

Monkeys LIB
Cerrado Edition
Monkeys



2024

Sumário

I	Begin	5
1.1	Who did this	5
1.2	tips	5
1.3	template - makefile - terminaltricks	7
1.4	math formulas	9
1.5	Stress Test	9
1.5.1	Vector Generator	10
1.5.2	Tree Generator	11
II	Algebra	13
2.1	Inteiro Modular	13
2.2	Matrix Power	14
2.2.1	Graph Paths I	14
2.2.2	Graph Paths II	15
2.3	FFT	16
2.3.1	Simple FFT	16
2.3.2	FFT modular (NTT)	19
2.3.3	FFT To solve String with Wildcards	21
2.3.4	FWHT (fft using xor/and/or)	25
2.4	Gauss	26
2.4.1	Markov Chains	26
2.4.2	Modular 2 (bitset)	28
2.4.3	Matrix rank	30
2.4.4	Xor Basis	32
2.5	Chinese Remainder Theorem	32
2.5.1	Algorithm	32
2.5.2	Lucas Theorem	33
2.6	Simplex	35
2.7	GCD Array Trick	36
2.8	Implementation of Polynomials	37
2.9	Combinatorial	38
2.9.1	Derangements	40
2.10	Lagrange Interpolation	40
2.11	Discrete Log	42
2.12	Nelsi's anotations	43
2.12.1	Kaplansky's Lemma	43
2.12.2	Divisor Analysis - CSES	44
2.12.3	Bracket Sequences II - CSES	44
2.13	Multiplicative Functions	45
2.13.1	Mobius Inversion	46
III	Data Structures	49
3.1	Ordered Set	49
3.2	Fenwick Tree	49
3.2.1	Simples	49
3.2.2	2D	50
3.2.3	Fenwick Lifting	51
3.3	Sparse Table	52
3.3.1	Simple Sparse Table	52
3.3.2	Disjoint Sparse Table	53
3.4	SQRT Decomposition	55
3.4.1	MO	55
3.4.2	MO com updates	56
3.4.3	Blocking	58
3.4.4	Blocking com cyclic shift	59
3.4.5	Batching	62

3.5	Treap	62
3.6	DSU	64
3.6.1	Persistent DSU	64
3.6.2	DSU with rollback	65
3.6.3	Connected Components With Segments	65
3.6.4	Dynamic Connectivity Offline	67
3.6.5	DSU with Small to Large	69
IV	Segment Tree	73
4.1	Sem/Com Lazy	73
4.2	Implicita	75
4.3	Implicita com lazy	75
4.4	Persistente	76
4.5	update é uma PA	78
4.6	Sub-segmento contiguo com maior soma	80
4.7	Contar minimos (união de area de retangulos)	80
4.8	Merge Sort Tree	82
4.9	Segment Tree Beats	83
4.10	Wavelet Tree	87
V	Dynammmic Programming	89
5.1	Fast Knapsack 3k trick	89
5.2	SOS DP	91
5.3	Permutation Trick	93
5.4	Open Interval Trick	94
5.5	Dp Optimizations	94
5.5.1	D&C	95
5.5.2	Knuth Optimization	96
5.5.3	Convex Hull Trick	96
5.6	Steiner Tree DP	101
VI	String	103
6.1	String Hash	103
6.1.1	Simple Hash	103
6.1.2	Hash 2D _{ayllon}	104
6.1.3	Hash with Updates	106
6.2	Suffix Array	108
6.3	Suffix Automaton	109
6.4	Suffix Tree	112
6.5	Palindromic Tree	113
6.6	Minimal Rotation	116
6.7	Aho Corasick	116
6.8	Trie	117
VII	Graph	119
7.1	Flow	119
7.1.1	Teoremas de Fluxo	119
7.1.2	Dinic	120
7.1.3	MinCostMaxFlow	121
7.2	Matching	123
7.2.1	HopCroft Karp	123
7.2.2	Bipartide Weighted Hungarian Method	125
7.2.3	General Graph (Edmonds Blossom)	127
7.3	Bellman Ford	130
7.4	2-SAT	131
7.5	Componentes Fortemente Conexas	132
7.6	Stable Marriage	134
7.7	Planar Graphs	135
7.8	Dominator Tree	135
7.9	Chinese Postman Problem	137
7.10	Some Bridge related topics	138
7.10.1	Finding Bridges	138
7.10.2	Articulation Points	139
7.10.3	Block Cut Tree	140
7.10.4	Two Edge Connected Components	143

8.1	Diameter	145
8.2	LCA	145
8.3	Euler tour tree	148
8.4	Small to Large	148
8.5	HLD	148
8.5.1	Can you answer these queries VII	151
8.5.2	Can you answer these queries VI	155
8.5.3	HLD with subtree queries	160
8.6	Centroid Decomposition	161
8.7	Tree Isomorphism	168
8.8	Virtual Tree	170
8.9	Link Cut Tree	172
IX	Geometry	175
9.1	Simple Geometry	175
9.1.1	Points and Lines	175
9.1.2	Circles	177
9.2	Convex Hull	180
9.2.1	Rotation Calipers	180
9.3	Line Sweep	182
9.3.1	Points Inside Triangles	182
9.3.2	Ranking Problem	187
9.3.3	Balls Falls and Segments	189
9.3.4	Checking Points Inside Convex Polygon	193
9.3.5	Radial Sweep	194
9.3.6	Radial Sweep sem Double	196
9.4	Minimum Perimeter Triangle	197
X	Miscellaneous	201
10.1	Game Theory	201
10.1.1	Nim Multiplication	202
10.2	Binary Search	204
10.2.1	Parallel Binary Search	204
10.3	Big Num	205
10.3.1	D&C for Multipl yng two big numbers	207
10.4	Bitsets	208
10.5	Built in functions	208
10.6	Priority Queue and Set Comparators	209
10.7	Lambda Expressions	210
10.8	Fast input output	210
10.9	Trick for faster Unordered Map	211
10.10	Faster Hash Table	211
10.11	StringStream	212
10.12	Karmarkar-Karp	213

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

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //template
5 #pragma region
6
7 #include <ext/pb_ds/assoc_container.hpp> // Common file
8 #include <ext/pb_ds/tree_policy.hpp> // Including
   tree_order_statistics_node_update
9 using namespace __gnu_pbds;
10 template<class T> using ordset = tree<T,null_type, less<T>,
11 rb_tree_tag, tree_order_statistics_node_update>;
12
13 // #pragma GCC optimize("Ofast") // for fast N^2
14 // #pragma GCC target("avx2") // for fast N^2
15 // #pragma GCC target("popcnt") // for fast popcount
16
17 #define all(x) x.begin(), x.end()
18 #define lef(x) (x << 1)
19 #define rig(x) (lef(x) | 1)
20
21 using ll = long long int;
22 using ld = long double;
23 using pll = pair<ll,ll>;
24 using pii = pair<int,int>;
25 using pdb = pair<ld,ld>;
26
27 //read and print pair
28 template<typename T, typename T1>
29 ostream & operator<<(ostream &os, pair<T, T1> p){
30     os << "(" << p.ft << "," << p.sd << ")";
31     return os;
32 }
33 template<typename T, typename T1>
34 istream & operator>>(istream &is, pair<T, T1>& p){
35     is >> p.ft >> p.sd;
36     return is;
37 }
38
39 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
40 #pragma endregion
41
42 void test() {
43
44 }
45
46 int32_t main() {
47     ios::sync_with_stdio(false), cin.tie(nullptr);
48     int ttt_ = 1;
49     // cin >> ttt_;
50     while(ttt_-->0) test();
51     return 0;
52 }

```

```

1 // 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.
2
3 CXX = g++
4 CXXFLAGS = -fsanitize=address -fno-omit-frame-pointer -g -O2 -Wall -
   Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare
5
6 // 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.

```
7  
8 CXX = g++  
9 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
```

```
1 command time -v ./out //saber a alocao e mais uma porrada de coisas  
roubadas  
2 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 $f(x_1, x_2, \dots, x_n)$ subject to the constraints $g_i(x_1, x_2, \dots, x_n) = k_i$ 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:

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

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

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

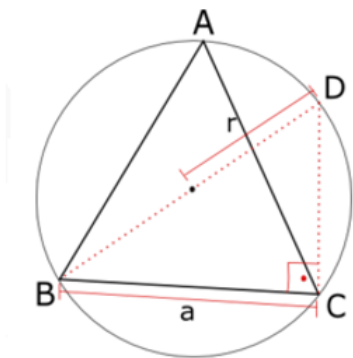


Figura I.1: Lei dos senos

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

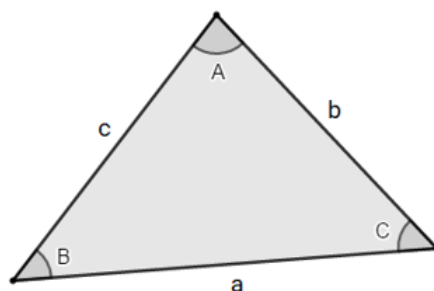


Figura I.2: Lei dos cossenos

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:

```

1 #!/bin/bash
2
3 make brute
4 make gen
5 make at # codigo que muda
6
7 while true; do
8     ./gen 2 > in
9
10    start=$(date +%s)
11    ./at < in > out
12    end=$(date +%s)
13
14    echo "Elapsed Time in test $c: $((($end - $start)) seconds."
15    ./brute < in > aout
16    diff -B out aout > /dev/null || break
17    echo "Passou no caso."
18    cat out
19 done
20
21 echo "WA on the following case:"
22 cat in
23 echo "Your answer is:"
24 cat out
25 echo "Correct answer is:"
26 cat aout
27
28 // Para mais de uma solucao: mandar para um validador e deixar ele
29    resolver tudo.
30
31 #!/bin/bash
32
33 make brute          # fazer o codigo forca bruta do problema
34 make gen            # fazer o gerador do problema
35 make solution       # a sua solucao do problema
36 make validator      # o validador da sua solucao
37
38 for((c = 0; c <= 500; c++)); do
39     ./gen 2 > in
40     start = $(date +%s)
41     ./solution < in > out
42     end = $(date +%s)
43     echo "Elapsed Time in test $c: $((($end - $start)) seconds."
44     ./brute < in > a_in
45     ./validator < a_in > aout
46     diff -B out aout > /dev/null || break
47     echo "Passou no caso $c."
48 done
49
50 echo "WA on the following case:"
51 cat in
52 echo "Your answer is:"
53 cat aout
54 echo "The possible solutions are:"
55 cat out

```

1.5.1 Vector Generator

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

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //template
4

```

```

5 int main(int argc, char *argv[]) {
6     int n = stoi(argv[1]);
7     int lim = stoi(argv[2]);
8     cout << n << '\n';
9     rep(i, 0, n)
10         cout << (rng()%lim) << ' ';
11     cout << '\n';
12     return 0;
13 }

```

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).

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 //template
4
5 vector<pii> edges;
6 int n;
7
8 void random_tree()
9 {
10     rep(i, 1, n)
11         edges.eb(i, rng()%i);
12 }
13 void catter()
14 {
15     int k = rng()%(n/2) + n/2;
16     rep(i, 1, k)
17         edges.eb(i-1, i);
18     rep(i, k, n)
19         edges.eb(rng()%(i), i);
20 }
21
22 void star()
23 {
24     rep(i, 1, n)
25         edges.eb(0, i);
26 }
27
28 void print_tree()
29 {
30     vector<int> p;
31     rep(i, 1, n+1)
32         p.pb(i);
33     shuffle(all(p), rng);
34     shuffle(all(edges), rng);
35     cout << n << '\n';
36     for(auto [u, v]: edges)
37         cout << p[u] << ' ' << p[v] << '\n';
38 }
39
40 int32_t main(int argc, char *argv[])
41 {
42     n = stoi(argv[1]);
43     int tp = stoi(argv[2]);
44     if(!tp)
45         star();
46     else
47     {
48         if(tp%3 == 0)
49             catter();
50         else
51             random_tree();

```

```
52     }
53     print_tree();
54     return 0;
55 }
```

Capítulo II

Algebra

2.1 Inteiro Modular

Classe para lidar com problemas relacionados a operações modulares.

```
1 const int MOD = 1'000'000'007;
2
3 struct mi {
4     int v;
5     explicit operator int() const { return v; }
6     mi() { v = 0; }
7     mi(long long int _v) : v(_v % MOD + (_v % MOD < 0) * MOD) {}
8
9     friend mi operator -(const mi & at){
10         return mi(-at.v);
11     }
12
13     void operator +=(const mi & ot){
14         v += ot.v;
15         v -= (v >= MOD) * MOD;
16     }
17     void operator -=(const mi & ot) {
18         v -= ot.v;
19         v += (v < 0) * MOD;
20     }
21     void operator *=(const mi & ot){
22         v = 1ll * v * ot.v % MOD;
23     }
24     void operator /=(const mi & ot){
25         v = 1ll * v * inv(ot).v % MOD;
26     }
27     friend mi operator +(const mi & a, const mi & b){
28         return mi(a.v + b.v);
29     }
30     friend mi operator -(const mi & a, const mi & b){
31         return mi(a.v - b.v);
32     }
33     friend mi operator *(const mi & a, const mi & b){
34         return mi(1ll * a.v * b.v);
35     }
36     friend mi operator /(const mi & a, const mi & b){
37         return mi(1ll * a.v * inv(b).v);
38     }
39
40     friend mi fexp(mi a, long long int b){
41         mi ans(1);
42         while(b){
43             if(b&1) ans = ans*a;
44             a *= a;
45             b >>= 1;
46         }
47         return ans;
48     }
49     friend mi inv(const mi & a){
```

```

50         assert(a.v != 0);
51         return fexp(a, MOD - 2);
52     }
53
54     friend ostream& operator <<(ostream & out, const mi & at){
55         return out << at.v;
56     }
57     friend istream& operator >>(istream & in, mi & at){
58         return in >> at.v;
59     }
60 };

```

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.

```

1
2 typedef long long ll;
3 const int N = 100;
4 const ll mod = 1000000007LL;
5 ll mat[N][N], resp[N][N], aux[N][N];
6 void mult(ll a[N][N], ll b[N][N])
7 {
8     for(int i = 0; i < N; i++)
9         for(int j = 0; j < N; j++){
10             aux[i][j] = 0;
11             for(int k = 0; k < N; k++)
12                 aux[i][j] = (aux[i][j] + a[i][k] * b[k][j] % mod) % mod;
13         }
14
15     for(int i = 0; i < N; i++)
16         for(int j = 0; j < N; j++)
17             a[i][j] = aux[i][j];
18 }
19 void show(ll a[N][N], int n)
20 {
21     for(int i = 0; i < n; i++){
22         for(int j = 0; j < n; j++)
23             cout << a[i][j] << ' ';
24         cout << '\n';
25     }
26 }
27 void fasexp(ll b)
28 {
29     for(int i = 0; i < N; i++){
30         resp[i][i] = 1;
31     }
32     while(b)
33     {
34         if(b&1)
35             mult(resp, mat);
36         mult(mat, mat);
37         b >>= 1;
38     }
39 }
40 int dp[N];
41
42 int main()
43 {
44     int n, m;
45     ll k;
46     cin >> n >> m >> k;
47     while(m --)

```

```

48     {
49         int u, v;
50         cin >> u >> v;
51         u--, v--;
52         mat[u][v]++;
53     }
54     fasexp(k);
55     // show(resp, n);
56     cout << resp[0][n-1] << '\n';
57     return 0;
58 }

```

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.

