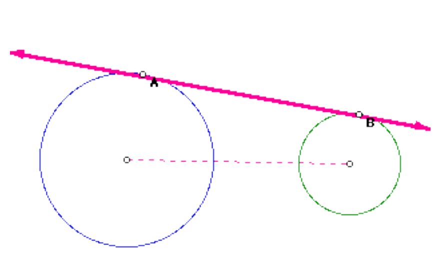
A tangent to a circle is a line in the plane of a circle which intersects the circle in exactly one point. This point is called the point of tangency.

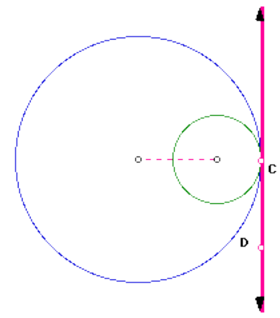
A tangent of two circles is a common internal tangent if the intersection of the tangent and the line segment joining the centers is not empty.





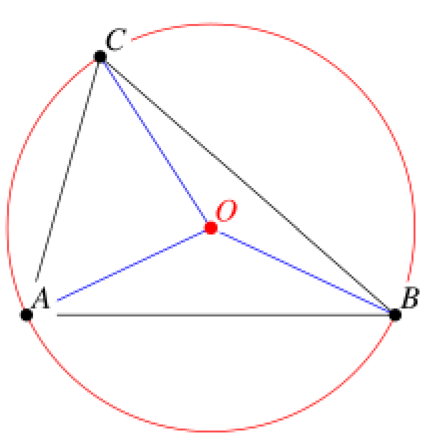
A tangent of two circles is a common external tangent if the intersection of the tangent and the line segment joining the centers is empty.

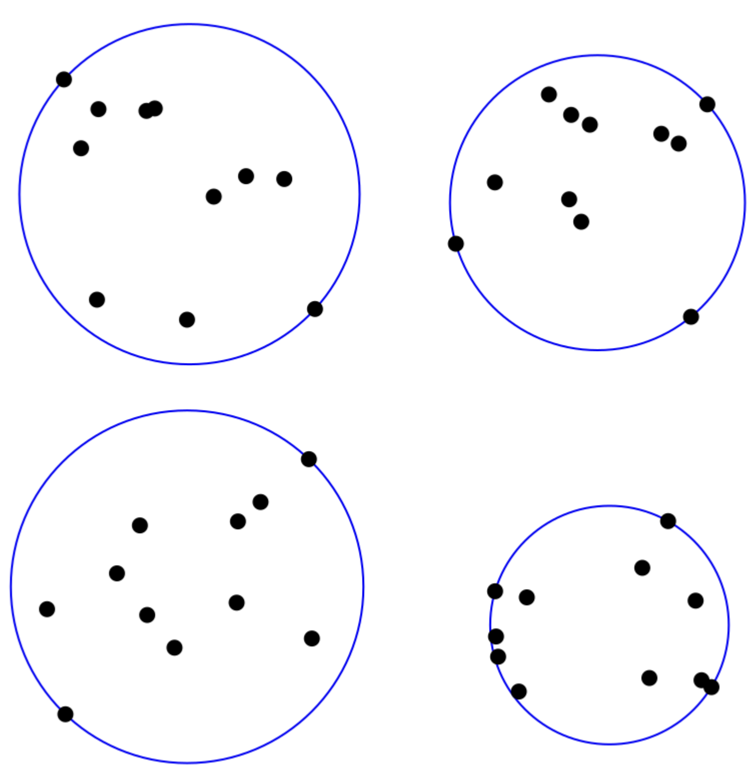





```
pt tangent(pt x, circle y, int t = 0) {
  y.sd = abs(y.sd); // abs needed because internal calls y.s < 0
  if (y.sd == 0) return y.ft;
  T d = abs(x-y.ft);
  pt a = pow(y.sd/d,2)*(x-y.ft)+y.ft;
  pt b = sqrt(d*d-y.sd*y.sd)/d*y.sd*unit(x-y.ft)*dir(PI/2);
  return t == 0 ? a+b : a-b;
}
vector<pair<pt,pt>> external(circle x, circle y) {
  vector<pair<pt,pt>> v;
  if (x.sd == y.sd) {
    pt tmp = unit(x.ft-y.ft)*x.sd*dir(PI/2);
    v.eb(x.ft+tmp,y.ft+tmp);
    v.eb(x.ft-tmp,y.ft-tmp);
  } else {
    pt p = (y.sd*x.ft-x.sd*y.ft)/(y.sd-x.sd);
    rep(i, 0, 2) v.eb(tangent(p,x,i),tangent(p,y,i));
  }
  return v;
}
vector<pair<pt,pt>> internal(circle x, circle y) {
  return external({x.ft,-x.sd},y); }

```

The CircumCenter of a triangle is the minimum enclosing circle of a triangle. We can also calculated the minimum enclosing circle of a polygon.





```
// return the minimum enclosing circle of a triangle
circle ccCenter(pt a, pt b, pt c) {
  b -= a; c -= a;
  pt res = b*c*(conj(c)-conj(b))/(b*conj(c)-conj(b)*c);
```

```
  return {a+res,abs(res)};
}

// return the minimum enclosing circle of a set of points O(N)
circle mec(vpt ps) {
  shuffle(all(ps), rng);
  pt o = ps[0];
    T r = 0, EPS = 1+1e-8;
  rep(i, 0, (int)ps.size())
        if (abs(o-ps[i]) > r*EPS) {
            o = ps[i], r = 0; // point is on MEC
            rep(j, 0, i)
                if (abs(o-ps[j]) > r*EPS) {
                    o = (ps[i]+ps[j])/2, r = abs(o-ps[i]);
                    rep(k, 0, j) if (abs(o-ps[k]) > r*EPS)
                        tie(o,r) = ccCenter(ps[i],ps[j],ps[k]);
                }
        }
  return {o,r};
}

```

9.2 Convex Hull

9.2.1 Rotation Calipers

Diameter from a convex Polygon.

Solution $O(n)$. Original Problem (NCPC 2013 - D).

```
Implementacao do Convex Hull do CP-ALGORITHMS
Para usar crie um vetor com os pontos,
  vector<pt> pts;
e execute
  convex_hull(|pts|);
Apos isso, o vector pts sera alterado e ele so tera os pontos do Convex Hull
na ordem horaria, comecando do elemento de menor x, como segunda condicao menor y

Como funciona o algoritmo
- Acha os dois extremos de x.
- Monta dois subgrupos, os "up" e os "down", em relacao
Convex Hull na parte superior e inferior.
- Itera pelos pontos e usa o produto vetorial para ver se
ele forma um sentido horario, se formar adiciona.
- Remove todos os anteriores ao ultimo ponto adicionado que agora, com o ponto
atual, formando um sentido anti-horario
```

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

typedef pair<int, int> Point;

int cross(Point a, Point b, Point c) {
  return a.first * (b.second - c.second)
      + b.first * (c.second - a.second)
      + c.first * (a.second - b.second);
}

const int N = 100100;
pair<int, int> p[N], p2[N], upper[N], lower[N];
int uppersize = 0, lowersize = 0;

void convexHull(int n) {
```

```cpp
  sort(p, p+n);
  uppersize = 0;
  lowersize = 0;

  for (int i = 0; i < n; i++) {
    while (uppersize >= 2 && cross(upper[uppersize-2], upper[uppersize-1], p[i]) >= 0) {
      uppersize--;
    }

    upper[uppersize++] = p[i];
  }

  for (int i = 0; i < n; i++) {
    while (lowersize >= 2 && cross(lower[lowersize-2], lower[lowersize-1], p[i]) <= 0) {
      lowersize--;
    }

    lower[lowersize++] = p[i];
  }
}

double dist(Point a, Point b) {
  return hypot(a.first - b.first, a.second - b.second);
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  int c;

  cin >> c;

  for (int i = 0; i < c; i++) {
    cin >> p2[i].first >> p2[i].second;
  }

  sort(p2, p2+c);

  int n = 0;

  p[n++] = p2[0];

  for (int i = 1; i < c; i++) {
    if (p2[i] != p2[i-1]) {
      // cout << "adding " << p2[i].first << " " << p2[i].second << endl;
      p[n++] = p2[i];
    }
  }