```

1
2 typedef long long ll;
3 const int N = 100;
4 ll mat[N][N], resp[N][N], aux[N][N];
5 void mult(ll a[N][N], ll b[N][N])
6 {
7     for(int i = 0; i < N; i++)
8         for(int j = 0; j < N; j++){
9             aux[i][j] = -1;
10            for(int k = 0; k < N; k++){
11                if(a[i][k] == -1 || b[k][j] == -1)
12                    continue;
13                if(aux[i][j] == -1)
14                    aux[i][j] = a[i][k] + b[k][j];
15                aux[i][j] = min(aux[i][j], a[i][k] + b[k][j]);
16            }
17        }
18
19    for(int i = 0; i < N; i++)
20        for(int j = 0; j < N; j++)
21            a[i][j] = aux[i][j];
22 }
23 void fasexp(ll b)
24 {
25     while(b)
26     {
27         if(b&1)
28             mult(resp, mat);
29         mult(mat, mat);
30         b >>= 1;
31     }
32 }
33 int dp[N];
34
35 int main()
36 {
37     ios::sync_with_stdio(false);
38     cin.tie(NULL);
39     int n, m;
40     ll k;
41     cin >> n >> m >> k;
42     for(int i = 0; i < N; i++){
43         for(int j = 0; j < N; j++){
44             mat[i][j] = -1;
45             resp[i][j] = -1;
46         }
47         resp[i][i] = 0;
48     }
49     while(m --)

```

```

50 {
51     int u, v, c;
52     cin >> u >> v >> c;
53     u--, v--;
54     if(mat[u][v] == -1)
55         mat[u][v] = c;
56     mat[u][v] = min(mat[u][v], (11)c);
57 }
58 fasexp(k);
59 cout << resp[0][n-1] << '\n';
60 return 0;
61 }

```

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.

```

1 struct FFT{
2     typedef complex<double> C;
3     typedef vector<double> vd;
4     typedef vector<long long int> vl;
5     typedef vector<int> vi;
6
7     /*
8      * Author: Ludo Pulles, chilli, Simon Lindholm
9      * Date: 2019-01-09
10     * License: CC0
11     * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf (do
12     read, it's excellent)
13     Accuracy bound from http://www.daemonology.net/papers/fft.pdf
14     * Description: fft(a) computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx / N)$  for all  $k$ .  $N$  must be a power of 2.
15     Useful for convolution:
16     \texttt{conv(a, b) = c}, where  $c[x] = \sum a[i]b[x-i]$ .
17     For convolution of complex numbers or more than two vectors: FFT
18     , multiply
19     pointwise, divide by n, reverse(start+1, end), FFT back.
20     Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ 
21     (in practice  $10^{16}$ ; higher for random inputs).
22     Otherwise, use NTT/FFTMod.
23     * Time:  $O(N \log N)$  with  $N = |A| + |B|$  ( $\tilde{1}s$  for  $N = 2^{22}$ )
24     * Status: somewhat tested
25     * Details: An in-depth examination of precision for both FFT and
26     FFTMod can be found
27     * here (https://github.com/simonlindholm/fft-precision/blob/master/fft-precision.md)
28     */
29     void fft(vector<C>& a) {
30         int n = a.size(), L = 31 - __builtin_clz(n);
31         static vector<complex<long double>> R(2, 1);
32         static vector<C> rt(2, 1); // (~ 10% faster if double)

```



```

30     for (static int k = 2; k < n; k *= 2) {
31         R.resize(n); rt.resize(n);
32         auto x = polar(1.0L, acos(-1.0L) / k);
33         for(int i=k; i<2*k; i++) rt[i] = R[i] = i&1 ? R[i/2] * x : R
[i/2];
34     }
35     vi rev(n);
36     for(int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L)
/ 2;
37     for(int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i
]]);
38     for (int k = 1; k < n; k *= 2)
39         for (int i = 0; i < n; i += 2 * k) for(int j = 0; j < k; j
++ ) {
40             // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
rolled) /// include-line
41             auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];
/// exclude-line
42             C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
/// exclude-line
43             a[i + j + k] = a[i + j] - z;
44             a[i + j] += z;
45         }
46     }
47     // multiplica dois polinomios
48     vd conv(const vd& a, const vd& b) {
49         if (a.empty() || b.empty()) return {};
50         vd res(a.size() + b.size() - 1);
51         int L = 32 - __builtin_clz(res.size()), n = 1 << L;
52         vector<C> in(n), out(n);
53         copy(a.begin(), a.end(), in.begin());
54         for(int i = 0; i < b.size(); i++) in[i].imag(b[i]);
55         fft(in);
56         for (C& x : in) x *= x;
57         for(int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i
]);
58         fft(out);
59         for(int i = 0; i < res.size(); i++) res[i] = imag(out[i]) / (4 *
n);
60         return res;
61     }
62     vl conv(const vl& a, const vl& b) {
63         if (a.empty() || b.empty()) return {};
64         vl res(a.size() + b.size() - 1);
65         int L = 32 - __builtin_clz(res.size()), n = 1 << L;
66         vector<C> in(n), out(n);
67         copy(a.begin(), a.end(), in.begin());
68         for(int i = 0; i < b.size(); i++) in[i].imag(b[i]);
69         fft(in);
70         for (C& x : in) x *= x;
71         for(int i = 0; i < n; i++) out[i] = in[-i & (n - 1)] - conj(in[i
]);
72         fft(out);
73         for(int i = 0; i < res.size(); i++) res[i] = round(imag(out[i])
/ (4 * n));
74         return res;
75     }
76
77     /*
78     * Author: chilli
79     * Date: 2019-04-25
80     * License: CC0
81     * Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf
82     * Description: Higher precision FFT, can be used for
convolutions modulo arbitrary integers
83     * as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (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  $[0, \text{mod})$ .
    * 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;
    }
}

```

```

140     vi rev(n);
141     for(int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L)
/ 2;
142     for(int i = 0; i < n; i++) if (i < rev[i]) swap(a[i], a[rev[i]])
;
143     for (int k = 1; k < n; k *= 2)
144         for (int i = 0; i < n; i += 2 * k) for(int j = 0; j < k; j
++ ) {
145             ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
146             a[i + j + k] = ai - z + (z > ai ? mod : 0);
147             ai += (ai + z >= mod ? z - mod : z);
148         }
149     }
150     vl conv_ntt(const vl &a, const vl &b) {
151         if (a.empty() || b.empty()) return {};
152         int s = a.size() + b.size() - 1, B = 32 - __builtin_clz(s),
153             n = 1 << B;
154         int inv = modpow(n, mod - 2);
155         vl L(a), R(b), out(n);
156         L.resize(n), R.resize(n);
157         ntt(L), ntt(R);
158         for(int i = 0; i < n; i++)
159             out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;
160         ntt(out);
161         return {out.begin(), out.begin() + s};
162     }
163     ll modpow(ll b, ll e) {
164         ll ans = 1;
165         for (; e; b = b * b % mod, e /= 2)
166             if (e & 1) ans = ans * b % mod;
167         return ans;
168     }
169 };

```

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

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

$root_1 = root^{-1}$

$root_{pw} = 2^k$

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5
6 const ll mod = 998244353;
7 const int root = 565042129; //3^(mod/root_pw)
8 const int root_1 = 950391366; //inv(root)
9 const int root_pw = 1<<20;
10 const int N=100100;
11
12 int fat[N];
13 vector<int> a[N];
14 int exp(int n,int b)
15 {
16     int rs=1;
17     while(b)
18     {
19         if(b&1)
20             rs=(int)(1LL*rs*n%mod);
21         n=(int)(1LL*n*n%mod);
22         b>>=1;
23     }

```

```

24     return rs;
25 }
26
27 void fft(vector<int> & a, bool invert) {
28     int n = a.size();
29
30     for (int i = 1, j = 0; i < n; i++) {
31         int bit = n >> 1;
32         for (; j & bit; bit >>= 1)
33             j ^= bit;
34         j ^= bit;
35
36         if (i < j)
37             swap(a[i], a[j]);
38     }
39
40     for (int len = 2; len <= n; len <= 1) {
41         int wlen = invert ? root_1 : root;
42         for (int i = len; i < root_pw; i <= 1)
43             wlen = (int)(1LL * wlen * wlen % mod);
44
45         for (int i = 0; i < n; i += len) {
46             int w = 1;
47             for (int j = 0; j < len / 2; j++) {
48                 int u = a[i+j], v = (int)(1LL * a[i+j+len/2] * w % mod);
49                 a[i+j] = u + v < mod ? u + v : u + v - mod;
50                 a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
51                 w = (int)(1LL * w * wlen % mod);
52             }
53         }
54     }
55
56     if (invert) {
57         int n_1 = exp(n, mod-2);
58         for (int & x : a)
59             x = (int)(1LL * x * n_1 % mod);
60     }
61 }
62
63 void multiply(vector<int> &a, vector<int> &b) {
64     if(b.size()<=30)
65     {
66         vector<int> res(b.size()+a.size(),0);
67         for (int i = 0; i < a.size(); ++i)
68         {
69             for (int j = 0; j < b.size(); ++j)
70             {
71                 res[i+j]+=(int)(1LL*a[i]*b[j]%mod);
72                 if(res[i+j]>mod)
73                     res[i+j]-=mod;
74             }
75         }
76         a=res;
77         return ;
78     }
79     int n = 1;
80     while (n < a.size() + b.size())
81         n <= 1;
82     a.resize(n);
83     b.resize(n);
84
85     fft(a, 0);
86     fft(b, 0);
87     for (int i = 0; i < n; i++)
88         a[i] = (int)(1LL*a[i]*b[i]%mod);
89     fft(a, 1);

```

```

90 }
91
92
93
94 int main(int argc, char const *argv[])
95 {
96     ios::sync_with_stdio(false);
97     cin.tie(NULL);
98     // cout<<exp(3,952LL)<<" "<<exp(exp(3,952LL),mod-2)<<'\\n';
99
100     fat[0]=1;
101     for (int i = 1; i < N; ++i)
102     {
103         fat[i]=(int)(1LL*fat[i-1]*i%mod);
104     }
105     int q;
106     cin>>q;
107     while(q--)
108     {
109         int n;
110         cin>>n;
111         for (int i = 0; i < n; ++i)
112         {
113             a[i]=vector<int>(2);
114             int x;
115             cin>>x;
116             a[i][0]=(1);
117             a[i][1]=(x%mod);
118         }
119         for (int i = 1; i < n; i<=1)
120         {
121             for (int j = 0; j+i < n ; j+=(i<1))
122             {
123                 multiply(a[j],a[j+i]);
124             }
125         }
126         int rs=0;
127         for (int i = 1; i <= n; ++i)
128         {
129             rs += (int)(1LL*(1LL*a[0][i]*fat[i]%mod)*fat[n-i]%mod);
130             if(rs>mod)
131                 rs-=mod;
132         }
133         rs = (int)(1LL*rs*exp(fat[n],mod-2)%mod);
134         cout<<rs<<"\\n";
135     }
136     return 0;
137 }

```

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 occurrence of t in s . Solution $O(n \log n)$.

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

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5
6 #ifndef __WIN32__
7 #define getchar_unlocked getchar
8 #define putchar_unlocked putchar
9 #endif
10
11 inline void writeChar(char c) {

```

```

12     putchar_unlocked(c);
13 }
14
15 bool readChar(char &c) {
16     c = getchar_unlocked();
17     return c != EOF;
18 }
19
20 template<typename T>
21 inline void writeInt(T n){
22     register int idx = 20;
23     if( n < 0 ) putchar_unlocked('-');
24     n = abs(n);
25     char out[21];
26     out[20] = ' ';
27     do{
28         idx--;
29         out[idx] = n % 10 + '0';
30         n /= 10;
31     }while(n);
32     do{ putchar_unlocked(out[idx++]); } while (out[idx] != ' ');
33 }
34
35 struct complex_t {
36     double a {0.0}, b {0.0};
37     complex_t(){
38     }
39     complex_t(double na, double nb = 0.0) : a{na}, b{nb} {}
40     const complex_t operator+(const complex_t &c) const {
41         return complex_t(a + c.a, b + c.b);
42     }
43     const complex_t operator-(const complex_t &c) const {
44         return complex_t(a - c.a, b - c.b);
45     }
46     const complex_t operator*(const complex_t &c) const {
47         return complex_t(a * c.a - b * c.b, a * c.b + b * c.a);
48     }
49     const complex_t operator/(double r) const {
50         return complex_t(a / r, b / r);
51     }
52 };
53
54 using cd = complex_t;
55 const double PI = acos(-1);
56
57 void fft(vector<cd> & a, bool invert) {
58     int n = a.size();
59
60     for (int i = 1, j = 0; i < n; i++) {
61         int bit = n >> 1;
62         for (; j & bit; bit >>= 1)
63             j ^= bit;
64         j ^= bit;
65
66         if (i < j)
67             swap(a[i], a[j]);
68     }
69
70     for (int len = 2; len <= n; len <<= 1) {
71         double ang = 2 * PI / len * (invert ? -1 : 1);
72         cd wlen(cos(ang), sin(ang));
73         for (int i = 0; i < n; i += len) {
74             cd w(1);
75             for (int j = 0; j < len / 2; j++) {
76                 cd u = a[i+j], v = a[i+j+len/2] * w;
77                 a[i+j] = u + v;

```

```

78         a[i+j+len/2] = u - v;
79         w = w * wlen;
80     }
81 }
82 }
83
84     if (invert) {
85         for (cd & x : a)
86             x = x / n;
87     }
88 }
89
90 void multiply(vector< cd > const &a, vector< cd > const &b, vector< int
    > &result){
91     vector< cd > fa(a.begin(), a.end()), fb(b.begin(), b.end());
92     int n = 1;
93     while(n < a.size() + b.size()){
94         n <<= 1;
95     }
96
97     fa.resize(n);
98     fb.resize(n);
99
100    fft(fa, false);
101    fft(fb, false);
102
103    for(int i = 0 ; i < n ; i++){
104        fa[i] = fa[i] * fb[i];
105        // printf("%lf\n", fa[i].a);
106    }
107
108    fft(fa, true);
109
110    for(int i = 0 ; i < result.size() ; i++){
111        result[i] = floor(fa[i].a + 1e-5);
112    }
113 }
114
115 vector< int > sum;
116 const int N = int(1e5 + 10);
117 vector< cd > a, b;
118 vector< int > r;
119
120 char s[N], t[N];
121 double pre[30];
122
123 int main(){
124     int n = 0, m = 0;
125     char c;
126
127     for(int i = 0 ; i < 26 ; i++){
128         pre[i] = (2.0 * PI * i) / 26.0;
129     }
130
131     while(readChar(c)){
132         n = m = 0;
133
134         s[n++] = c;
135
136         while(readChar(c)){
137             if(c == '\n')
138                 break;
139
140             s[n++] = c;
141         }
142

```

```

143 while(readChar(c)){
144     if(c == '\n'){
145         break;
146     }
147
148     t[m++] = c;
149 }
150
151 if(m > n){
152     writeChar('0');
153     writeChar('\n');
154     continue;
155 }
156
157 a.resize(n);
158 b.resize(m);
159 r.resize(n);
160
161 sum.resize(n);
162
163 reverse(t, t + m);
164
165 for(int i = 0 ; i < n ; i++){
166     double alphai = pre[s[i] - 'a'];
167
168     a[i] = {cos(alphai), sin(alphai)};
169 }
170
171 int c = 0;
172
173 for(int i = 0 ; i < m ; i++){
174     if(t[i] == '?'){
175         c++;
176         b[i] = {0.0, 0.0};
177
178         continue;
179     }
180
181     double bi = pre[t[i] - 'a'];
182
183     b[i] = {cos(bi), -sin(bi)};
184 }
185
186 multiply(a, b, r);
187
188 int cnt = 0;
189
190 for(int i = m - 1 ; i < n ; i++){
191     // printf("%d\n", r[i]);
192
193     if(r[i] == m - c){
194         cnt++;
195         // printf("%d\n", i);
196     }
197 }
198
199 writeInt(cnt);
200 writeChar('\n');
201
202 sum.clear();
203 }
204
205 return 0;
206 }

```


2.3.4 FWHT (fft using xor/and/or)

FFT but the exponent 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.

```

1
2
3 void FWHT(vector<int>& a, bool inv) {
4     int n = a.size();
5     for (int step = 1; step < n; step *= 2) {
6         for (int i = 0; i < n; i += 2 * step)
7             rep(j,i,i+step) {
8                 int &u = a[j], &v = a[j + step];
9                 tie(u, v) = inv ? mp(v - u, u) : mp(v, u + v); // AND
10                // tie(u, v) = inv ? mp(v, u - v) : mp(u + v, u); // OR
11                /// include-line
12                // tie(u, v) = mp(u + v, u - v); // XOR /// include-line
13            }
14        // if (inv) for (int& x : a) x /= n; // XOR only /// include-line
15    }
16    vector<int> multiply(vector<int> a, vector<int> b) {
17        int n = 1;
18        while(n < a.size())
19            n<<=1;
20        a.resize(n);
21        b.resize(n);
22        FWHT(a, 0);
23        FWHT(b, 0);
24        rep(i,0,n)
25            a[i] *= b[i];
26        FWHT(a, 1);
27        return a;
28    }
29
30    pii a[N];
31    // multiply alot of polinoms of the form (x^0 + x^i) / can be done in
32    // Alog(A)^2, where A is max i
33    // probably it will be modular
34    pair<vector<int>,vector<int>> Multi_FWHT(int l, int r)
35    {
36        if(l == r)
37        {
38            vector<int> x(1,a[l].ft);
39            vector<int> y(1,a[l].sd);
40            return mp(x,y);
41        }
42        int mid = (l+r) >> 1;
43        auto [a,c] = Multi_FWHT(l,mid);
44        auto [b,d] = Multi_FWHT(mid+1,r);
45        int m = r-l+1;
46        vector<int> ab,bc,ad,cd;
47        ab = multiply(a,b);
48        bc = multiply(b,c);
49        ad = multiply(a,d);
50        cd = multiply(c,d);
51        ab.resize(m);
52        bc.resize(m);
53        rep(i,0,(m)/2){
54            ab[i+m/2] = cd[i];
55            bc[i+m/2] = ad[i];
56        }
57        return mp(ab,bc);
58    }

```

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.

```

1 struct Gauss{
2     typedef vector<double> vd;
3     typedef vector<long long int> vl;
4     typedef vector<int> vi;
5     const double eps = 1e-12;
6
7     /**
8     * Author: Per Austrin, Simon Lindholm
9     * Date: 2004-02-08
10    * License: CC0
11    * Description: Solves $A * x = b$. If there are multiple solutions,
12    an arbitrary one is returned.
13    * Returns rank, or -1 if no solutions. Data in $A$ and $b$ is lost.
14    * Time: O(n^2 m)
15    * Status: tested on kattis:equationsolver, and bruteforce-tested mod
16    3 and 5 for n,m <= 3
17    */
18    int solveLinear(vector<vd>& A, vd& b, vd& x) {
19        int n = A.size(), m = x.size(), rank = 0, br, bc;
20        if (n) assert(A[0].size() == m);
21        vi col(m); iota(col.begin(), col.end(), 0); // fills the range
22        with increasing values starting with value 0
23
24        for(int i = 0; i < n; i++){
25            double v, bv = 0;
26
27            // busca pelo maior elemento
28            for(int r = i; r < n; r++) for(int c = i; c < m; c++){
29                if ((v = fabs(A[r][c])) > bv)
30                    br = r, bc = c, bv = v;
31            }
32            // bv eh o maior elemento, se todos os elementos sao iguais
33            a zero
34            if (bv <= eps) {
35                // se no vetor de resposta tiver um cara diferente de
36                zero, nao existe x tal que 0*x != 0
37                for(int j = i; j < n; j++) if (fabs(b[j]) > eps) return
38                -1;
39                break;
40            }
41
42            swap(A[i], A[br]); // trocar a linha atual pela linha que
43            tem o maior elemento
44            swap(b[i], b[br]); // trocar a linha pra ficar igual no
45            vetor de resposta
46            swap(col[i], col[bc]); // trocar no vetor de coluna
47            for(int j = 0; j < n; j++) swap(A[j][i], A[j][bc]); //
48            trocar os elementos da coluna atual pelos elementos da coluna do
49            maior elemento

```

```

40         bv = 1/A[i][i]; // a razao necessaria pra fazer o elemento
        atual da diagonal ser 1
41         // 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
42         for(int j = i+1; j < n; j++){
43             double fac = A[j][i] * bv; // para zerar o elemento A[j]
        [i] precisamos subtrair dele A[i][i]*(1/A[i][i] * A[j][i]) = A[i][i]*
        fac
44             // 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
45             // ta fazendo aqui embaixo
46             b[j] -= fac * b[i];
47             for(int k = i+1; k < m; k++) A[j][k] -= fac*A[i][k];
48         }
49         // se eu consegui zerar essa coluna, entao a variavel dessa
        coluna eh diferente de zero (eh uma equacao linearmente independente)
50         rank++;
51     }
52
53     x.assign(m, 0);
54     for (int i = rank; i--;) {
55         // atribuo as respostas
56         b[i] /= A[i][i];
57         x[col[i]] = b[i];
58         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
59     }
60     return rank; // (multiple solutions if rank < m)
61 }
62 // tem que passar o modulo
63 template <int M> int solveLinear(vector<vl>& A, vl& b, vl& x) {
64     int n = A.size(), m = x.size(), rank = 0, br, bc;
65     if (n) assert(A[0].size() == m);
66     vi col(m); iota(col.begin(), col.end(), 0); // fills the range
        with increasing values starting with value 0
67
68     for(int i = 0; i < n; i++) for(int j = 0; j < m; j++) A[i][j] =
        (A[i][j]%M + M)%M;
69     for(int i = 0; i < n; i++) b[i] = (b[i]%M + M)%M;
70
71     for(int i = 0; i < n; i++){
72         ll v, bv = 0;
73
74         // busca pelo maior elemento
75         for(int r = i; r < n; r++) for(int c = i; c < m; c++)
76             if ((v = A[r][c]))
77                 br = r, bc = c, bv = v;
78         // bv eh o maior elemento, se todos os elementos sao iguais
        a zero
79         if (bv == 0) {
80             // se no vetor de resposta tiver um cara diferente de
        zero, nao existe x tal que 0*x != 0
81             for(int j = i; j < n; j++) if (b[j]) return -1;
82             break;
83         }
84
85         swap(A[i], A[br]); // trocar a linha atual pela linha que
        tem o maior elemento
86         swap(b[i], b[br]); // trocar a linha pra ficar igual no
        vetor de resposta
87         swap(col[i], col[bc]); // trocar no vetor de coluna
88         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
89

```

```

90         bv = inv(A[i][i],M); // a razao necessaria pra fazer o
    elemento atual da diagonal ser 1
91         // 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
92         for(int j = i+1; j < n; j++){
93             ll fac = A[j][i] * bv % M; // para zerar o elemento A[j]
    [i] precisamos subtrair dele A[i][i]*(1/A[i][i] * A[j][i]) = A[i][i]*
    fac
94             // 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
95             // ta fazendo aqui embaixo
96             b[j] = (b[j] - fac * b[i]%M + M)%M;
97             for(int k = i+1; k < m; k++) A[j][k] = (A[j][k] - fac*A[
    i][k]%M + M)%M;
98         }
99         // se eu consegui zerar essa coluna, entao a variavel dessa
    coluna eh diferente de zero (eh uma equacao linearmente independente)
100         rank++;
101     }
102
103     x.assign(m, 0);
104     for (int i = rank; i--;) {
105         // atribuo as respostas
106         b[i] = b[i] * inv(A[i][i],M)%M;
107         x[col[i]] = b[i];
108         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
109     }
110     return rank; // (multiple solutions if rank < m)
111 }
112
113 void show(vector<vl>& A, vl& b){
114     int n = A.size(), m = b.size();
115     for(int i = 0; i < n; i++){
116         for(int j = 0; j < m; j++) cout << A[i][j] << ' ';
117         cout << "= " << b[i] << '\n';
118     }
119 }
120
121 ll modpow(ll b, ll e, ll mod) {
122     ll ans = 1;
123     for (; e; b = b * b % mod, e /= 2)
124         if (e & 1) ans = ans * b % mod;
125     return ans;
126 }
127
128 ll inv(ll x, ll mod){
129     return modpow(x,mod-2,mod);
130 }
131 };

```

2.4.2 Modular 2 (bitset)

Problem: a grid of switches 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 switches pressed.

```

1 //https://www.spoj.com/problems/DFLOOR/en/
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 const int N = 300;
7
8 int gauss(vector< bitset< N > > a, int n, int m, bitset< N > &ans){

```

```

9      vector< int > where(m, -1);
10     for(int col = 0, row = 0 ; col < m && row < n ; ++col){
11         for(int i = row ; i < n ; ++i){
12             if(a[i][col]){
13                 swap(a[i], a[row]);
14                 break;
15             }
16         }
17
18         if(!a[row][col]){
19             continue;
20         }
21
22         where[col] = row;
23
24         for(int i = 0 ; i < n ; i++){
25             if(i != row && a[i][col]){
26                 a[i] ^= a[row];
27             }
28         }
29
30         ++row;
31     }
32
33     for(int i = 0 ; i < m ; i++){
34         if(where[i] != -1){
35             ans[i] = a[where[i]][m];
36         }
37     }
38
39     // cout << " ----- \n";
40     // for(auto u: a){
41     //     cout << u << "\n";
42     // }
43
44     for(int i = 0 ; i < n ; i++){
45         int sum = 0;
46         for(int j = 0 ; j < m ; j++){
47             sum += ans[j] * a[i][j];
48         }
49
50         if((sum % 2) != a[i][m]){
51             return 0;
52         }
53     }
54
55     return 1;
56 }
57
58 char tab[20][20];
59 int vi[4] = {0, 0, 1, -1};
60 int vj[4] = {1, -1, 0, 0};
61 int x, y;
62
63 bool ok(int a, int b){
64     return 0 <= a && a < x && 0 <= b && b < y;
65 }
66
67 int main(){
68     while(scanf("%d %d", &x, &y) != EOF){
69         if(x == 0 && y == 0){
70             break;
71         }
72
73         swap(x, y);
74

```

```

75     for(int i = 0 ; i < x ; i++){
76         for(int j = 0 ; j < y ; j++){
77             scanf("\n%c", &tab[i][j]);
78         }
79     }
80
81     vector< bitset< N > > q;
82
83     for(int i = 0 ; i < x ; i++){
84         for(int j = 0 ; j < y ; j++){
85             bitset< N > v;
86
87             for(int k = 0 ; k < 4 ; k++){
88                 int xi = vi[k] + i;
89                 int xj = vj[k] + j;
90
91                 if(ok(xi, xj)){
92                     v[xi * y + xj] = 1;
93                 }
94             }
95
96             v[i * y + j] = 1;
97             v[x * y] = !(tab[i][j] - '0');
98             q.push_back(v);
99             // cout << v << "\n";
100         }
101     }
102
103     bitset< N > ans;
104     if(gauss(q, q.size(), x * y, ans)){
105         vector< int > ians;
106         for(int i = 0 ; i < x * y ; i++){
107             if(ans[i]){
108                 ians.push_back(i);
109             }
110         }
111
112         printf("%lu\n", ians.size());
113
114         for(auto u: ians){
115             printf("%d %d\n", (u % y) + 1, (u / y) + 1);
116         }
117     }else{
118         printf("-1\n");
119     }
120 }
121
122 return 0;
123 }

```

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 used to combinatorial problems.

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

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int N = 200100;
5 const int M = 60;
6
7 const ll bmod = 1000'000'007LL;
8 const ll mod = 2;
9