  // cout << "n: " << n << endl;
  convexHull(n);

  // cout << "lower: \n";
  // for (int i = 0; i < lowersize; i++) {
  //  cout << lower[i].first << ", " << lower[i].second << endl;
  // }

  // cout << "upper: \n";
  // for (int i = 0; i < uppersize; i++) {
  //  cout << upper[i].first << ", " << upper[i].second << endl;
  // }

```

```
  int i = 0, j = lowersize-1;
  double ans = 0;


  while (i < uppersize-1 || j > 0) {
    // cout << "(" << i << ") " << upper[i].first << ", " << upper[i].
 second << " dist to "
    //  << "(" << j << ") " << lower[j].first << ", " << lower[j].second
    //  << " = " << dist(upper[i], lower[j]) << endl;

    ans = max(ans, dist(upper[i], lower[j]));

    if (i == uppersize-1) {
      j--;
    } else if (j == 0) {
      i++;
    } else {
      if ((upper[i+1].second - upper[i].second) * (lower[j].first -
 lower[j-1].first)
        > (lower[j].second - lower[j-1].second) * (upper[i+1].first -
 upper[i].first)) {
        i++;
      } else {
        j--;
      }
    }
  }

  cout << setprecision(10) << fixed;
  cout << ans << "\n";

  return 0;
}
```

9.3 Line Sweep

9.3.1 Points Inside Triangles

Given n triangles with vertices (x_i, y_i), $(x_i + d, y_i)$, $(x_i, y_i + d)$, and q points, compute for each triangle, the number of points that lie inside or in boundary of him. $O(nlogn)$. (Original Problem: CODECHEF - TRIANGULAR QUERIES)

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

#define getcx getchar_unlocked
#define pc(x) putchar_unlocked(x)

using namespace std;

typedef pair<int, int> pii;
typedef long double ld;
typedef long long ll;
typedef unsigned long long ull;

inline int readInt()
{
    bool minus = false;
    register ll result = 0;
    register char ch = getchar_unlocked();
    while (true)
    {
        if (ch == '-') break;
        if (ch >= '0' && ch <= '9') break ;
        ch = getchar_unlocked();
    }
    if (ch == '-') minus = true;
```

```
    else result = ch-'0';
    while (true)
    {
        ch = getchar_unlocked();
        if (ch < '0' || ch > '9') break;
        result = result*10LL + (ch - '0');
    }
    if (minus) return -result;
    else return result;
}

inline void writeInt ( int n )
{
    register ll N = n, rev, count = 0 ;
    rev = N ;
    if (N == 0)
    {
        pc('0');  return ;
    }
    while ((rev % 10LL) == 0LL)
    {
        count++; rev /= 10LL;
    } //obtain the count of the number of 0s
    rev = 0;
    while (N != 0LL)
    {
        rev = (rev<<3LL) + (rev<<1LL) + N % 10LL; N /= 10LL;
    }  //store reverse of N in rev
    while (rev != 0LL)
    {
        pc(rev % 10LL + '0'); rev /= 10LL ;
    }
    while (count--) pc('0') ;
}

inline void write_string(char *str)
{
    register char c = 0;
    register int i = 0;
    while (c < 33)
        c = getchar_unlocked() ;
    while (c != '\n')
    {
        str[i] = c;
        c = getchar_unlocked() ;
        i = i + 1;
    }
    str[i] = '\0';
}

class Triangle{
    public:
    int x, y, d;
    int idx;

    bool operator<(Triangle &other){
        return (x + y) < (other.x + other.y);
    }
};

enum events_types{points_ap, beg_triangle, end_triangle};

class Event{
    public:
    int x, y;
    int type;//0 -> POINT EVENT
```

```
            //1 -> BEGIN TRIANGLE
            //2 -> END OF A TRIANGLE
    int idx;

    bool operator<(Event &other){
        int w1 = x + y;
        int w2 = other.x + other.y;

        if(w1 < w2){
            return true;
        }else if(w1 > w2){
            return false;
        }else{
            return type < other.type;
        }
    }
};

class Point{
    public:
    int x, y;
};

const int N = int(3e5 + 10);
Point points[N];
Triangle triangles[N];
int bag[N];
int BIT_X[N], BIT_Y[N];
vector< Event > events;

int LSOne(int x){
    return x & (-x);
}

void update(int BIT[], int x, int value){
    while(x < N){
        BIT[x] += value;
        x += LSOne(x);
    }
}

int query(int BIT[], int x){
    int sum = 0;

    while(x > 0){
        sum += BIT[x];
        x -= LSOne(x);
    }

    return sum;
}

int rsq(int BIT[], int l, int r){
    r = min(r, N - 1);

    return query(BIT, r) - query(BIT, l - 1);
}

void run_sweep(int n, int q){
    for(int i = 0 ; i < n ; i++){
        events.push_back({points[i].x, points[i].y, points_ap, i});
    }

    for(int i = 0 ; i < q ; i++){
        events.push_back({triangles[i].x - 1, triangles[i].y - 1,
   beg_triangle, i});
```

```
        events.push_back({triangles[i].x + triangles[i].d, triangles[i].y, end_triangle, i});
    }

    sort(events.begin(), events.end());

    int size_active_set = 0;

    for(int i = 0 ; i < events.size() ; i++){
        if(events[i].type == points_ap || events[i].type == beg_triangle){
            if(events[i].x == 0){
                continue;
            }

            if(events[i].type == points_ap){
                size_active_set++;
                update(BIT_X, events[i].x, 1);
                update(BIT_Y, events[i].y, 1);
                // printf("adding %d\n", events[i].idx);
            }else{
                bag[events[i].idx] = rsq(BIT_X, 1, events[i].x) + rsq(BIT_Y, 1, events[i].y) - size_active_set;
                // printf("1: %d->%d\n", events[i].idx, bag[events[i].idx]);
            }
        }else{
            events[i].x -= triangles[events[i].idx].d;
            bag[events[i].idx] = size_active_set - (rsq(BIT_X, 1, events[i].x - 1) + rsq(BIT_Y, 1, events[i].y - 1) - bag[events[i].idx]);
            // printf("2: %d->%d\n", events[i].idx, bag[events[i].idx]);
        }
    }
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n, q;

    n = readInt();
    q = readInt();
    // scanf("%d %d", &n, &q);v

    for(int i = 0 ; i < n ; i++){
        points[i].x = readInt();
        points[i].y = readInt();
        // scanf("%d %d", &points[i].x, &points[i].y);
    }

    for(int i = 0 ; i < q ; i++){
        triangles[i].x = readInt();
        triangles[i].y = readInt();
        triangles[i].d = readInt();
        // scanf("%d %d %d", &triangles[i].x, &triangles[i].y, &triangles[i].d);
    }

    run_sweep(n, q);

    for(int i = 0 ; i < q ; i++){
        writeInt(bag[i]);
        pc('\n');
        // printf("%d\n", bag[i]);
    }
```

```

    return 0;
}
```

9.3.2 Ranking Problem

Given n students that attended to 3 contests, and all of them have different ranks, between 1 and n inclusive, for each contest. We say that one student a is better that student b, if all ranks of student a in each contest is lesser that student b. A student is said to be excellent if no other student is better than him. How many excelent students there exists ?

This solutions runs in $O(nlogn)$. (Original Problem: SPOJ: NICEDAY)

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

using namespace std;
typedef pair<int, int> pii;
typedef long double ld;
typedef long long ll;
typedef unsigned long long ull;

class Rank{
    public:
    int a, b, c;

    bool operator <(Rank &other){
        return this->a < other.a;
    }
};

const int N = int(1e5 + 10);
Rank ranks[N];
int BIT[N];

int LSOne(int x){
    return x & (-x);
}

int query(int x){
    int best = INT32_MAX;

    while(x != 0){
        if(BIT[x] != -1)
            best = min(best, BIT[x]);
        x -= LSOne(x);
    }

    return best;
}

void update(int x, int val){
    while(x < N){
        if(BIT[x] == -1){
            BIT[x] = val;
        }

        BIT[x] = min(BIT[x], val);
        x += LSOne(x);
    }
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int t;

    cin >> t;

    while(t--){
        memset(BIT, -1, sizeof BIT);

```

```
        int n;

        cin >> n;

        for(int i = 0 ; i < n ; i++){
            cin >> ranks[i].a >> ranks[i].b >> ranks[i].c;
        }

        sort(ranks, ranks + n);

        int cnt = n;

        for(int i = 0 ; i < n ; i++){
            int q = query(ranks[i].b - 1);
            // cout << "q: " << q << endl;

            if(q < ranks[i].c){
                cnt--;
            }

            update(ranks[i].b, ranks[i].c);
        }

        cout << cnt << endl;
    }

    return 0;
}
```

9.3.3 Balls Falls and Segments

Given n segments (non horizontal and with no common points), and m balls in a 2D plane, say for each ball the x point which he falls. (This solution was upload, the source code is just for only one ball, but the overall complexity is still $O(nlogn)$ if the number of balls is $O(n)$).

This solutions runs in $O(nlogn)$. Original Problem (NCPC 2013 - H).

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

using namespace std;

typedef pair< int, int >  pii;
typedef double  ld;
typedef long long ll;
typedef unsigned long long  ull;
typedef pair< ll, ll >  pll;

const int N = int(1e5 + 10);
pll rectangle[N][2];

struct event{
public:
  int type;
  int idx;

  event(){

  }

  event(int tpe, int id){
    this->type = tpe;
    this->idx = id;
  }
};

ll cross(pll a, pll b, pll c){
    ll dx1 = a.first - b.first, dy1 = a.second - b.second;
    ll dx2 = c.first - b.first, dy2 = c.second - b.second;

    return dx1 * dy2 - dx2 * dy1;
}

struct MNode{
public:
  int idx;

  MNode(){

  }

  MNode(int id){
    this->idx = id;
  }

  bool operator <(MNode other) const{
        if(rectangle[idx][0].first >= rectangle[other.idx][0].first){
            ll a =  cross(rectangle[other.idx][0], rectangle[idx][0], rectangle[other.idx][1]);

            return cross(rectangle[other.idx][0], rectangle[idx][0], rectangle[other.idx][1]) > 0;
        }else{
            ll a = cross(rectangle[idx][0], rectangle[other.idx][0], rectangle[idx][1]);

            return cross(rectangle[idx][0], rectangle[other.idx][0], rectangle[idx][1]) < 0;
```

```
        }
    }
};

vector< event > events;
pll balls[N];
int adj[N];
int adj_balls[N];
set< MNode > st;
ll resp[N];
int prio[] = {0, 2, 1};

bool compar(event &a, event &b){
  ll xa, xb;
  ll ya, yb;

  if(a.type <= 1){
    xa = rectangle[a.idx][a.type].first;
    ya = rectangle[a.idx][a.type].second;
  }else{
    xa = balls[a.idx].first;
    ya = balls[a.idx].second;
  }

  if(b.type <= 1){
    xb = rectangle[b.idx][b.type].first;
    yb = rectangle[b.idx][b.type].second;
  }else{
    xb = balls[b.idx].first;
    yb = balls[b.idx].second;
  }

  if(xa < xb){
    return true;
  }else if(xa > xb){
    return false;
  }else{
    if(prio[a.type] == prio[b.type]){
            if(a.type == 1){
                return ya > yb;
            }

      return ya < yb;
    }

    return prio[a.type] < prio[b.type];
  }
}

ll m_point(int u){
  ll mx;

  if(rectangle[u][0].second > rectangle[u][1].second){
    mx = rectangle[u][1].first;
  }else{
    mx = rectangle[u][0].first;
  }

  return mx;
}

bool intersect(int u, int v){
  ll mx = m_point(u);

  return rectangle[v][0].first <= mx && mx <= rectangle[v][1].first;
}
```

```

void line_sweep(){
  sort(events.begin(), events.end(), compar);

  for(int i = 0 ; i < events.size() ; i++){
    event u = events[i];

    if(u.type == 2){
      rectangle[N - 1][0] = rectangle[N - 1][1] = balls[u.idx];

      if(st.size() == 0) continue;

      adj_balls[u.idx] = st.begin()->idx;
    }else{
      if(u.type == 1){
        auto d = st.find(MNode(u.idx));

                if(rectangle[u.idx][0].second > rectangle[u.idx][1].
 second){
                    auto nxt = next(d);

                    if(nxt != st.end()){
                        adj[d->idx] = nxt->idx;
                    }
                }
        st.erase(d);
      }else{
        st.insert(MNode(u.idx));
        auto d = st.find(MNode(u.idx));

        if(rectangle[u.idx][0].second < rectangle[u.idx][1].second){
                    auto nxt = next(d);

                    if(nxt != st.end()){
                        adj[d->idx] = nxt->idx;
                    }
                }
      }
    }
  }

  st.clear();
  events.clear();
}

int get_resp(int u){
  if(resp[u] != -1){
    return resp[u];
  }

  if(adj[u] == -1){
    return m_point(u);
  }

  return resp[u] = get_resp(adj[u]);
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(NULL);

  memset(adj_balls, -1, sizeof adj_balls);
  memset(adj, -1, sizeof adj);
  memset(resp, -1, sizeof resp);

  int n;
```

```
  cin >> n;

  for(int i = 0 ; i < n ; i++){
    cin >> rectangle[i][0].first >> rectangle[i][0].second >> rectangle[
   i][1].first >> rectangle[i][1].second;

    if(rectangle[i][1].first < rectangle[i][0].first){
      swap(rectangle[i][1], rectangle[i][0]);
    }
  }

  int m = 1;// m = number of balls

  for(int i = 0 ; i < m ; i++){
    cin >> balls[i].first;

    balls[i].second = ll(1e9);
  }

  for(int i = 0 ; i < n ; i++){
    events.push_back(event(0, i));
    events.push_back(event(1, i));
  }

  for(int i = 0 ; i < m ; i++){
    events.push_back(event(2, i));
  }

  line_sweep();

  for(int i = 0 ; i < m ; i++){
    if(adj_balls[i] == -1){
      cout << balls[i].first << endl;
    }else{
      cout << get_resp(adj_balls[i]) << endl;
    }
  }

  return 0;
}
```

9.3.4 Checking Points Inside Convex Polygon

Online

Given one point and a Convex Polygon, with vertices in counter clockwise order, check if this point is inside or in boundary of him.