```

```

10 ll a[N][M];
11
12 ll fexp(ll x, ll b, ll mod)
13 {
14     ll rs = 1;
15     while(b)
16     {
17         if(b&1)
18             rs = rs * x % mod;
19         x = x * x % mod;
20         b >>= 1;
21     }
22     return rs;
23 }
24 ll inv(ll x)
25 {
26     return 1;
27 }
28 int n;
29 ll compute_rank() {
30
31     int rank = 0;
32     vector<bool> row_selected(n, false);
33     for (int i = 0; i < M; ++i) {
34         int j;
35         for (j = 0; j < n; ++j) {
36             if (!row_selected[j] && a[j][i] > 0)
37                 break;
38         }
39         // cout << i << endl;
40         if (j != n) {
41             ++rank;
42             row_selected[j] = true;
43             for (int p = i + 1; p < M; ++p)
44                 a[j][p] = a[j][p] * inv(a[j][i]) % mod;
45             for (int k = 0; k < n; ++k) {
46                 if (k != j && a[k][i] > 0) {
47                     for (int p = i + 1; p < M; ++p)
48                         a[k][p] = (a[k][p] - a[j][p] * a[k][i] % mod +
mod) % mod;
49                 }
50             }
51         }
52     }
53     // error(rank);
54     return fexp(mod, n-rank, bmod);
55 }
56
57
58 int main()
59 {
60     ios::sync_with_stdio(false);
61     cin.tie(NULL);
62
63     cin >> n;
64     for(int i = 0; i < n; i++){
65         ll x;
66         cin >> x;
67         for(int j = 0; j < M; j++)
68         {
69             if(x & (1LL<<j))
70                 a[i][j] = 1;
71         }
72     }
73
74     ll at = compute_rank();

```

```

75     // error(at);
76     cout << (fexp(mod, n, bmod) - at + bmod) %bmod << '\n';
77
78     return 0;
79 }

```

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 $2^{B.size()}$.

```

1 struct Basis {
2     vector<ll> B;
3     ll reduce(ll x) {
4         for (auto b : B) x = min(x, x^b);
5         return x;
6     }
7     void insert(ll x) {
8         ll r = reduce(x);
9         if (r) B.push_back(r);
10    }
11 };

```

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 $\text{LCM}(n, m)$.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 #define ll long long
5
6 ll gcd(ll a, ll b, ll &x, ll &y) {
7     if(a == 0){
8         x = 0;
9         y = 1;
10        return b;
11    }
12    ll x0, y0;
13    ll g = gcd(b%a, a, x0, y0);
14    y = x0;
15    x = y0-x0*(b/a);
16    return g;
17 }
18
19 // Return (X, Y) where Y = lcm(n, m) and X = a mod n and X = b mod m and
20 // 0 <= X < Y
21 // X = -1 if there is not solution
22 pair<ll, ll> crt(ll a, ll n, ll b, ll m){
23     if(a < 0) a += n;
24     if(a >= n) a %= n;
25     if(b < 0) b += m;
26     if(b >= m) b %= m;
27
28     ll x1, x2;
29     ll d = gcd(n, m, x1, x2);
30     ll lcm = n * m / d;
31     if((a - b) % d != 0)
32         return {-1, lcm};
33
34     ll x = (a + (n*x1) * (__int128)((b - a)/d)) % lcm;
35     if(x < 0) x += lcm;

```



```

35     return {x, lcm};
36 }
37
38 int main(){
39     ios::sync_with_stdio(0);
40     cin.tie(0);
41
42     int T;
43     cin >> T;
44     while(T--){
45         ll n, m, a, b;
46         cin >> a >> n >> b >> m;
47
48         pair<ll, ll> ans = crt(a, n, b, m);
49         if(ans.first == -1)
50             cout << "no solution\n";
51         else cout << ans.first << " " << ans.second << "\n";
52     }
53
54
55
56     return 0;
57 }

```

2.5.2 Lucas Theorem

Given n, k and m , find $\binom{n}{k} \bmod m$, 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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define ft first
5 #define sd second
6 #define mp make_pair
7 #define mt make_tuple
8 #define eb emplace_back
9 #define pb push_back
10
11 using ll = long long;
12
13 vector<ll> fat, ifat, primes;
14
15 ll gcd(ll a, ll b, ll &x, ll &y) {
16     if(a == 0){
17         x = 0;
18         y = 1;
19         return b;
20     }
21     ll x0, y0;
22     ll g = gcd(b%a, a, x0, y0);
23     y = x0;
24     x = y0-x0*(b/a);
25     return g;
26 }
27
28 ll fexp(ll a, ll b, ll p) {
29     a %= p;
30     ll rs = 1;
31     while(b>0){
32         if(1ll&b) rs = rs*a%p;
33         a = a*a%p;
34         b>>=1ll;
35     }
36     return rs;

```

```

37 }
38
39 void _set(int p) {
40     fat.assign(p, 0);
41     ifat.assign(p, 0);
42
43     fat[0]=fat[1]=1;
44     ifat[0]=ifat[1]=1;
45     for(ll i=2; i<p; i++){
46         fat[i]=(fat[i-1]*i)%p;
47         ifat[i]=fexp(fat[i],p-2, p);
48     }
49 }
50
51 void factorize(int m) {
52     primes.clear();
53     for(int i=2; i<= 50; i++) {
54         if(m%i==0){
55             primes.pb(i);
56             while(m%i==0) m/=i;
57         }
58     }
59 }
60
61 ll Lucas(ll n, ll r, ll p) {
62     if(r<0 || r>n) return 0;
63     if(r==0 || r==n) return 1;
64
65     if(n>=p) return (Lucas(n/p,r/p,p)*Lucas(n%p,r%p,p))%p;
66     return ((fat[n]*ifat[r])%p*ifat[n-r])%p;
67 }
68
69 pair<ll,ll> crt(ll a, ll n, ll b, ll m) {
70     if(a<0) a += n;
71     if(a>=n) a %= n;
72     if(b<0) b += m;
73     if(b >= m) b %= m;
74     ll x1, x2;
75     ll d = gcd(n, m, x1, x2);
76     ll lcm = n*m/d;
77     if((a-b)%d != 0){
78         return mp(-1, lcm);
79     }
80
81     ll x = (a+(n*x1)*(__int128_t)((b-a)/d))%lcm;
82     if(x<0) x+=lcm;
83     return mp(x, lcm);
84 }
85
86 void test() {
87     ll n, r, m;
88     cin >> n >> r >> m;
89     factorize(m);
90     ll rs=0, M = 1;
91     for(int i=0; i<primes.size(); i++) {
92         _set(primes[i]);
93         tie(rs, M) = crt(rs, M, Lucas(n, r, primes[i]), primes[i]);
94     }
95     cout<<rs<<"\n";
96 }
97
98 int32_t main(){
99     cin.tie(nullptr)->sync_with_stdio(false);
100     // cout<<fixed<<setprecision(10);
101
102     int t = 1;

```

```

103     cin >> t;
104
105     for(int i=1; i<=t; i++){
106         // cout<<"Case "<<i<<" ";
107         test();
108     }
109
110     return 0;
111 }

```

2.6 Simplex

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

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int MAX_N = 505;
5  const double eps = 1e-9;
6  const int inf = 1e9;
7  const int MAX_M = 20005;
8  class Simplex
9  {
10 public:
11     int n, m;
12     double A[MAX_M][MAX_N], B[MAX_M], C[MAX_N];
13     void pivot(int l, int e)
14     {
15         B[l] /= A[l][e];
16
17         for (int i = 1; i <= n; ++i)
18             if (i != e)
19                 A[l][i] /= A[l][e];
20
21         A[l][e] = 1 / A[l][e];
22
23         for (int i = 1; i <= m; ++i)
24         {
25             if (i != l && fabs(A[i][e]) > 0)
26             {
27                 B[i] -= B[l] * A[i][e];
28                 for(int j = 1; j <= n; ++j)
29                     if (j != e)
30                         A[i][j] -= A[l][j] * A[i][e];
31                 A[i][e] = -A[i][e] * A[l][e];
32             }
33         }
34         for (int i = 1; i <= n; ++i)
35             if (i != e)
36                 C[i] -= C[e] * A[l][i];
37         C[e] = -C[e] * A[l][e];
38     }
39     double simplex()
40     {
41         double res = 0;
42         while (true)
43         {
44             double tmp = 0;
45             int e = 0, l = 0;
46             for (int i = 1; i <= n; ++i)
47                 if (C[i] > tmp)
48                     e = i, tmp = C[i];
49
50             if (!e)
51                 return res;

```

```

52         tmp = inf;
53         for (int i = 1; i <= m; ++i)
54         {
55             if(A[i][e] > 0 && tmp > B[i] / A[i][e])
56             {
57                 tmp = B[i] / A[i][e];
58                 l = i;
59             }
60         }
61         if(tmp == inf)
62             return tmp;
63         else
64             res += tmp * C[e], pivot(l, e);
65     }
66 }
67 } solve;
68
69 int x[MAX_N];
70 int y[MAX_N];
71
72 const double pi = acos(-1);
73
74 double sq(double a)
75 {
76     return a*a;
77 }
78
79 double dist(int i, int j)
80 {
81     return sqrt(sq(x[i] - x[j]) + sq(y[i] - y[j]));
82 }
83
84 int main()
85 {
86     int n;
87
88     scanf("%d",&n);
89
90     for(int i = 1; i <= n; i++)
91     {
92         scanf("%d %d",&x[i], &y[i]);
93     }
94     int idx = 1;
95     solve.n = n;
96     for(int i = 1; i <= n; i++)
97     {
98         for(int j = i+1; j <= n; j++)
99         {
100             solve.A[idx][i] = 1;
101             solve.A[idx][j] = 1;
102             solve.B[idx] = dist(i,j);
103             idx++;
104         }
105         solve.C[i] = 1;
106     }
107     solve.m = idx;
108
109     printf("%.10lf\n",2*pi*solve.simplex());
110     return 0;
111 }

```

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 $Max(gcd(A[l], \dots, A[r]) * (r - l + 1))$.

```

1 //UVALive-6582
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int N=100010;
6 typedef long long ll;
7 const ll mod=998244353LL;
8
9 map<ll, ll> sub_gcd[N];
10 ll a[N];
11 int main(){
12     ios::sync_with_stdio(false);
13     cin.tie(NULL);
14     int q;
15     cin>> q;
16     while(q--){
17         {
18             int n;
19             cin>>n;
20             for (int i = 0; i < n; ++i)
21             {
22                 cin>>a[i];
23                 sub_gcd[i].clear();
24             }
25             sub_gcd[0][a[0]] = 1;
26             ll ans=0;
27             for(int i = 1; i < n; i++){
28                 {
29                     sub_gcd[i][a[i]] = 1;
30                     for(auto it: sub_gcd[i - 1])
31                     {
32                         ll new_gcd = __gcd(it.first, a[i]);
33                         sub_gcd[i][new_gcd] = max(sub_gcd[i][new_gcd], it.second
+ 1);
34                     }
35                 }
36                 for (int i = 0; i < n; ++i)
37                     for(auto it: sub_gcd[i])
38                         ans=max(ans,it.first*it.second);
39                 cout<<ans<<'\n';
40             }
41             return 0;
42 }

```

2.8 Implementation of Polynomials

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

```

1 typedef complex<double> cdouble;
2 int cmp(cdouble x, cdouble y = 0) {
3     return cmp(abs(x), abs(y));
4 }
5 const int TAM = 200;
6 struct poly {
7     cdouble poly[TAM]; int n;
8     poly(int n = 0): n(n) { memset(p, 0, sizeof(p)); }
9     cdouble& operator [](int i) { return p[i]; }
10    poly operator ~() {
11        poly r(n-1);
12        for (int i = 1; i <= n; i++)
13            r[i-1] = p[i] * cdouble(i);
14        return r;
15    }
16    //divide poly by binom

```

```

17 pair<poly, cdouble> ruffini(cdouble z) {
18     if (n == 0) return make_pair(poly(), 0);
19     poly r(n-1);
20     for (int i = n; i > 0; i--) r[i-1] = r[i] * z + p[i];
21     return make_pair(r, r[0] * z + p[0]);
22 }
23 cdouble operator()(cdouble z) { return ruffini(z).second; }
24
25 cdouble find_one_root(cdouble x) {
26     poly p0 = *this, p1 = ~p0, p2 = ~p1;
27     int m = 1000;
28     while (m--) {
29         cdouble y0 = p0(x);
30         if (cmp(y0) == 0) break;
31         cdouble G = p1(x) / y0;
32         cdouble H = G * G - p2(x) - y0;
33         cdouble R = sqrt(cdouble(n-1) * (H * cdouble(n) - G * G));
34         cdouble D1 = G + R, D2 = G - R;
35         cdouble a = cdouble(n) / (cmp(D1, D2) > 0 ? D1 : D2);
36         x -= a;
37         if (cmp(a) == 0) break;
38     }
39     return x;
40 }
41 vector<cdouble> roots() {
42     poly q = *this;
43     vector<cdouble> r;
44     while (q.n > 1) {
45         cdouble z(rand() / double(RAND_MAX), rand() / double(
RAND_MAX));
46         z = q.find_one_root(z); z = find_one_root(z);
47         q = q.ruffini(z).first;
48         r.push_back(z);
49     }
50     return r;
51 }
52 };

```

2.9 Combinatorial

Multiple combinatorial solutions.

Number of ways to take K objects, between $N \rightarrow toma(N, K)$

Number of ways to put N objects in K boxes (stars and bars) $\rightarrow toma(N + K - 1, K - 1)$

Count how to label $N + K$ pairs of parentheses with K '()'s already fixed $\rightarrow catalan(N, K)$

```

1 struct Comb{
2     vector<ll> fato,fatoinv;
3     vector<vector<ll>> tomatoma;
4
5     // setar so os fatorias e inversos
6     Comb(int n){
7         set_tam(n);
8     }
9
10    // setar so o triangulo de pascal
11    Comb(int n,int m){
12        set_tomatoma(n,m);
13    }
14
15    // setar os fatorias e triangulo de pascal
16    Comb(int n2,int n,int m){
17        set_tam(n2);
18        set_tomatoma(n,m);
19    }
20

```

```

21 Comb(){}
22
23 void set_tam(int n){
24     fato.resize(n+10,-1);
25     fatoinv.resize(n+10,-1);
26 }
27
28 void set_tomatoma(int n,int m){
29     tomatoma.resize(n+10,vector<ll>(m+10));
30     for(int i=0; i<=m; i++) tomatoma[0][i]=0;
31     for(int i=0; i<=n; i++) tomatoma[i][0]=1%mod;
32     for(int i=1; i<=n; i++) for(int j=1; j<=m; j++) tomatoma[i][j] =
33 (tomatoma[i-1][j-1] + tomatoma[i-1][j])%mod;
34 }
35
36 ll inv(ll a){
37     return fexp(a,mod-2);
38 }
39
40 ll fexp(ll a, ll b){
41     ll ans=1;
42     while(b){
43         if(b&1) ans=ans*a%mod;
44         b>>=1;
45         a=a*a%mod;
46     }
47     return ans;
48 }
49
50 ll fat(ll x){
51     if(ll(fato.size()) <= x){
52         fato.resize(x+10,-1);
53     }
54     if(x <= 1) return 1;
55     if(fato[x] != -1) return fato[x];
56     return fato[x]=fat(x-1)*x%mod;
57 }
58
59 ll fatinv(ll x){
60     if(ll(fatoinv.size()) <= x){
61         fatoinv.resize(x+10,-1);
62     }
63     if(x <= 1) return 1;
64     if(fatoinv[x] != -1) return fatoinv[x];
65     return fatoinv[x]=fexp(fat(x),mod-2);
66 }
67
68 ll toma(ll n, ll k){
69     if(n < 0) return 0;
70     if(k > n) return 0;
71     ll ans;
72     if(tomatoma.size() > n && tomatoma[0].size() > k){
73         ans = tomatoma[n][k];
74     }else{
75         ans=fat(n);
76         ans=ans*fatinv(k)%mod;
77         ans=ans*fatinv(n-k)%mod;
78     }
79     return ans;
80 }
81
82 //Count how to label N + K pairs of parentheses with K left ones
83 //already fixed
84 //(k=0 means normal catalan)
85 ll catalan(ll n, ll k){
86     ll ans = (k+1) * toma(2*n + k, n) % mod;

```