This solutions runs in $O(logn)$.

```
struct pt{
    ll x, y;

    pt(){}

    pt(ll _x, ll _y):x(_x), y(_y){}

    pt operator+(pt p){
        return pt(x + p.x, y + p.y);
    }

    pt operator-(pt p){
        return pt(x - p.x, y - p.y);
    }

    ll cross(pt p){
        return x * p.y - y * p.x;
    }

    ll dot(pt p){
        return x * p.x + y * p.y;
    }

    ll cross(pt a, pt b){
        return (a - *this).cross(b - *this);
    }

    ll dot(pt a, pt b){
        return (a - *this).dot(b - *this);
    }

    ll sqrLen(){
        return this->dot(*this);
    }
};

bool lexComp(pt l, pt r){
    return l.x < r.x || (l.x == r.x && l.y < r.y);
}

int sgn(ll val){
    return val > 0 ? 1 : (val == 0 ? 0 : -1);
}

bool pointsInTriangle(pt a, pt b, pt c, pt point){
    ll s1 = abs(a.cross(b, c));
    ll s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs(point.
   cross(c, a));

    return s1 == s2;
}

bool pointInConvexPolygon(pt point, vector< pt > points){
    int n = points.size();

    pt p1 = points[1];
    pt p0 = points[0];
    pt pn = points[n - 1];

```

```
    if((p1 - p0).cross(point - p0) != 0 && sgn((p1 - p0).cross(point -
   p0)) != sgn((p1 - p0).cross(pn - p0))){
        return false;
    }

    if((pn - p0).cross(point - p0) != 0 && sgn((pn - p0).cross(point -
   p0)) != sgn((pn - p0).cross(p1 - p0))){
        return false;
    }

    if((p1 - p0).cross(point - p0) == 0){
        return (p1 - p0).sqrLen() >= (point - p0).sqrLen();
    }

    int lo = 0;
    int hi = n - 2;
    int r = -1;

    while(lo <= hi){
        int mid = (lo + hi) >> 1;

        if((point - p0).cross(points[mid] - p0) <= 0){
            r = mid;
            lo = mid + 1;
        }else{
            hi = mid - 1;
        }
    }

    assert(r != -1);

    return pointsInTriangle((points[r] - p0), points[(r + 1) % n] - p0,
   pt(0, 0), point - p0);
}
```

9.3.5 Radial Sweep

You are given N red points and M blue points on a 2D plane.

You are required to delete the minimum number of points(the deleted points can be of both colors) so that it's possible to draw a line such that all remaining red points are on one side of the line while all remaining blue points are on the other side of the line.

This solutions runs in $O(n^2 logn)$. Original Problem (Codechef, REDBLUE - DEC17)

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

using namespace std;

const double eps = 1e-10;

struct Point {
  int x, y, color;
} points[2001];

int M, N;
double pi;

int ComputeMin(int a1, int b1, int a2, int b2) {
  return min(b1 + a2, a1 + b2);
}

int cnt[2];
vector<pair<double, int>> vp;

int Solve(int idx) {
  vp.clear();
  for (int i = 0; i < M + N; ++i) {
```

```
    if (i == idx) continue;
    const int dx = points[i].x - points[idx].x, dy = points[i].y - points[idx].y;
    assert(dx != 0 || dy != 0);
    const double u = atan2(dy, dx);
    vp.push_back({u, i});
    vp.push_back({u + 2.0 * pi, i});
  }

  sort(vp.begin(), vp.end());

  int ans = M + N, color_idx = points[idx].color;

  memset(cnt, 0, sizeof(cnt));
  for (int i = 0, j = 0; i < M + N - 1; ++i) {
    while (j < int(vp.size()) && vp[j].first - vp[i].first <= pi - eps) {
      ++cnt[points[vp[j].second].color];
      ++j;
    }

    // idx inside, vp[i].second inside.
    ++cnt[color_idx];
    ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));

    // idx inside, vp[i].second outside.
    --cnt[points[vp[i].second].color];
    ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));

    // idx outside, vp[i].second outside.
    --cnt[color_idx];
    ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));

    // idx outside, vp[i].second inside.
    ++cnt[points[vp[i].second].color];
    ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));

    // Move vp[i].second outside.
    --cnt[points[vp[i].second].color];
  }

  return ans;
}

int main() {
  pi = 2.0 * acos(0.0);
  int T;
  scanf("%d", &T);
  assert(1 <= T && T <= 100);
  while (T--) {
        scanf("%d %d", &M, &N);

      for (int i = 0; i < M; ++i) {
        scanf("%d %d", &points[i].x, &points[i].y);

        points[i].color = 0;
      }

      for (int i = M; i < M + N; ++i) {
        scanf("%d %d", &points[i].x, &points[i].y);

        points[i].color = 1;
      }

    int ans = M + N;
    for (int idx = 0; idx < M + N; ++idx) ans = min(ans, Solve(idx));
```

```
    printf("%d\n", ans);
  }
  return 0;
}
```

9.3.6 Radial Sweep sem Double

You are in a point inside a square NxN. There are some rocks(polygons) inside the square, u need to discover how many points form the perimeter of the square, are visible, form your oirigin

In this problem, there are three types of events: when our ray hits a fence point, enters a rock, or exits a rock.

The second and third types of events can be found for each rock by sorting the rays to its vertices by bearing and then taking the two endpoints of the sorted list. These two rays are the two tangents to the rock.

We can then perform a radial sweep to find the fence posts that Farmer Don can see - these fence posts are simply the ones where the number of type-2 and type-3 events we've processed so far are equal.

```
#include <bits/stdc++.h>
#define x first
#define y second
typedef long long ll;
using namespace std;

const double PI = 4 * atan(1);

struct Event {
  short type, id;
  pair<ll, ll> loc;
};

pair<ll, ll> origin, polygon[22];

// Cross product
ll cross(pair<ll, ll> a, pair<ll, ll> b) {
  return (a.y - origin.y) * (b.x - origin.x) -
         (a.x - origin.x) * (b.y - origin.y);
}

// Which half of the plane some point lies in
int half(pair<ll, ll> p) {
  if (p.x != origin.x) return (p.x < origin.x) - (p.x > origin.x);
  return (p.y < origin.y) - (p.y > origin.y);
}

// Custom comparator to sort by bearing
bool operator<(Event a, Event b) {
  int ah = half(a.loc), bh = half(b.loc);
  if (ah == bh) {
    ll c = cross(a.loc, b.loc);
    if (c == 0) return a.type > b.type;
    return c > 0;
  }
  return ah < bh;
}

// Generates the next fence post in clockwise order
Event get_next_post(Event curr, int n) {
  if (curr.loc.x == n) {
    if (curr.loc.y) return {0, 0, {n, curr.loc.y - 1}};
    return {0, 0, {n - 1, 0}};
  } else if (!curr.loc.x) {
    if (curr.loc.y != n) return {0, 0, {0, curr.loc.y + 1}};
    return {0, 0, {1, n}};
  } else if (curr.loc.y == n) {
    if (curr.loc.x != n) return {0, 0, {curr.loc.x + 1, n}};
```

```
    return {0, 0, {n, n - 1}};
  } else {
    if (curr.loc.x) return {0, 0, {curr.loc.x - 1, 0}};
    return {0, 0, {0, 1}};
  }
}

vector<Event> events;
bool before[44444];

int main() {
  cin.tie(0)->sync_with_stdio(0);
  int n, r;
  cin >> n >> r >> origin.x >> origin.y;

  for (int i = 0; i < r; i++) {
    int m;
    cin >> m;
    for (int j = 0; j < m; j++) cin >> polygon[j].x >> polygon[j].y;
    // Sort the polygon's vertices to find the 2 "tangents" from the
   origin
    sort(polygon, polygon + m,
         [](pair<ll, ll> a, pair<ll, ll> b) { return cross(a, b) > 0; })
   ;
    events.push_back({1, i, polygon[0]});
    events.push_back({-1, i, polygon[m - 1]});
  }
  sort(events.begin(), events.end());

  int active = 0;
  // Do an initial sweep to handle rocks containing the ray with bearing
    0
  // This way, `active` won't be messed up
  for (Event i : events) {
    if (i.type == 1) before[i.id] = true;
    if (i.type == -1 && !before[i.id]) active++;
  }

  int ans = 0, ptr = 0;
  Event curr_post = {0, 0, {origin.x, n}};
  for (Event i : events) {
    while (ptr != 4 * n && curr_post < i) {
      // If there are no rocks that our current ray intersects...
      if (!active) ans++;
      ptr++;
      curr_post = get_next_post(curr_post, n);
    }

    if (i.type == 1) active++;
    else active--;
  }
  if (!active) ans += 4 * n - ptr;

  cout << ans;
  return 0;
}
```

9.4 Minimum Perimeter Triangle

Given n points in a 2D plane, return the minimum perimeter that can be formed taking three points, collinear points are allowed. $O(nlogn)$. (Original Problem - Google Code Jam WF 2009 - B)

```
#include <algorithm>
#include <cassert>
#include <cmath>
```

```
#include <cstdio>
#include <cstdlib>
#include <vector>
using namespace std;
#define REP(i,n) for(int i=0;i<(n);++i)
template<class T> inline int size(const T&c) { return c.size();}

const int BILLION = 1000000000;
const double INF = 1e20;
typedef long long LL;

struct Point {
  int x,y;
  Point() {}
  Point(int x,int y):x(x),y(y) {}
};

inline Point middle(const Point &a, const Point &b) {
  return Point((a.x+b.x)/2, (a.y+b.y)/2);
}

struct CmpX {
  inline bool operator()(const Point &a, const Point &b) {
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
  }
} cmpx;

struct CmpY {
  inline bool operator()(const Point &a, const Point &b) {
    if(a.y != b.y) return a.y < b.y;
    return a.x < b.x;
  }
} cmpy;

inline LL sqr(int x) { return LL(x) * LL(x); }

inline double dist(const Point &a, const Point &b) {
  return sqrt(double(sqr(a.x-b.x) + sqr(a.y-b.y)));
}

inline double perimeter(const Point &a,
                        const Point &b,
                        const Point &c) {
  return dist(a,b) + dist(b,c) + dist(c,a);
}

double calc(int n, const Point points[],
            const vector<Point> &pointsByY) {
  if(n<3) return INF;
  int left = n/2;
  int right = n-left;
  Point split = middle(points[left-1], points[left]);
  vector<Point> pointsByYLeft, pointsByYRight;
  pointsByYLeft.reserve(left);
  pointsByYRight.reserve(right);
  REP(i,n) {
    if(cmpx(pointsByY[i], split))
      pointsByYLeft.push_back(pointsByY[i]);
    else
      pointsByYRight.push_back(pointsByY[i]);
  }
  double res = INF;
  res = min(res, calc(left, points, pointsByYLeft));
  res = min(res, calc(right, points+left, pointsByYRight));
  static vector<Point> closeToTheLine;
```

```
  int margin = (res > INF/2) ? 2*BILLION : int(res/2);
  closeToTheLine.clear();
  closeToTheLine.reserve(n);
  int start = 0;
  for(int i=0;i<n;++i) {
    Point p = pointsByY[i];
    if(abs(p.x - split.x) > margin) continue;
    while(start < size(closeToTheLine) &&
          p.y - closeToTheLine[start].y > margin) ++start;
    for(int i=start;i<size(closeToTheLine);++i) {
      for(int j=i+1;j<size(closeToTheLine);++j) {
        res = min(res, perimeter(p, closeToTheLine[i],
                                 closeToTheLine[j]));
      }
    }
    closeToTheLine.push_back(p);
  }
  return res;
}

double calc(vector<Point> &points) {
  sort(points.begin(), points.end(), cmpx);
  vector<Point> pointsByY = points;
  sort(pointsByY.begin(), pointsByY.end(), cmpy);
  return calc(size(points), &points[0], pointsByY);
}

int main() {
  assert(0==system("cat > Input.java"));
  fprintf(stderr, "Compiling generator\n");
  assert(0==system("javac Input.java"));
  fprintf(stderr, "Running generator\n");
  assert(0==system("java -Xmx512M Input > input.tmp"));
  fprintf(stderr, "Solving\n");
  FILE *f = fopen("input.tmp", "r");
  int ntc; fscanf(f, "%d", &ntc);
  REP(tc,ntc) {
    int n; fscanf(f, "%d", &n);
    vector<Point> points;
    points.reserve(n);
    REP(i,n) {
      int x,y; fscanf(f, "%d%d", &x, &y);
      points.push_back(Point(2*x-BILLION,2*y-BILLION));
    }
    double res = calc(points);
    printf("Case #%d: %.15e\n", tc+1, res/2);
  }
  fclose(f);
}
```

Capítulo X

Miscellaneous

10.1 Game Theory

Pontos importantes:

- *Teoria do Espelhamento*: Se o seu oponente pode espelhar todas as suas ações, este é um estado de derrota.
- Dois nim games são combinados usando o XOR.
- Podem existir ciclos modulares, vale brutar.
- Em algum momento pode se tornar sempre win, ou sempre loss.
- Se a gente transformar o problema num nim, podemos usar o teorema de grundy, que basicamente é achar os casos derrota + fazer o mex.

Example 1:

There is a heap of n coins and two players who move alternately. On each move, a player chooses a heap and divides into two nonempty heaps that have a different number of coins. The player who makes the last move wins the game.

Here for big N the answer is always first.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1000010;
const ll mod = 1000000007LL;
int dp[N];
int mex(set<int> s)
{
    int x = 0;
    while(s.find(x) != s.end())
        x++;
    return x;
}
int grundy(int n)
{
    if(n <= 2)
        return 0;
    if(dp[n]!=-1)
        return dp[n];
    set<int> s;
    for(int i = 1; i + i < n; i++)
    {
        s.insert(grundy(i)^grundy(n-i));
    }
    return dp[n] = mex(s);
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    memset(dp,-1,sizeof(dp));
    // for(int i = 1; i <= 50000; i++){
    //     if(grundy(i) == 0 )
```

```
    //          cout << i << endl;
    // }
    // cout << '\n';
    int q;
    cin >> q;
    while(q--)
    {
        int n;
        cin >> n;
        if(n > 1222)
            cout << "first\n";
        else
            cout << (grundy(n)?"first\n":"second\n");
    }
    return 0;
}
```

Example 2:

There is a staircase consisting of n stairs, numbered $1, 2, \ldots, n$. Initially, each stair has some number of balls.

There are two players who move alternately. On each move, a player chooses a stair k where $k \neq 1$ and it has at least one ball. Then, the player moves any number of balls from stair k to stair $k-1$. The player who moves last wins the game.