```
85         ans = ans * inv(n+k+1) % mod;
86         return ans;
87     }
88 };
```

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 receives their own hat.

Principle of Inclusion-Exclusion:

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

We subtract from $n!$ the number of ways for each event to occur; that is, consider the quantity $n! - |E_1| - |E_2| - \dots - |E_n|$. 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:

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

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 you 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 a 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 distinct values (we take $0, 1, 2, \dots, K$ 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)

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

```
1  /*
2      Alteracoes:
3
4      Alterar a funcao g de dentro da funcao de lagrange
5      Na funcao que faz o lagrange mesmo, alterar os parametros da funcao
6      g de acordo com o que faz
7  */
8  struct Lagrange{
9      vector<ll> pref,suf;
10     vector<ll> invfat;
11     vector<ll> y;
12     // valor x que quer calcular e o grau do polinomio n
13     ll x,n;
14     ll mod;
15     // geralmente o negocio da funcao de dentro precisa de mais
16     informacao
17     ll pte;
```



```

17 void set__(){
18     pref[0]=1;
19     suf[n+2]=1;
20     for(ll i=1; i<=n+1; i++) pref[i]=pref[i-1]*((x-i)%mod)%mod;
21     for(ll i=n+1; i>0; i--) suf[i]=suf[i+1]*((x-i)%mod)%mod;
22     ll fat=1;
23     invfat[0]=1;
24     for(ll i=1; i<=n+1; i++){
25         fat=fat*i%mod;
26         invfat[i]=fexp(fat,mod-2);
27     }
28     y[0]=0;
29     for(int i=1; i<=n+1; i++) y[i]=(y[i-1]+g())%mod;
30 }
31
32 void set2__(){
33     pref[0]=1;
34     suf[n+2]=1;
35     for(ll i=1; i<=n+1; i++) pref[i]=pref[i-1]*((x-i)%mod)%mod;
36     for(ll i=n+1; i>0; i--) suf[i]=suf[i+1]*((x-i)%mod)%mod;
37 }
38
39 Lagrange(ll x, ll n, ll mod, ll pte){
40     pref.resize(n+3);
41     suf.resize(n+3);
42     invfat.resize(n+3);
43     y.resize(n+3);
44
45     this->x=x;
46     this->n=n;
47     this->mod=mod;
48     this->pte=pte;
49
50     set__();
51 }
52
53 // exponenciacao binaria
54 ll fexp(ll a, ll b){
55     a%=mod;
56     ll ans=1;
57     while(b){
58         if(b&1) ans=ans*a%mod;
59         b>>=1;
60         a=a*a%mod;
61     }
62     return ans;
63 }
64
65 // se eu so quiser mudar o x que eu calculo do polinomio, so preciso
66 // alterar o prefixo e o sufixo
67 void mudax(ll x){
68     this->x = x;
69
70     set2__();
71 }
72
73 // funcao de dentro do polinomio de lagrange
74 ll g(){
75 }
76
77 // o solve, calcula o valor do polinomio no ponto x, com grau n (lg
78 // - lagrange)
79 ll lg(){
80     if(x <= n+1){
81         ll ans=0;

```

```

81         // calcula o somatorio ate o n normal
82         for(int i=1; i<=x; i++) ans=(ans+g())%mod;
83         return ans;
84     }else{
85         ll ans=0;
86         for(int i=1; i<=n+1; i++){
87             // g(i)*produtorio(x-j)/produtorio(i-j)
88             ll aux=y[i]*pref[i-1]%mod;
89             aux=aux*suf[i+1]%mod;
90             aux=aux*invfat[n+1-i]%mod;
91             aux=aux*invfat[i-1]%mod;
92
93             // o -1 do produtorio(i-j)
94             if((n+1-i)&1) aux*=-1;
95             while(aux < 0) aux+=mod;
96
97             // adiciona esse cara no polinomio de lagrange
98             ans=(ans+aux)%mod;
99         }
100
101         return ans;
102     }
103 }
104 };

```

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, \dots, 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 $0 \leq r < p$, and with that you can also write $x = (q+1)*p - r$, where $0 \leq r < p$):

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

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

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

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

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 $0 \leq p \leq \sqrt{m} + 1$ (the plus 1 comes from the minus from the r) and $0 \leq r < \sqrt{m}$, so you can precalculate the values of a^{q*p} for all $0 \leq p \leq \sqrt{m}$ and $b * a^r$ for all $0 \leq r < \sqrt{m}$, 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 $\gcd(a, m) \neq 1$, 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 $\gcd(a, m) \neq 1$. But after every time you do this operation you eliminate one a from a^x .

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

$$a^{x-1} * a \equiv b \pmod{m}$$

$$a^{x-1} * \frac{a}{g} \equiv \frac{b}{g} \pmod{\frac{m}{g}}$$

...

$$a^{x-add} * \frac{a^{add}}{k} \equiv \frac{b}{k} \pmod{\frac{m}{k}}$$

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

```

1 // a^x === b (mod m)
2 ll lgd(ll a, ll b, ll m){
3     a%=m;
4     b%=m;
5     if(a == 0 && b != 0) return -1;
6     ll k=1,add=0,g;
7     while((g = __gcd(a,m)) > 1LL){
8         if(b == k) return add;
9         if(b%g) return -1;
10        b/=g;
11        m/=g;
12        add++;
13        k = k*(a/g)%m;
14    }
15    ll n = sqrt(m)+1;
16
17    ll fat=1;
18    for(int i=0; i<n; i++) fat=fat*a%m;
19
20    vector<pair<ll,ll>> big;
21    ll aux=1;
22    for(int i=0; i<n+10; i++){
23        aux=aux*fat%m;
24        big.push_back({aux*k%m,i+1});
25    }
26    // os caras que sao k*a^(p*n)
27    sort(big.begin(),big.end(),[&](pair<ll,ll> x, pair<ll,ll> y){
28        return x.first < y.first;
29    });
30
31    aux=1;
32    vector<pair<ll,ll>> small;
33    for(int i=0; i<=n; i++){
34        small.push_back({b*aux%m,i});
35        aux=aux*a%m;
36    }
37    // os caras que sao b*a^q
38    sort(small.begin(),small.end(), [&] (pair<ll,ll> x, pair<ll,ll> y){
39        return x.first < y.first;
40    });
41
42    // se ele quiser que a resposta tenha alguma propriedade
43    ll ans=-1;
44    int i=0,j=0;
45    while(i < big.size() && j < small.size()){
46        if(big[i].first == small[j].first) {
47            if(ans == -1) ans=big[i].second*n-small[j].second+add;
48            else ans=big[i].second*n-small[j].second+add;
49            i++,j++;
50        }else if(big[i].first < small[j].first) i++;
51        else j++;
52    }
53    return ans;
54 }

```

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} .

$$\begin{aligned} x_1, x_{K+1} &\geq 0 \\ x_i &> 0 \quad \forall 2 \leq i \leq K \end{aligned}$$

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:

$$x'_1 - 1 + x_2 + x_3 + \dots + x_{K-1} + x_K + x'_{K+1} - 1 = N - K$$

$$x'_1 + x_2 + x_3 + \dots + x_{K-1} + x_K + x'_{K+1} = N - K + 2$$

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 .

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

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 idea of multiplicative functions is to use the linear sieve or just the sieve to calculate them. Normally it will lead to some principle and you 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 are not coprime,

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

```
1 std::vector<int> prime;
2 bool is_composite[MAXN];
3 int phi[MAXN];
4
5 void sieve (int n) {
6     std::fill (is_composite, is_composite + n, false);
7     phi[1] = 1;
8     for (int i = 2; i < n; ++i) {
9         if (!is_composite[i]) {
10             prime.push_back (i);
11             phi[i] = i - 1;           //i is prime
12         }
13         for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
14             is_composite[i * prime[j]] = true;
15             if (i % prime[j] == 0) {
16                 phi[i * prime[j]] = phi[i] * prime[j]; //prime[j] divides i
17                 break;
18             } else {
19                 phi[i * prime[j]] = phi[i] * phi[prime[j]]; //prime[j] does not
20                 divide i
21             }
22         }
23     }
```

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

```

1 vector <int> prime;
2 bool is_composite[MAXN];
3 int func[MAXN], cnt[MAXN];
4
5 void sieve (int n) {
6     fill (is_composite, is_composite + n, false);
7     func[1] = 1;
8     for (int i = 2; i < n; ++i) {
9         if (!is_composite[i]) {
10             prime.push_back (i);
11             func[i] = 1; cnt[i] = 1;
12         }
13         for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
14             is_composite[i * prime[j]] = true;
15             if (i % prime[j] == 0) {
16                 func[i * prime[j]] = func[i] / cnt[i] * (cnt[i] + 1);
17                 cnt[i * prime[j]] = cnt[i] + 1;
18                 break;
19             } else {
20                 func[i * prime[j]] = func[i] * func[prime[j]];
21                 cnt[i * prime[j]] = 1;
22             }
23         }
24     }
25 }

```

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:

$$\begin{aligned}
 & \sum_{i=1}^n \sum_{j=1}^n [gcd(i, j) = 1] \\
 & \sum_{i=1}^n \sum_{j=1}^n \sum_{d|gcd(i, j)} \mu(d) \\
 & \sum_{i=1}^n \sum_{j=1}^n \sum_{d=1}^n [d|gcd(i, j)] * \mu(d) \\
 & \sum_{i=1}^n \sum_{j=1}^n \sum_{d=1}^n [d|i] * [d|j] * \mu(d) \\
 & \sum_{d=1}^n \mu(d) * \left(\sum_{i=1}^n [d|i] \right) * \left(\sum_{j=1}^n [d|j] \right) \\
 & \sum_{d=1}^n \mu(d) * \left(\left\lfloor \frac{n}{d} \right\rfloor \right) * \left(\left\lfloor \frac{n}{d} \right\rfloor \right)
 \end{aligned}$$

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:

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

Next:

$$\begin{aligned}
& \sum_{i=1}^n \sum_{j=1}^n \text{lcm}(i, j) \\
& \sum_{i=1}^n \sum_{j=1}^n \frac{i * j}{\gcd(i, j)} \\
& \sum_{k=1}^n \sum_{i=1}^n \sum_{j=1}^n [\gcd(i, j) = k] * \frac{i * j}{k} \\
& \sum_{k=1}^n k * \sum_{a=1}^{\lfloor \frac{n}{k} \rfloor} \sum_{b=1}^{\lfloor \frac{n}{k} \rfloor} [\gcd(a, b) = 1] * a * b \\
& \sum_{k=1}^n k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * \left(\sum_{a=1}^{\lfloor \frac{n}{k} \rfloor} [d|a] * a \right) * \left(\sum_{b=1}^{\lfloor \frac{n}{k} \rfloor} [d|b] * b \right) \\
& a = d * p, b = d * q \\
& \sum_{k=1}^n k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * \left(\sum_{p=1}^{\lfloor \frac{n}{k*d} \rfloor} d * p \right) * \left(\sum_{q=1}^{\lfloor \frac{n}{k*d} \rfloor} d * q \right) \\
& \sum_{k=1}^n k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * \left(\sum_{p=1}^{\lfloor \frac{n}{k*d} \rfloor} p \right) * \left(\sum_{q=1}^{\lfloor \frac{n}{k*d} \rfloor} q \right) \\
& \sum_{k=1}^n k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * \left(\sum_{p=1}^{\lfloor \frac{n}{k*d} \rfloor} p \right) * \left(\sum_{q=1}^{\lfloor \frac{n}{k*d} \rfloor} q \right)
\end{aligned}$$

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 $l = k * d$

$$\begin{aligned}
& \sum_{k=1}^n k * \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) * d^2 * (PA(1, \left\lfloor \frac{n}{l} \right\rfloor)^2) \\
& \sum_{l=1}^n (PA(1, \left\lfloor \frac{n}{l} \right\rfloor)^2) * \sum_{d'|l} \mu(d') * d' * l
\end{aligned}$$

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

$$\begin{aligned}
& \sum_{i=1}^n \sum_{j=1}^n lcm(A[i], A[j]) \\
& \sum_{d=1}^M \sum_{(a \in A \ \& \ d|a)} \sum_{(b \in A \ \& \ d|b)} [gcd(\frac{a}{d}, \frac{b}{d}) = 1] * \frac{a * b}{d} \\
& \sum_{d=1}^M \sum_{(a \in A \ \& \ d|a)} \sum_{(b \in A \ \& \ d|b)} \sum_{l=1}^M \mu(l) * [l|\frac{a}{d}][l|\frac{b}{d}] * \frac{a * b}{d} \\
& \sum_{d=1}^M \frac{1}{d} * \sum_{l=1}^M \mu(l) * (\sum_{a \in A} \sum_{b \in A} [dl|a][dl|b] * a * b) \\
& \text{let } t = dl \\
& \sum_{t=1}^M \frac{1}{d} * \sum_{l|t} \frac{l}{t} * \mu(l) * (\sum_{a \in A} \sum_{b \in A} [t|a][t|b] * a * b) \\
& \sum_{t=1}^M (\sum_{l|t} \frac{l}{t} * \mu(l)) * (\sum_{a \in A \ \& \ t|a} a)^2
\end{aligned}$$

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).