Here, every odd position is losing, u can see them as "dark holes", using the "Teoria do Espelhamento", so it's just N/2 Nim games.

```
int main()
{
    int q;
    cin >> q;
    while(q--)
    {
        int n;
        cin >> n;
        int rs = 0;
        for(int i = 0; i < n; i++)
        {
            int x;
            cin >> x;
            if(i&1)
                rs ^= x;
        }
        cout << (rs?"first":"second") << '\n';
    }
    return 0;
}
```

10.1.1 Nim Multiplication

A 2D NIM Game int the form of NXM can be reduced to the Nim Multiplication of 2 1D NIM GAMES in the form of NimMult(Grundy(N),Grundy(M))

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <cassert>

using namespace std;

#define forn(i,n) for(int i=0;i<int(n);i++)
#define all(c) begin(c), end(c)

#define SIZE(c) int((c).size())

typedef unsigned long long Nimber;

```

```
vector<int> exponents(Nimber value)
{
    vector<int> ret;
    unsigned long long x = value;
    for (; x; x = (x-1)&x)
        ret.push_back(__builtin_ctzll(x));
    return ret;
}

map<pair<Nimber,Nimber>, Nimber> cache;

Nimber nimProduct(Nimber a, Nimber b)
{
    auto it = cache.find(make_pair(a,b));
    if (it != cache.end())
        return it->second;
    Nimber &ret = cache[make_pair(a,b)];
    if (a == 0 || b == 0)
        ret = 0;
    else if (a == 1 || b == 1)
        ret = a^b^1;
    else
    {
        vector<int> aExponents = exponents(a);
        vector<int> bExponents = exponents(b);
        if (aExponents.size() == 1 && bExponents.size() == 1)
        {
            // Computes nim product of 2^a and 2^b
            // Decompose exponents = write as product of fermats
            vector<int> aExpBits = exponents(aExponents[0]);
            vector<int> bExpBits = exponents(bExponents[0]);
            #define FERMAT(index) Nimber(1ULL<<(1ULL<<(index)))
            ret = Nimber(1);
            int i = 0, j = 0;
            while (i < SIZE(aExpBits) && j < SIZE(bExpBits))
            {
                if (aExpBits[i] < bExpBits[j])
                    ret *= FERMAT(aExpBits[i++]);
                else if (aExpBits[i] > bExpBits[j])
                    ret *= FERMAT(bExpBits[j++]);
                else
                {
                    ret = (ret * FERMAT(aExpBits[i])) ^ nimProduct(ret, FERMAT(aExpBits[i])/2);
                    i++;
                    j++;
                }
            }
            for (; i < SIZE(aExpBits); i++) ret = ret * FERMAT(aExpBits[i]);
            for (; j < SIZE(bExpBits); j++) ret = ret * FERMAT(bExpBits[j]);
        }
        else
        {
            ret = 0;
            for (int aExp : aExponents)
            for (int bExp : bExponents)
                ret ^= nimProduct(1ULL<<aExp, 1ULL<<bExp);
        }
    }
    return ret;
}


int main()
```

```
{
    forn(i,16)
    forn(j,16)
        cout << i << " " << j << " " << nimProduct(i,j) << endl;
    return 0;
}
```

10.2 Binary Search

10.2.1 Parallel Binary Search

(Original Problem: Atcoder Grand Contest 2 D)

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
#define sd second
#define ft first
const int N=100100;
int pai[N],rk[N],que[N],lo[N],hi[N];
vector<int> lista[N],mid[N];

  vector<pair<pair<int,int>,int>> v;




void init(){
  for(int i=0;i<N;i++){
    pai[i]=i,rk[i]=1;
  }
}

int find(int x){
  if(x==pai[x])
    return x;
  return pai[x]=find(pai[x]);
}



void join(int a,int b){
  a=find(a);
  b=find(b);
  if(a!=b){
    if(rk[a]<rk[b])
      swap(a,b);
    pai[b]=a;
    rk[a]+=rk[b];
  }
}

vector<pair<int,int>> p;
int main(){
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  int n,m;
  int q,a,b,z;
  cin>>n>>m;
  int t=m;
  while(t--){
    cin>>a>>b;
    p.pb({a,b});
  }

```

```
  cin>>q;
  for(int i=0;i<q;i++){
    cin>>a>>b>>z;
    lo[i]=0,hi[i]=m;
    v.pb({{a,b},z});
  }
  bool need =true;
  while(need){
    for(int i=0;i<m;i++)
      mid[i].clear();
    for (int i = 0; i < q; ++i)
    {
      if(lo[i]<=hi[i])
      {
        mid[(lo[i]+hi[i])/2].pb(i);
      }
    }
    init();
    need=false;
    for (int i = 0; i < m; ++i)
    {
      join(p[i].ft,p[i].sd);
      for(auto at:mid[i])
      {
        need=true;
        int val;
        if(find(v[at].ft.ft)==find(v[at].ft.sd))
        {
          val=rk[find(v[at].ft.ft)];
        }else{
          val=rk[find(v[at].ft.ft)]+rk[find(v[at].ft.sd)];
        }
        if(val>=v[at].sd)
        {
          hi[at]=i-1;
          que[at]=i;
        }else{
          lo[at]=i+1;
        }
      }
    }

  }
  for (int i = 0; i < q; ++i)
  {
    cout<<que[i]+1<<"\n";
  }

}
```

10.3 Big Num

Implementação de Big Number em C++.

```
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;

int cmp(int a, int b) {
    if(a < b) return -1;
    else if (a == b) return 0;
    else return 1;
}

struct bigint {
```

```
    int v[TAM], n;
    bigint(int x = 0): n(1) {
        memset(v, 0, sizeof(v));
        v[n++] = x; fix();
    }
    bigint(char *s): n(1) {
        memset(v, 0, sizeof(v));
        int sign = 1;
        while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
        char *t = strdup(s), *p = t + strlen(t);
        while (p > t) {
            *p = 0; p = max(t, p - DIG);
            sscanf(p, "%d", &v[n]);
            v[n++] *= sign;
        }
        free(t); fix();
    }
    bigint& fix(int m = 0) {
        n = max(m, n);
        int sign = 0;
        for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
            v[i] += e; e = v[i] / BASE; v[i] %= BASE;
            if (v[i]) sign = (v[i] > 0) ? 1 : -1;
        }

        for (int i = n - 1; i > 0; i--)
            if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign;
   }
        while (n && !v[n]) n--;
        return *this;
    }

    int cmp(const bigint& x = 0) const {
        int i = max(n, x.n), t = 0;
        while(1){
            cout<<mp(v[i], x.v[i])<<"\n";
            if((t = ::cmp(v[i], x.v[i])) || i-- == 0){
                cout<<t<<"\n";
                return t;
            }
        }
    }
    bool operator <(const bigint& x) const { return cmp(x) < 0; }
    bool operator ==(const bigint& x) const { return cmp(x) == 0; }
    bool operator !=(const bigint& x) const { return cmp(x) != 0; }
    bool operator <=(const bigint& x) const { return cmp(x) <= 0; }
    bool operator >(const bigint& x) const { return cmp(x) > 0; }
    bool operator >=(const bigint& x) const { return cmp(x) >= 0; }

    operator string() const {
        ostringstream s; s << v[n];
        for (int i = n - 1; i > 0; i--) {
            s.width(DIG); s.fill('0'); s << abs(v[i]);
        }
        return s.str();
    }
    friend ostream& operator <<(ostream& o, const bigint& x) {
        return o << (string) x;
    }

    bigint& operator +=(const bigint& x) {
        for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
        return fix(x.n);
    }
    bigint operator +(const bigint& x) { return bigint(*this) += x; }
    bigint& operator -=(const bigint& x) {
```

```
        for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
        return fix(x.n);
    }
    bigint operator -(const bigint& x) { return bigint(*this) -= x; }
    bigint operator -() { bigint r = 0; return r -= *this; }
    void ams(const bigint& x, int m, int b) { // *this += (x * m) << b;
        for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
            v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
        }
    }
    bigint operator *(const bigint& x) const {
        bigint r;
        for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
        return r;
    }
    bigint& operator *=(const bigint& x) { return *this = *this * x; }
    // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
    bigint div(const bigint& x) {
        if (x == 0) return 0;
        bigint q; q.n = max(n - x.n + 1, 0);
        int d = x.v[x.n] * BASE + x.v[x.n-1];
        for (int i = q.n; i > 0; i--) {
            int j = x.n + i - 1;
            q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
            ams(x, -q.v[i], i-1);
            if (i == 1 || j == 1) break;
            v[j-1] += BASE * v[j]; v[j] = 0;
        }
        fix(x.n); return q.fix();
    }
    bigint& operator /=(const bigint& x) { return *this = div(x); }
    bigint& operator %=(const bigint& x) { div(x); return *this; }
    bigint operator /(const bigint& x) { return bigint(*this).div(x); }
    bigint operator %(const bigint& x) { return bigint(*this) %= x; }
    bigint pow(int x) {
        if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
        bigint r = 1;
        for (int i = 0; i < x; i++) r *= *this;
        return r;
    }
    bigint root(int x) {
        if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
        if (*this == 1 || x == 1) return *this;
        if (cmp() < 0) return -(-*this).root(x);
        bigint a = 1, d = *this;

        while (d != 1) {
            bigint b = a + (d /= 2);
            if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
        }
        return a;
    }
};
```

10.3.1 D&C for Multiplyng two big numbers

```
typedef vector<int> poly;

poly mult(const poly& p, const poly& q) {
    int sz = p.size(), half = sz/2;
    assert(sz == q.size() && !(sz&(sz-1)));

    if(sz <= 64) {
        poly ret(2*sz);
        for(int i = 0; i < sz; i++)
```

```
            for(int j = 0; j < sz; j++)
                ret[i+j] += p[i] * q[j];
        return ret;
    }

    poly p1(p.begin(), p.begin() + half), p2(p.begin() + half, p.end());
    poly q1(q.begin(), q.begin() + half), q2(q.begin() + half, q.end());
    poly p1p2(half), q1q2(half);
    for(int i = 0; i < half; i++)
        p1p2[i] = p1[i] + p2[i], q1q2[i] = q1[i] + q2[i];

    poly low = mult(p1, q1), high = mult(p2, q2), mid = mult(p1p2, q1q2);
    for(int i = 0; i < sz; i++)
        mid[i] -= high[i] + low[i];

    low.resize(2*sz);
    for(int i = 0; i < sz; i++)
        low[i+half] += mid[i], low[i+sz] += high[i];

    return low;
}
```

10.4 Bitsets

Some $O(N^2)$ solutions can be optimized using bitsets. Mainly knapsack problems. An important point is to use the "popcount"optimization in target of pragma.

how to initialize:

```
bitset<size> variable_name;
bitset<size> variable_name(DECIMAL_NUMBER);
bitset<size> variable_name("BINARY_STRING")
```

Every position can be accessed as a vector, also size must be constant. We get an constant optimization of 32. every bit operation used in integers can be used in bitsets.

Some functions in bitsets:

set()	Set the bit value at all bitset to 1.
reset()	Set the bit value at all bitset to 0.
flip()	Flip the bit value at all bitset.
set(idx)	Set the bit value at the given idx to 1.
reset(idx)	Set the bit value at a given idx to 0.
flip(idx)	Flip the bit value at the given idx.
count()	Count the quantity of bits set.
any()	Checks if any bit is set.
all()	Check if all bit is set.
none()	Checks if none bit is set.
size()	Returns the size of the bitset.
to_string()	Converts bitset to std::string.
to_ulong()	Converts bitset to unsigned long.
to_ullong()	Converts bitset to unsigned long long.
_Find_first()	return index of first bit set.
_Find_next(idx)	return index of first bit set after idx.

10.5 Built in functions

Some Built in functions in GCC(remember that if u are using ll, need to add ll at the end):

__builtin_popcount(x)	Counts the quantity of one's(set bits) in an integer.
__builtin_parity(x)	Checks the Parity of a integer. Returns true(1) if odd parity(odd quantity of set bits) Returns false(0) for even parity(even quantity of set bits).
__builtin_clz(x)	Counts the leading quantity of zeros of the integer.
__builtin_ctz(x)	Counts the trailing quantity of zeros of the integer.
__builtin_popcountll(x)	Counts the quantity of one's(set bits) in an long long.
__builtin_parityll(x)	Checks the Parity of a long long. Returns true(1) if odd parity(odd quantity of set bits) Returns false(0) for even parity(even quantity of set bits).
__builtin_clzll(x)	Counts the leading quantity of zeros of the long long.
__builtin_ctzll(x)	Counts the trailing quantity of zeros of the long long.

clz can be used to find the first set bit, since u can use:

```
int x;
ll xl;
int fsi =  31 - __builtin_clz(x);
int fsll = 63 - __builtin_clzll(xl);
```

10.6 Priority Queue and Set Comparators

Set Comparators Operator Overloading:

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

struct Edge {
  int a, b, w;
  bool operator<(const Edge &y) const { return w < y.w; }
};

int main() {
  int M = 4;
  set<Edge> v;
  for (int i = 0; i < M; ++i) {
    int a, b, w;
    cin >> a >> b >> w;
    v.insert({a, b, w});
  }
  for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";
}
```

Functors:

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

struct Edge {
  int a, b, w;
};

struct cmp {
  bool operator()(const Edge &x, const Edge &y) const { return x.w < y.w
   ; }
};

int main() {
  int M = 4;
  set<Edge, cmp> v;
  for (int i = 0; i < M; ++i) {
    int a, b, w;
    cin >> a >> b >> w;
    v.insert({a, b, w});
  }
  for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";
}
```

Functors can also be used in priority queues, but needs vector as a container:

```
priority_queue<int, vector<int>, cmp> c;
```

Built in Functors:

```
set<int, greater<int>> a; //max set
map<int, string, greater<int>> b; // max map
priority_queue<int, vector<int>, greater<int>> c; // min heap
```

10.7 Lambda Expressions

Basic Lambda Syntax A lambda expression consists of the following:

[capture list] (parameter list) function body

The capture list and parameter list can be empty, so the following is a valid lambda:

```
[](){cout << "Hello, world!" << endl;}
auto func1 = [](int i) {cout << ":" << i << ":";};
func1(x);
```

Using & in capture list it will have access to the scope.

Recursive lambda using function:

```
int main() {
  function<int(int, int)> gcd = [&](int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
  };
  cout << gcd(20, 30) << '\n';  // outputs 10
}
```

10.8 Fast input output

```
inline int next_int() {
    int n = 0;
    char c = getchar_unlocked();
  bool neg = false;
    while (!(c >= '0' && c <= '9')){
    if(c == '-'){
      c = getchar_unlocked();
      if(c >= '0' && c <= '9'){
        neg = true;
      }
    }else c = getchar_unlocked();
    }

    while ('0' <= c && c <= '9') {
    n = n * 10 + c - '0';
    c = getchar_unlocked();
    }

  if(neg) return -n;
  else return n;
}

inline char next_char(){
  char c = getchar_unlocked();
  while(c == ' ' || c == '\n') c = getchar_unlocked();
  return c;
}

inline string next_string(){
  string out;
  char c = getchar_unlocked();
  while(c == ' ' || c == '\n') c = getchar_unlocked();
  while(!(c == ' ' || c == '\n')){
```

```
        out.pb(c);
        c = getchar_unlocked();
    }

    return out;
}


bool read( int &n ) {
    n = 0;
    register bool neg = false;
    register char c = getchar_unlocked();
    if( c == EOF) { n = -1; return false; }
    while (!('0' <= c && c <= '9')) {
        if( c == '-' ) neg = true;
        c = getchar_unlocked();
    }
    while ('0' <= c && c <= '9') {
        n = n * 10 + c - '0';
        c = getchar_unlocked();
    }
    n = (neg ? (-n) : (n));
    return true;
}

inline void writeInt(int n){
    register int idx = 20;
    if( n < 0 ) putchar_unlocked('-');
    n = abs(n);
    char out[21];
    out[20] = ' ';
    do{
        idx--;
        out[idx] = n % 10 + '0';
        n/= 10;
    }while(n);
    do{ putchar_unlocked(out[idx++]); } while (out[idx] != ' ');
}
```