```
1 #include <ext/pb_ds/assoc_container.hpp> // Common file
2 #include <ext/pb_ds/tree_policy.hpp> // Including
   tree_order_statistics_node_update
3 using namespace __gnu_pbds;
4
5 template<class T> using ordset = tree<
6 T,
7 null_type,
8 less<T>,
9 rb_tree_tag,
10 tree_order_statistics_node_update>;
11
12 ordset<int> os;
```

3.2 Fenwick Tree

3.2.1 Simples

```
1 /*
2     Indexado em 1, n esqueca de especificar se a BIT for de sufixo
3
4     Alteracoes:
5
6     Adicionar o valor do off (valor dummy)
7     Atualizar a funcao da BIT f
8 */
9 template <class TT = ll>
10 struct Fen{
11     vector<TT> fen;
12     int n; // tamanho da BIT
13     int pref; // flag que indica se a Fenwick eh no prefixo(1), ou no
   sufixo(0)
14     const TT off_fen = ; // valor dummy
15
16     Fen(int n_, int pref_=1) : pref(pref_),n(n_){
17         fen.resize(n+10,off_fen);
18     }
19     ~Fen(){fen.clear();}
20
21     // operacao da BIT
22     TT f(TT x, TT y){
23         return
24     }
```

```

25
26 void update(int x, TT v){
27     if(x <= 0 || x > n) return;
28     if(pref){
29         while(x <= n){
30             fen[x]=f(fen[x],v);
31             x+=(x&-x);
32         }
33     }else{
34         while(x){
35             fen[x]=f(fen[x],v);
36             x-=(x&-x);
37         }
38     }
39 }
40
41 TT query(int x){
42     if(x <= 0 || x > n) return off_fen;
43     TT ans = off_fen;
44     if(pref){
45         while(x){
46             ans=f(ans,fen[x]);
47             x-=(x&-x);
48         }
49     }else{
50         while(x <= n){
51             ans=f(ans,fen[x]);
52             x+=(x&-x);
53         }
54     }
55     return ans;
56 }
57 // TT query(int l, int r){
58 //     return query(r) OPERACAO query(l);
59 // }
60 };

```

3.2.2 2D

You are given an $n \times n$ 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.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5 typedef pair<int,int> pii;
6
7 const int N = 1010;
8
9 int bit[N][N], a[N][N];
10
11 void updt(int x,int y,int v)
12 {
13     if(!v)
14         return;
15     for(int i = x; i < N; i+=(i&-i))
16     {
17         for(int j = y; j < N; j+=(j&-j))
18         {
19             bit[i][j] += v;
20         }
21     }
22 }

```

```

23 int query(int x,int y)
24 {
25     int v = 0;
26     if(!x || !y)
27         return 0;
28     for(int i = x; i ; i-=(i&-i))
29     {
30         for(int j = y; j; j-=(j&-j))
31         {
32             v += bit[i][j];
33         }
34     }
35     return v;
36 }
37
38
39 int main()
40 {
41     ios::sync_with_stdio(false);
42     cin.tie(NULL);
43     int n,q;
44     string s;
45     cin >> n >> q;
46     for(int i = 1; i <= n; i++){
47         cin >> s;
48         for(int j = 1; j <= n; j++)
49         {
50             if(s[j-1] == '*'){
51                 a[i][j] = 1;
52                 updt(i, j, 1);
53             }
54         }
55     }
56
57     for(int i = 0; i < q; i++)
58     {
59         int t;
60         cin >> t;
61         if(t == 1)
62         {
63             int x, y;
64             cin >> x >> y;
65             updt(x, y, -a[x][y]);
66             a[x][y] ^= 1;
67             updt(x, y, a[x][y]);
68         }else
69         {
70             int x1, x2, y1, y2;
71             cin >> x1 >> y1 >> x2 >> y2;
72             cout << query(x2, y2) - query(x2, y1-1) - query(x1-1, y2) +
73             query(x1-1, y1-1) << '\n';
74         }
75     }
76     return 0;
77 }

```

3.2.3 Fenwick Lifting

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

```

1 // This is equivalent to calculating lower_bound on prefix sums array
2 // LOGN = log(N)
3
4 int bit[N]; // BIT array
5
6 int bit_search(int v)

```

```

7 {
8     int sum = 0;
9     int pos = 0;
10
11     for(int i=LOGN; i>=0; i--)
12     {
13         if(pos + (1 << i) < N and sum + bit[pos + (1 << i)] < v)
14         {
15             sum += bit[pos + (1 << i)];
16             pos += (1 << i);
17         }
18     }
19
20     return pos + 1; // +1 because 'pos' will have position of largest
21                     // value less than 'v'
22 }

```

3.3 Sparse Table

3.3.1 Simple Sparse Table

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

```

1  /*
2      Indexado de 1.
3      Responde uma operacao num subarray, a operacao precisa ser
4      associativa e de preferencia ter uma identidade.
5      Init:  $O(N \cdot \log(N))$ .
6      Query:  $O(\log(N))$  ou  $O(1)$  para operacoes idempotentes.
7
8      Alteracoes:
9      Funcao f() da tabela, valor off
10 */
11 template <class TT = ll>
12 struct SparseTable{
13     int n; // tamanho
14     vector<vector<TT>> tab; // sparse table
15     vector<int> pot2; // log de cada valor
16     int TETO; // Teto do log2(n)
17     const TT off = ;
18
19     SparseTable(int n_) : n(n_){
20         pot2.resize(n+10);
21         pot2[1] = 0;
22         for(int i=2; i<=n; i++) pot2[i] = pot2[i>>1] + 1;
23         tab.resize(n+10, vector<TT>(pot2[n]+1));
24         TETO = pot2[n]+1;
25     }
26
27     SparseTable(int n_, vector<TT> & a) : n(n_){
28         pot2.resize(n+10);
29         pot2[1] = 0;
30         for(int i=2; i<=n; i++) pot2[i] = pot2[i>>1] + 1;
31         tab.resize(n+10, vector<TT>(pot2[n]+1));
32         TETO = pot2[n]+1;
33         build(a); // build da Sparse Table
34     }
35
36     SparseTable(int n_, TT * a) : n(n_){
37         pot2.resize(n+10);
38         pot2[1] = 0;
39         for(int i=2; i<=n; i++) pot2[i] = pot2[i>>1] + 1;
40         tab.resize(n+10, vector<TT>(pot2[n]+1));
41         TETO = pot2[n]+1;
42         build(a); // build da Sparse Table
43     }
44
45     ~SparseTable(){tab.clear(); pot2.clear();}
46 }

```

```

43
44 void init(int n_new, vector<TT> & a){
45     n = n_new;
46     TETO = pot[n] + 1;
47     build(a);
48 }
49 void init(int n_new, TT * a){
50     n = n_new;
51     TETO = pot[n] + 1;
52     build(a);
53 }
54
55 void build(vector<TT> & a){
56     for(int i=1; i<=n; i++) tab[i][0] = a[i];
57     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]);
58 }
59 void build(TT * a){
60     for(int i=1; i<=n; i++) tab[i][0] = a[i];
61     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]);
62 }
63
64 // operacao da sparse table
65 TT f(TT x, TT y){
66     return ;
67 }
68
69 TT query(int l, int r){
70     if(l > r) return off;
71     TT ans = off;
72     for(int j=TETO-1; j>=0; j--){
73         if(l+(1<<j)-1 <= r){
74             ans = f(ans, tab[l][j]);
75             l +=(1<<j);
76         }
77     }
78     return ans;
79 }
80 // query idempotente
81 TT query_id(int l, int r){
82     if(l > r) return off;
83     int diff = pot2[r-l+1];
84     return f(tab[l][diff], tab[r-(1<<diff)+1][diff]);
85 }
86 };

```

3.3.2 Disjoint Sparse Table

All queries in $O(1)$.

```

1  /*
2     Indexado de 1.
3     Responde uma operacao num subarray, a operacao precisa ser
4     associativa e de preferencia ter uma identidade.
5     Init:  $O(N \log(N))$ .
6     Query:  $O(1)$ 
7
8     Alteracoes:
9     Funcao f(), valor off
10 */
11 template <class TT = ll>
12 struct DST{
13     int n; // tamanho do vetor original
14     vector<vector<TT>> tab; // disjoint sparse table
15     const TT off = ; // identidade

```

```

15  int TETO_DST; // menor potencia tal que 2^TETO_DST >= n
16  vector<int> pot2; // log2 de cada numero
17
18  DST(int n_) : n(n_){
19      TETO_DST = 0;
20      int lim = 1;
21      while(lim < n) lim <= 1, TETO_DST++;
22      pot2.resize(lim);
23      pot2[1] = 0;
24      for(int i=2; i<lim; i++) pot2[i] = pot2[i>>1] + 1;
25      tab.resize(TETO_DST+1,vector<TT>(lim,off));
26  }
27  DST(int n_, vector<TT> & a) : n(n_){
28      TETO_DST = 0;
29      int lim = 1;
30      while(lim < n) lim <= 1, TETO_DST++;
31      pot2.resize(lim);
32      pot2[1] = 0;
33      for(int i=2; i<lim; i++) pot2[i] = pot2[i>>1] + 1;
34      tab.resize(TETO_DST+1,vector<TT>(lim,off));
35      init(a);
36  }
37
38  void init(vector<TT> & a){
39      int lim = 1<<TETO_DST;
40      // qtd de blocos
41      for(int i=0; i<TETO_DST; i++){
42          int tam = (1<<(TETO_DST-i)); // tamanho do bloco
43          // bloco atual
44          for(int j=0; j<(1<<i); j++){
45              int init = j*tam;
46              int end = (j+1)*tam-1;
47              int mid = (init + end)>>1;
48
49              // primeira metade eh um sufixo
50              tab[i][mid] = (mid+1 <= n ? a[mid+1] : off); // vetor de
51              entrada eh indexado de 1
52              for(int k=mid-1; k>=init; k--) tab[i][k] = f((k+1 <= n ?
53              a[k+1] : off),tab[i][k+1]);
54
55              // segunda metade eh um prefixo
56              tab[i][mid+1] = (mid+2 <= n ? a[mid+2] : off); // vetor
57              de entrada eh indexado de 1
58              for(int k=mid+2; k<=end; k++) tab[i][k] = f((k+1 <= n ?
59              a[k+1] : off),tab[i][k-1]);
60          }
61      }
62      for(int j=0; j<lim; j++) tab[TETO_DST][j] = (j+1 <= n ? a[j+1] :
63      off);
64  }
65
66  // operacao da DST
67  TT f(TT x, TT y){
68      return ;
69  }
70
71  // query inclusiva nos ranges
72  TT query(int l,int r){
73      if(l > r) return off;
74      l--, r--;
75      if(l == r) return tab[TETO_DST][l];
76      int level = TETO_DST - pot2[l^r] - 1;
77      return f(tab[level][l],tab[level][r]);
78  }
79  };

```

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.

```
1  /*
2      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
3
4      Alteracoes:
5
6      Ver como calcula a query
7      Ler as queries
8      Ver como imprime a resposta
9      Ver as funcoes de add e rmv do mo
10 */
11 template <class TT = int>
12 struct Mo{
13     const int K = 450;
14
15     struct query{
16         int idx,l,r;
17
18         bool operator <(const query &o){
19             if(1/K == o.1/K){
20                 if((1/K)&1) return r>o.r;
21                 return r<o.r;
22             }
23             return 1/K < o.1/K;
24         }
25     };
26
27     vector<query> q;
28     vector<TT> ans;
29
30     // qtd de queries
31     Mo(int n_q){
32         q.resize(n_q);
33     }
34
35
36     // adiciona na posicao x
37     void add(int x){
38
39     }
40
41     // remove na posicao x
42     void rmv(int x){
43
44     }
45
46     // calcula a resposta pra query
47     TT calc(){
48
49     }
50
51     void solve(){
52         sort(q.begin(),q.end());
53         ans.resize(q.size());
54         int i,j;
55         i=1;
56         j=0;
57         for(auto [idx,l,r] : q){
58             while(j < r){
59                 add(++j);
60             }
```

```

61     while(i > l){
62         add(--i);
63     }
64
65     while(j > r){
66         rmv(j--);
67     }
68     while(i < l){
69         rmv(i++);
70     }
71
72     ans[idx]=calc();
73 }
74 }
75
76 // ler as queries
77 void ler(){
78     for(int i=0; i<q.size(); i++) {
79         cin >> q[i].l >> q[i].r;
80         q[i].idx=i;
81     }
82 }
83
84 void show(){
85     // imprime a resposta
86 }
87 };

```

3.4.2 MO com updates

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

```

1  const int N = 100005
2  int n, q;
3  int v[N];
4  int vv[N];
5
6  namespace mo
7  {
8      struct query
9      {
10         int idx, l, r, t;
11     };
12     struct update
13     {
14         int i, prevx, x;
15     };
16
17     int block;
18     vector<query> queries;
19     vector<update> updates;
20     vector<int> ans;
21
22     bool cmp(query x, query y)
23     {
24         if (x.l / block != y.l / block)
25             return x.l / block < y.l / block;
26         if (x.r / block != y.r / block)
27             return x.r / block < y.r / block;
28         return x.t < y.t;
29     }
30     void run()
31     {
32         block = 3153; // (2 * n) ^ 0.666
33         sort(queries.begin(), queries.end(), cmp);

```



```

34     ans.resize(queries.size());
35     int cl = 0, cr = -1, sum = 0, t = 0;
36     auto add = [&](int x)
37     {
38         sum += x;
39     };
40     auto rem = [&](int x)
41     {
42         sum -= x;
43     };
44     for (int i = 0; i < queries.size(); i++)
45     {
46         while (cl > queries[i].l)
47         {
48             cl--;
49             add(v[cl]);
50         }
51         while (cr < queries[i].r)
52         {
53             cr++;
54             add(v[cr]);
55         }
56         while (cl < queries[i].l)
57         {
58             rem(v[cl]);
59             cl++;
60         }
61         while (cr > queries[i].r)
62         {
63             rem(v[cr]);
64             cr--;
65         }
66         while (t > queries[i].t)
67         {
68             t--;
69             if (queries[i].l <= updates[t].i && queries[i].r >= updates[t].i
70         )
71             {
72                 rem(updates[t].x);
73                 add(updates[t].prevx);
74             }
75             v[updates[t].i] = updates[t].prevx;
76         }
77         while (t < queries[i].t)
78         {
79             if (queries[i].l <= updates[t].i && queries[i].r >= updates[t].i
80         )
81             {
82                 rem(updates[t].prevx);
83                 add(updates[t].x);
84             }
85             v[updates[t].i] = updates[t].x;
86             t++;
87         }
88         ans[queries[i].idx] = sum;
89     }
90 }
91 signed main()
92 {
93     ios_base::sync_with_stdio(false);
94     cin.tie(NULL);
95     cin >> n >> q;
96     for (int i = 0; i < n; i++)
97     {
98         cin >> v[i];

```

```

98     vv[i] = v[i];
99 }
100 for (int i = 0; i < q; i++)
101 {
102     int type;
103     cin >> type;
104     if (type == 1)
105     {
106         mo::update curr;
107         cin >> curr.i >> curr.x;
108         curr.prevx = vv[curr.i];
109         vv[curr.i] = curr.x;
110         mo::updates.pb(curr);
111     }
112     else
113     {
114         mo::query curr;
115         cin >> curr.l >> curr.r;
116         curr.r--;
117         curr.idx = mo::queries.size();
118         curr.t = mo::updates.size();
119         mo::queries.pb(curr);
120     }
121 }
122 mo::run();
123 for (auto const &i : mo::ans)
124     cout << i << endl;
125 }

```

3.4.3 Blocking

We partition the array into blocks of size `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 $\frac{x}{\text{block_size}}$.