10.9 Trick for faster Unordered Map

```
struct pair_hash {
    size_t operator()(const pair<int,int>&x)const{
        return hash<ll>()(((ll)x.first)^(((ll)x.second)<<32));
      }
};
unordered_map<pair<int,int>,int,pair_hash> mp;

int main()
{
    //usar o tamanho esperado do umap
    mp.reserve(N*4);
    mp.max_load_factor(0.25);
}
```

10.10 Faster Hash Table

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

//hash definida
```

```
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;

//hash de pair
struct chash {
    int operator()(pii x) const { return x.first* 31 + x.second; }
};

gp_hash_table<pii, int, chash> table;
// hash de vector
struct VectorHasher {
    int operator()(const vector<int> &V) const {
        int hash = V.size();
        for(auto &i : V) {
            hash ^= i + 0x9e3779b9 + (hash << 6) + (hash >> 2);
        }
        return hash;
    }
};
```

10.11 StringStream

A way to parse strings in c++.

```
//istringstream
//ler linhas do tipo: +0 +2 -0 +1 -1 -2
string s;
istringstream iss;
rep(i, 0, n){
    getline(cin, s);
    iss.clear();
    iss.str(s);

    while(iss >> ch){
        iss >> in;
    }
}

//ostringstream
ostringstream oss;

int idade1 = 30;
string nome1 = "Alice";
double salario1 = 2500.75;
oss << "óRelatrio 1\n";
oss << "Nome: " << nome1 << ", Idade: " << idade1 << ", áSalrio: R$ " <<
    salario1 << "\n";
cout << oss.str() << endl;

oss.str("");
oss.clear();

int idade2 = 45;
string nome2 = "Bob";
double salario2 = 3200.50;
oss << "óRelatrio 2\n";
oss << "Nome: " << nome2 << ", Idade: " << idade2 << ", áSalrio: R$ " <<
    salario2 << "\n";
cout << oss.str() << endl;
```

10.12 Karmarkar-Karp

Heuristic algorithm to divide a set in two others set, in the way that the difference between the sum of this two new sets will be minimal. The error of this heuristc algorithm is $\mathcal{O}(n^{-lg(n)}) = \mathcal{O}(\frac{1}{n^{lg(n)}})$

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

int karmakarp(multiset<int>& items, multiset<int>& P1, multiset<int>& P2
    ){

    if (items.size() == 1){
        P1.clear();
        P1.insert(*items.begin());
        P2.clear();
        return *items.begin();
    }

    int bigger1 = *items.rbegin();
    items.erase(items.find(*items.rbegin()));
    int bigger2 = *items.rbegin();
    items.erase(items.find(*items.rbegin()));

    int diff = bigger1 - bigger2;
    items.insert(diff);

    int bestDiff = karmakarp(items, P1, P2);

    if (P1.find(diff) == P1.end()){
        swap(P1, P2);
    }

    P1.erase(P1.find(diff));
    P1.insert(bigger1);

    P2.insert(bigger2);
    return bestDiff;
}

int main(){

    vector<int> vet = {3, 8, 12, 16, 38, 41, 73};
    multiset<int> P1, P2;

    multiset<int> items;
    for (int v : vet)
        items.insert(v);

    cout << karmakarp(items, P1, P2) << "\n";

    for (int v : P1)
        cout << v << " ";
    cout << "\n";

    for (int v : P2)
        cout << v << " ";
    cout << "\n";

    return 0;
}
```

10.13 Fractions

Implementation of fractions.

```
template <class TT = ll>
```

```
struct Frac{
    TT num,den; // o valor negativo da fracao fica so no numerador

    Frac (TT num_ = 0,TT den_ = 1) : num(num_), den(den_){
        TT g = __gcd(num,den);
        num/=g;
        den/=g;
        fixSig(num,den);
    }
    void fixSig(TT & num_, TT & den_){
        if((num_ < 0 && den_ < 0) || (num_ > 0 && den_ > 0)){
            num_=abs(num_);
            den_=abs(den_);
        }else if(num_ > 0){
            num_*=-1;
            den_*=-1;
        }
    }

    TT fexp(TT a, TT b){
        TT ans = 1;
        while(b){
            if(b&1) ans = ans*a;
            b>>=1;
            a = a*a;
        }
        return ans;
    }

    TT fexp(TT a, TT b, TT mod){
        TT ans = 1;
        while(b){
            if(b&1) ans = ans*a%mod;
            b>>=1;
            a = a*a%mod;
        }
        return ans;
    }

    friend ostream &operator<<(ostream & os, const Frac & o){
        os << o.num << "/" << o.den;
        return os;
    }

    bool operator <(const Frac & o)const{
        if((den < 0 && o.den < 0) || (den > 0 && o.den > 0)) return num*
    o.den < o.num*den;
        return num*o.den > o.num*den;
    }

    bool operator ==(const Frac &o){
        return num == o.num && den == o.den;
    }

    bool operator !=(const Frac &o){
        return num != o.num || den != o.den;
    }

    friend Frac &operator -(Frac & at){
        at.num *= -1;
        return at;
    }

    void operator *=(const Frac & o){
        TT g1 = __gcd(num,o.den);
        TT g2 = __gcd(den,o.num);
```

```
        num = (num/g1)*(o.num/g2);
        den = (den/g2)*(o.den/g1);
        fixSig(num,den);
    }
    void operator /=(const Frac & o){
        TT g1 = __gcd(num,o.num);
        TT g2 = __gcd(den,o.den);
        num = (num/g1)*(o.den/g2);
        den = (den/g2)*(o.num/g1);
        fixSig(num,den);
    }
    void operator +=(const Frac & o){
        TT lcm = den/__gcd(den,o.den)*o.den;
        num = (lcm/den*num) + (lcm/o.den*o.num);
        den = lcm;
        TT g = __gcd(num,den);
        num/=g, den/=g;
        fixSig(num,den);
    }
    void operator -=(const Frac & o){
        TT lcm = den/__gcd(den,o.den)*o.den;
        num = (lcm/den*num) + (lcm/o.den*o.num);
        den = lcm;
        TT g = __gcd(num,den);
        num/=g, den/=g;
        fixSig(num,den);
    }

    Frac operator *(const Frac & o){
        TT g1 = __gcd(num,o.den);
        TT g2 = __gcd(den,o.num);
        return Frac((num/g1)*(o.num/g2), (den/g2)*(o.den/g1));
    }
    Frac operator /(const Frac & o){
        TT g1 = __gcd(num,o.num);
        TT g2 = __gcd(den,o.den);
        return Frac((num/g1)*(o.den/g2), (den/g2)*(o.num/g1));
    }
    Frac operator +(const Frac & o){
        TT lcm = den/__gcd(den,o.den)*o.den;
        return Frac((lcm/den*num) +(lcm/o.den*o.num),lcm);
    }
    Frac operator -(const Frac & o){
        TT lcm = den/__gcd(den,o.den)*o.den;
        return Frac((lcm/den*num) -(lcm/o.den*o.num),lcm);
    }

    // exponenciacao
    void operator ^=(TT x){
        if(num == 0) return;
        if(x < 0){
            swap(num,den);
            x*=-1;
        }
        TT aux = 1;
        num = fexp(num,x);
        den = fexp(den,x);
    }

    TT modular(TT mod){
        return num*fexp(den,mod-2,mod)%mod;
    }
};
```