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 $[0 \dots r]$, calculate

$$\sum_{i=0}^{R-1} \text{blocks}[i] + \sum_{R \cdot \text{block_size}}^r \text{nums}[i]$$

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

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

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Sqrt {
5     int block_size;
6     vector<int> nums;
7     vector<long long> blocks;
8     Sqrt(int sqrtn, vector<int> &arr) : block_size(sqrtn), blocks(sqrtn,
9         0) {
10         nums = arr;
11         for (int i = 0; i < nums.size(); i++) {
12             blocks[i / block_size] += nums[i];
13         }
14     }
15
16     /** O(1) update to set nums[x] to v */
17     void update(int x, int v) {
18         blocks[x / block_size] -= nums[x];

```

```

18     nums[x] = v;
19     blocks[x / block_size] += nums[x];
20 }
21
22 /** O(sqrt(n)) query for sum of [0, r) */
23 long long query(int r) {
24     long long res = 0;
25     for (int i = 0; i < r / block_size; i++) { res += blocks[i]; }
26     for (int i = (r / block_size) * block_size; i < r; i++) {
27         res += nums[i];
28     }
29     return res;
30 }
31
32 /** O(sqrt(n)) query for sum of [l, r) */
33 long long query(int l, int r) { return query(r) - query(l - 1); }
34 };
35
36 int main() {
37     int n, q;
38     cin >> n >> q;
39
40     vector<int> arr(n);
41     for (int i = 0; i < n; i++) { cin >> arr[i]; }
42     Sqrt sq((int)ceil(sqrt(n)), arr);
43
44     for (int i = 0; i < q; i++) {
45         int t, l, r;
46         cin >> t >> l >> r;
47         if (t == 1) {
48             sq.update(l - 1, r);
49         } else {
50             cout << sq.query(l, r) << "\n";
51         }
52     }
53 }

```

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 $a[l] = k, a[l+1] = k+1, \dots, a[r-1] = k + (r-l), a[r] = k + (r-l+1)$;

Given two integers l and r , returns the beauty.

```

1 // Example of sqrt decomposition
2 const int N = 400100;
3 const int K = 500;
4
5 int ans[K][N/K], bmi[K][N/K], bmx[K][N/K];
6 int pre[N/K], lazy[N/K], ami[N/K], amx[N/K];
7 int a[N], b[N];
8
9 int calc(int ini, int k)
10 {
11     ini%=K;
12     k%=K;
13     k -= ini;
14     if(k<0)
15         k+=K;
16     return k;
17 }
18
19 void updt(int id)
20 {
21     int j = id*K;

```

```

22     int mi = a[j];
23     pre[id] = 0;
24     amx[id] = a[j];
25     while(j/K == id)
26     {
27         pre[id] = max(pre[id], a[j] - mi);
28         mi = min(mi, a[j]);
29         ami[id] = mi;
30         amx[id] = max(amx[id], a[j]);
31         j++;
32     }
33 }
34
35 void push(int id)
36 {
37     // cout << id << ' ' << lazy[id] << '\n';
38     if(lazy[id] == -1)
39         return;
40     int ini = id*K;
41     rep(i, 0, K)
42     {
43         a[ini+i] = b[lazy[id]+i];
44     }
45     lazy[id] = -1;
46 }
47
48 void build(int n)
49 {
50     int mi;
51     rep(i, 0, K)
52     {
53         rep(j, i, n)
54         {
55             if((j-i)%K == 0)
56             {
57                 mi = b[j];
58                 ans[i][(j-i)/K] = 0;
59                 bmx[i][(j-i)/K] = b[j];
60             }
61             ans[i][(j-i)/K] = max(ans[i][(j-i)/K], b[j] - mi);
62             mi = min(mi, b[j]);
63             bmi[i][(j-i)/K] = mi;
64             bmx[i][(j-i)/K] = max(bmx[i][(j-i)/K], b[j]);
65         }
66     }
67     rep(j, 0, n)
68     {
69         if((j)%K == 0)
70         {
71             mi = a[j];
72             pre[(j)/K] = 0;
73             lazy[j/K] = -1;
74             amx[j/K] = a[j];
75         }
76         pre[(j)/K] = max(pre[(j)/K], a[j] - mi);
77         mi = min(mi, a[j]);
78         ami[j/K] = mi;
79         amx[j/K] = max(amx[j/K], a[j]);
80     }
81     // rep(j, 0, K)
82     //     cout << j << ' ' << pre[j] << '\n';
83 }
84
85 int ask(int l, int r)
86 {
87     // if(l%K != 0)

```

```

88     // {
89         push(l/K);
90         updt(l/K);
91     // }
92     int rs = 0, mi = a[l];
93     // cout << l << ' ' << l%K << '\n';
94     while(l%K != 0 && l <= r)
95     {
96         rs = max(rs, a[l]-mi);
97         mi = min(mi, a[l++]);
98     }
99     while(l+K-1 <= r)
100     {
101         // cout << "OPA " << l << ' ' << pre[l/K] << ' ' << amx[l/K] <<
102         "\n";
103         rs = max(rs, pre[l/K]);
104         rs = max(rs, amx[l/K]-mi);
105         mi = min(mi, ami[l/K]);
106         l+=K;
107     }
108     // cout << l << ' ' << r << ' ' << rs << "<<<\n";
109     if(l <= r){
110         push(l/K);
111         updt(l/K);
112         // cout << rs << "<<<\n";
113         while(l <= r)
114         {
115             // cout << a[l] << ' ';
116             rs = max(rs, a[l]-mi);
117             mi = min(mi, a[l++]);
118         }
119         // cout << '\n';
120     }
121     return rs;
122 }
123 void updt(int l, int r, int k)
124 {
125     int id = calc(l, k);
126     if(l%K != 0)
127     {
128         push(l/K);
129         while(l%K != 0 && l <= r)
130         {
131             a[l++] = b[k++];
132         }
133         updt((l-1)/K);
134     }
135     while(l+K-1 <= r)
136     {
137         // cout << l << ' ' << k << "<<<\n";
138         // cout << id << ' ' << pre[l/K] << ' ' << ans[id][(k-id)/K] <<
139         '\n';
140         pre[l/K] = ans[id][(k-id)/K];
141         amx[l/K] = bmx[id][(k-id)/K];
142         ami[l/K] = bmi[id][(k-id)/K];
143         lazy[l/K] = k;
144         l+=K;
145         k+=K;
146     }
147     if(l <= r)
148     {
149         // cout << l << ' ' << k << "<<<<<\n";
150         push(r/K);
151         while(l <= r)
152             a[l++] = b[k++];

```

```

152         updt(r/K);
153
154     }
155 }

```

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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int n, q;
5 vector<int> arr;
6 vector<long long> prefix;
7
8 /** Build the prefix array for arr */
9 void build() {
10     prefix[0] = 0;
11     for (int i = 1; i <= n; i++) { prefix[i] = prefix[i - 1] + arr[i - 1]; }
12 }
13
14 int main() {
15     cin >> n >> q;
16     arr.resize(n);
17     for (int i = 0; i < n; i++) { cin >> arr[i]; }
18     prefix.assign(n + 1, 0);
19     build();
20
21     vector<pair<int, int>> updates;
22     for (int i = 0; i < q; i++) {
23         int type, a, b;
24         cin >> type >> a >> b;
25         if (type == 1) {
26             a--;
27             updates.push_back({a, b - arr[a]});
28             arr[a] = b;
29         } else {
30             long long ans = prefix[b] - prefix[a - 1];
31             a--, b--;
32             for (const auto &[idx, val] : updates) {
33                 if (a <= idx && idx <= b) { ans += val; }
34             }
35             cout << ans << "\n";
36         }
37
38         // rebuild the prefix array once the buffer gets to sqrt(n)
39         if (updates.size() * updates.size() >= n) {
40             updates.clear();
41             build();
42         }
43     }
44 }

```

3.5 Treap

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

```

1 #define tipo int
2 mt19937 mt_rand(time(0));
3 struct Node{

```

```

4     int p,cnt;
5     tipo value;
6     bool rev;
7     struct Node * l;
8     struct Node * r;
9     node() { }
10    node(tipo value) : value(value), p(mt_rand()),cnt(1),rev(false), l(
        NULL), r(NULL) { }
11
12 };
13 typedef Node * pnode;
14
15 int cnt (pnode t) {
16     return t ? t->cnt : 0;
17 }
18
19 void upd_cnt (pnode t) {
20     if (t){
21
22         t->cnt = 1 + cnt(t->l) + cnt (t->r);
23     }
24 }
25
26 void push (pnode it) {
27     if (it && it->rev) {
28         it->rev = false;
29         swap (it->l, it->r);
30         if (it->l) it->l->rev ^= true;
31         if (it->r) it->r->rev ^= true;
32         //upd_cnt(it);
33     }
34 }
35
36 void split(pnode t, pnode &l, pnode &r,int key,int add = 0){
37
38     if(!t){
39         return void( l = r = NULL );
40     }
41     push(t);
42     int cur_key=add+cnt(t->l);
43     if(key<=cur_key){
44         split(t->l,l,t->l,key,add);
45         r=t;
46     }else{
47         split(t->r,t->r,r,key,add+1+cnt(t->l));
48         l=t;
49     }
50     upd_cnt (t);
51 }
52
53 void merge(pnode &t,pnode l, pnode r){
54     push(l);
55     push(r);
56     if (!l || !r)
57         t = l ? l : r;
58     else if(l->p>r->p){
59         merge(l->r,l->r,r),t=l;
60     }else{
61         merge(r->l,l,r->l),t=r;
62     }
63     upd_cnt (t);
64 }
65
66 void insert(pnode &t,pnode n,int key){
67     pnode t1,t2;
68     split(t,t1,t2,key);

```

```

69     // cout<<"oi\n";
70     merge(t1,t1,n);
71     merge(t,t1,t2);
72 }
73
74 void reverse (pnode t, int l, int r) {
75     if(l>=r)
76         return;
77     pnode t1,t2,t3;
78     split (t, t1, t2, l);
79     split (t2, t2, t3, r-l+1);
80     t2->rev ^= true;
81     merge (t, t1, t2);
82     merge (t, t, t3);
83 }
84 void output (pnode t) {
85     if (!t) return;
86     // push (t);
87     output (t->l);
88     cout<<t->value<<" ";
89     output (t->r);
90 }
91 void clr (pnode &t) {
92     if (!t) return;
93     clr (t->l);
94     clr (t->r);
95     delete t;
96     t=NULL;
97     // cout<<t<<"\n";
98 }
99
100 void erase(pnode &t,int key)
101 {
102     pnode t1,t2,t3;
103     split(t,t1,t2,key-1);
104     split(t2,t2,t3,key);
105     merge(t,t1,t3);
106 }

```

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 .

```

1 int pai[N];
2 int rk[N];
3 int tmp[N];
4
5 int findx(int x, int t)
6 {
7     if(tmp[x] > t || tmp[x] == 0)
8         return x;
9     return findx(pai[x],t);
10 }
11 int cs = 0;
12 void merge(int a, int b)
13 {
14     a = findx(a,cs);
15     b = findx(b,cs);
16     if(a != b)
17     {
18         if(rk[a] < rk[b])
19             swap(a,b);

```



```

20     rk[a] = max(rk[a], rk[b]+1);
21     pai[b] = a;
22     tmp[b] = ++cs;
23     tmp[a] = 0;
24 }
25 }

```

3.6.2 DSU with rollback

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

```

1
2 int pai[N];
3 vector<int> rk[N];
4
5 int findx(int x)
6 {
7     if(pai[x] == x)
8         return x;
9     return findx(pai[x]);
10 }
11 int cs = 0;
12 vector<pii> rmv;
13 void merge(int a, int b)
14 {
15     a = findx(a);
16     b = findx(b);
17     if(a != b)
18     {
19         if(rk[a].back() < rk[b].back())
20             swap(a,b);
21         rk[a].pb(max(rk[a].back(), rk[b].back()+1));
22         pai[b] = a;
23         rmv.pb(a,b);
24         cs++;
25     }
26 }
27
28 void roll_back()
29 {
30     pai[rmv.back().sd] = rmv.back().fd;
31     rk[rmv.back().fd].pop_back();
32     rmv.pop_back();
33     cs--;
34 }

```

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 idea 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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //template
4 const int N = 50100;
5 const int K = 250;
6
7 //DSU with Rollback
8
9 vector<pii> ed;
10 namespace mo
11 {
12     struct query
13     {
14         int idx, l, r;
15     };
16     vector<query> queries;
17     vector<int> ans;
18
19     bool cmp(query x, query y)
20     {
21         if (x.l / K != y.l / K)
22             return x.l / K < y.l / K;
23         return x.r < y.r;
24     }
25     void run(int n)
26     {
27         dsu::init(n);
28         sort(queries.begin(), queries.end(), cmp);
29         ans.resize(queries.size());
30         int lim, cr;
31         int check, at = -1;
32         for (int i = 0; i < queries.size(); i++)
33         {
34             int al = queries[i].l;
35             int ar = queries[i].r;
36             if(al/K != at)
37             {
38                 while(dsu::cs)
39                     dsu::roll_back();
40                 at = al/K;
41                 lim = min<int>(at*K + K-1, ed.size()-1);
42                 cr = lim;
43             }
44             if(ar/K == at)
45             {
46                 check = 0;
47                 rep(j, al, ar+1)
48                     dsu::merge(ed[j].ft, ed[j].sd);
49             } else
50             {
51                 while(cr < ar)
52                 {
53                     cr++;
54                     dsu::merge(ed[cr].ft, ed[cr].sd);
55                 }
56                 check = dsu::cs;
57                 rep(j, al, lim+1)
58                     dsu::merge(ed[j].ft, ed[j].sd);
59             }
60
61             ans[queries[i].idx] = n-dsu::cs;
62             while(dsu::cs != check)
63                 dsu::roll_back();
64         }
65     }

```

```

66 }
67
68
69 void test()
70 {
71     int n, m;
72     cin >> n >> m;
73     rep(i, 0, m)
74     {
75         int u, v;
76         cin >> u >> v;
77         ed.eb(u,v);
78     }
79     int k;
80     cin >> k;
81     rep(i, 0, k)
82     {
83         mo::query curr;
84         cin >> curr.l >> curr.r;
85         curr.l--;
86         curr.r--;
87         curr.idx = i;
88         mo::queries.pb(curr);
89     }
90     mo::run(n);
91     rep(i, 0, k)
92         cout << mo::ans[i] << '\n';
93 }
94
95
96 int32_t main()
97 {
98     ios::sync_with_stdio(false);
99     cin.tie(NULL);
100     int t = 1;
101     // cin >> t;
102     while(t--)
103         test();
104     return 0;
105 }

```

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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //template
4 const int N = 300100;
5 //DSU with Rollback
6 map<pii,int> id;
7 struct query

```

```

8 {
9     int l, r, u, v;
10 };
11 int ans[N], n;
12
13 vector<query> queries;
14 int sum = 0;
15 // vector<pair<pii,vector<int>>>
16 void solve(vector<int> &nw, int l, int r)
17 {
18     if(l > r)
19         return;
20     // sum++;
21     int check = dsu::cs;
22     if(l == r)
23     {
24         // cout << l << ' ' << r << "<<<\n";
25
26         for(int i:nw)
27             dsu::merge(queries[i].u,queries[i].v);
28         ans[l] = n-dsu::cs;
29
30     }else
31     {
32         int mid = (l+r) >> 1;
33         vector<int> lef,rig;
34         // cout << l << ' ' << r << "<<<\n";
35         for(int i:nw)
36         {
37             if(queries[i].l <=l && queries[i].r >= r){
38                 // cout << queries[i].l << ' ' << queries[i].r << ' ' << queries
39                 [i].u << ' ' << queries[i].v << '\n';
40                 dsu::merge(queries[i].u,queries[i].v);
41             }
42             else {
43                 if(queries[i].l <= mid && queries[i].r >= l)
44                     lef.pb(i);
45                 if(queries[i].r > mid && queries[i].l <= r)
46                     rig.pb(i);
47             }
48             // nw.clear();
49             solve(lef,l,mid);
50             solve(rig,mid+1,r);
51         }
52         while(dsu::cs != check)
53             dsu::roll_back();
54     }
55     vector<int> nw;
56     void test()
57     {
58         int m;
59         cin >> n >> m;
60         dsu::init(n);
61         int pass = 1;
62         rep(i, 1, m+1)
63         {
64             char ch;
65             cin >> ch;
66             if(ch == '?')
67             {
68                 pass++;
69             }else if(ch == '+')
70             {
71                 int u, v;
72                 cin >> u >> v;

```

```
73     if(u > v)
74         swap(u,v);
75     id[mp(u,v)] = pass;
76 }else
77 {
78     query at;
79     cin >> at.u >> at.v;
80     if(at.u > at.v)
81         swap(at.u,at.v);
82     at.l = id[mp(at.u,at.v)];
83     at.r = pass-1;
84     id[mp(at.u,at.v)] = 0;
85     if(at.l <= at.r)
86     {
87         nw.pb(queries.size());
88         queries.pb(at);
89     }
90 }
91 }
92 for(auto [ed,x]:id)
93 {
94     if(!x)
95         continue;
96     query at;
97     at.u = ed.ft;
98     at.v = ed.sd;
99     at.l = x;
100    at.r = pass-1;
101    if(at.l <= at.r)
102    {
103        nw.pb(queries.size());
104        queries.pb(at);
105    }
106 }
107 // for(auto at:queries)
108 //     cout << at.l << ' ' << at.r << ' ' << at.u << ' ' << at.v << '\n';
109 solve(nw,1,pass-1);
110 rep(i, 1, pass)
111     if(ans[i] != 0)
112         cout << ans[i] << '\n';
113
114 // cout << sum << "<<\n";
115 }
116
117
118 int32_t main()
119 {
120     ios::sync_with_stdio(false);
121     cin.tie(NULL);
122     int t = 1;
123     // cin >> t;
124     while(t--)
125         test();
126     return 0;
127 }
```

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.

```
1 /*
2     Indexado em 1
3
4     Alteracoes:
5
```

```

6      Ver se precisa adicionar coisa no gp do smol
7  */
8  struct DSU{
9      // o que cada grupo carrega de informacao
10     struct gp{
11         // se precisa adicionar coisa a mais
12         int tam = 0;
13
14         // inicializar o smol
15         void init(int x){
16             // se precisa inicializar coisa a mais, passa por parametro
17             tam=1;
18         }
19         void clear(){
20             tam = 0;
21         }
22         ~gp(){}
23     };
24
25     vector<int> repre;
26     vector<gp> smol;
27     int n;
28
29     // inicializar passando a qtd de vertices, grupos iniciais, se t = 0
30     // entao so aloca a memoria sem inicializar
31     DSU (int n_,int t = 1) : n(n_){
32         repre.resize(n+10);
33         smol.resize(n+10);
34         if(t){
35             for(int i=1; i<=n; i++){
36                 repre[i]=i;
37                 smol[i].init(i);
38             }
39         }
40         ~DSU(){
41             repre.clear();
42             smol.clear();
43         }
44         void init(int n){
45             this->n = n;
46             for(int i=1; i<=n; i++){
47                 repre[i] = i;
48                 smol[i].init(i);
49             }
50         }
51         void clear(){
52             for(int i=1; i<=n; i++){
53                 smol[i].clear();
54             }
55         }
56
57
58         // achar o representante do u
59         int rep(int u){
60             return repre[u] = (repre[u] == u ? u : rep(repre[u]));
61         }
62         // pegar o smol do cara u
63         gp & smoll(int u){
64             return smol[rep(u)];
65         }
66
67         // unir u e v
68         void unite(int u,int v,int t){
69             u=rep(u);
70             v=rep(v);

```

```
71         if(u == v) return;
72
73         auto &x=smol[u];
74         auto &y=smol[v];
75         if(y.tam > x.tam){
76             unite(v,u,t);
77             return;
78         }
79
80         // da merge nos smols
81         merge(x,y,t);
82         repre[v]=u;
83     }
84
85     // fazer o merge de 2 grupos
86     void merge(gp &x, gp &y,int t){
87         // faz o merge se precisar
88         x.tam += y.tam;
89         y.clear();
90     }
91 };
```


Capítulo IV

Segment Tree

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

4.1 Sem/Com Lazy

```
1  /*
2      Indexado de 1.
3      Responde uma operacao num subarray, suporta update em range
4      Init: O(4*N).
5      Query: O(4*log(N))
6      Update: O(4*log(N))
7  */
8  template <class TT = int>
9  struct Seg{
10     // inicializar so o tamanho da seg, n fazer o build
11     Seg(int n_) : n(n_){
12         seg.resize(n<<2);
13         lazy.resize(n<<2);
14     }
15
16     // fazer o build da estrutura
17     void init(int n_new, TT * a){
18         n = n_new;
19         vec = a;
20         build(1,1,n);
21     }
22
23     // o que vai ter dentro do no de cada seg
24     struct node{
25
26         bool operator ==(const node &ot)const{
27             return true;
28         }
29     };
30     // o que vai ter dentro do no de cada lazy
31     struct sono{
32
33         bool operator ==(const sono &ot)const{
34             return true;
35         }
36     };
37
38     TT * vec;
39     int n;
40     // no nulo
41     const node off = {};
42     // lazy nula
43     const sono off_lazy = {};
44     vector<node> seg;
45     vector<sono> lazy;
46     // operacao de unir dois nos
47     node merge(node x, node y){
48         if(x == off) return y;
```

```

49         if(y == off) return x;
50
51     }
52
53     // coisar a lazy pra baixo
54     void push(int u,int tl,int tr){
55         if(tl == tr || lazy[u] == off_lazy) return;
56         // atualizar os filhos
57
58         // atualizar as lazies dos filhos
59
60         lazy[u] = off_lazy;
61     }
62
63     // inicializar a seg
64     void build(int u,int tl,int tr){
65         if(tl == tr){
66             // inicializar os caras bases
67             seg[u] = {};
68             lazy[u] = off_lazy;
69             return;
70         }
71         int tmid = tl + tr;
72         tmid >>= 1;
73         build(lef(u), tl, tmid);
74         build(rig(u), tmid+1, tr);
75         seg[u] = merge(seg[lef(u)], seg[rig(u)]);
76         lazy[u] = off_lazy;
77     }
78
79     // consulta em range
80     node query_(int u,int tl,int tr,int l, int r){
81         if(l > r) return off;
82         if(tl == l && tr == r) return seg[u];
83         push(u, tl, tr);
84         int tmid= tl + tr;
85         tmid >>= 1;
86         return merge(query_(lef(u), tl, tmid, l, min(tmid,r)), query_(
87         rig(u), tmid+1, tr, max(tmid+1,l), r));
88     }
89     node query(int l, int r){
90         return query_(1, 1, n, l, r);
91     }
92
93     // update em range
94     void update_(int u, int tl, int tr, int l, int r, TT x){
95         if(l > r) return;
96         if(tl == l && tr == r){
97             // atualizar a seg e o lazy
98
99             return;
100         }
101         push(u, tl, tr);
102         int tmid = tl + tr;
103         tmid >>= 1;
104         update_(lef(u), tl, tmid, l, min(tmid,r), x);
105         update_(rig(u), tmid+1, tr, max(tmid+1,l), r, x);
106         seg[u] = merge(seg[lef(u)], seg[rig(u)]);
107     }
108     void update(int l, int r, TT x){
109         update_(1, 1, n, l, r, x);
110     }
};

```

4.2 Implicita

Essa eu roubei do tiagodfs XD

```

1 // SegTree Implicita O(nlogMAX)
2
3 struct node{
4     int val;
5     int l, r;
6     node(int a=0, int b=0, int c=0){
7         l=a;r=b;val=c;
8     }
9 };
10
11 int idx=2; // 1-> root / 0-> zero element
12 node t[8600010];
13 int N;
14
15 int merge(int a, int b){
16     return a + b;
17 }
18
19 void update(int pos, int x, int i=1, int j=N, int no=1){
20     if(i==j){
21         t[no].val+=x;
22         return;
23     }
24     int meio = (i+j)/2;
25
26     if(pos<=meio){
27         if(t[no].l==0) t[no].l=idx++;
28         update(pos, x, i, meio, t[no].l);
29     }
30     else{
31         if(t[no].r==0) t[no].r=idx++;
32         update(pos, x, meio+1, j, t[no].r);
33     }
34
35     t[no].val=merge(t[t[no].l].val, t[t[no].r].val);
36 }
37
38 int query(int A, int B, int i=1, int j=N, int no=1){
39     if(B<i or j<A)
40         return 0;
41     if(A<=i and j<=B)
42         return t[no].val;
43
44     int mid = (i+j)/2;
45
46     int ans1 = 0, ansr = 0;
47
48     if(t[no].l!=0) ans1 = query(A, B, i, mid, t[no].l);
49     if(t[no].r!=0) ansr = query(A, B, mid+1, j, t[no].r);
50
51     return merge(ans1, ansr);
52 }

```

4.3 Implicita com lazy

```

1 struct node{
2     pll val;
3     ll lazy;
4     ll l, r;
5     node(){
6         l=-1;r=-1;val={0,0};lazy=0;

```

```

7     }
8 };
9
10 node tree[40*MAX];
11 int id = 2;
12 ll N=1e9+10;
13
14 pll merge(pll A, pll B){
15     if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
16     return (A.ff<B.ff ? A:B);
17 }
18
19 void prop(ll l, ll r, int no){
20     ll mid = (l+r)/2;
21     if(l!=r){
22         if(tree[no].l!=-1){
23             tree[no].l = id++;
24             tree[tree[no].l].val = {0, mid-l+1};
25         }
26         if(tree[no].r!=-1){
27             tree[no].r = id++;
28             tree[tree[no].r].val = {0, r-(mid+1)+1};
29         }
30         tree[tree[no].l].lazy += tree[no].lazy;
31         tree[tree[no].r].lazy += tree[no].lazy;
32     }
33     tree[no].val.ff += tree[no].lazy;
34     tree[no].lazy=0;
35 }
36
37 void update(int a, int b, int x, ll l=0, ll r=2*N, ll no=1){
38     prop(l, r, no);
39     if(a<=l and r<=b){
40         tree[no].lazy += x;
41         prop(l, r, no);
42         return;
43     }
44     if(r<a or b<l) return;
45     int m = (l+r)/2;
46     update(a, b, x, l, m, tree[no].l);
47     update(a, b, x, m+1, r, tree[no].r);
48
49     tree[no].val = merge(tree[tree[no].l].val, tree[tree[no].r].val);
50 }
51
52 pll query(int a, int b, int l=0, int r=2*N, int no=1){
53     prop(l, r, no);
54     if(a<=l and r<=b) return tree[no].val;
55     if(r<a or b<l) return {INF, 0};
56     int m = (l+r)/2;
57     int left = tree[no].l, right = tree[no].r;
58
59     return tree[no].val = merge(query(a, b, l, m, left),
60                                query(a, b, m+1, r, right));
61 }

```

4.4 Persistente

Solution for Spoj Count on a Tree, Given a tree there are Q queries where you need to find the k th minimum weight 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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int,int> pii;
5
6 #define lef(x) (tree[x].l)
7 #define rig(x) (tree[x].r)
8
9 const int N = 200100;
10 const ll emp = 0; //NULL VALUE
11
12 struct node
13 {
14     int l, r;
15     ll v;
16     node(): l(-1), r(-1), v(0){};
17     node(int l, int r, ll v): l(l), r(r), v(v){};
18 };
19
20 ll a[N];
21 node tree[N << 6];
22 int last = 2;
23 int head[N];
24 ll f(ll a, ll b){return a+b;}//function
25
26 void build(int p, int tl, int tr)
27 {
28     if(tl == tr)
29     {
30         tree[p].v = a[tl];
31         return;
32     }
33
34     int tmid = (tl + tr) >> 1;
35     tree[p].l = last++;
36     tree[p].r = last++;
37     build(lef(p), tl, tmid);
38     build(rig(p), tmid+1, tr);
39     tree[p].v = f(tree[lef(p)].v, tree[rig(p)].v);
40
41 }
42
43 void updt(int p,int np, int tl, int tr, int k, ll v)
44 {
45     tree[np] = tree[p];
46     if(tl == tr){
47         tree[np].v = v;
48         return;
49     }
50     int tmid = (tl+tr) >> 1;
51     if (tmid < k){
52         tree[np].r = last++;
53         updt(rig(p), rig(np), tmid+1, tr, k, v);
54     }
55     else{
56         tree[np].l = last++;
57         updt(lef(p), lef(np), tl, tmid, k, v);
58     }
59     tree[np].v = f(tree[lef(np)].v, tree[rig(np)].v);
60
61 }
62
63 ll query(int p, int tl, int tr, int l, int r)
64 {
65     if(l > r)
66         return emp;

```

```

67     if(tl == l && tr == r)
68         return tree[p].v;
69     int tmid = (tl+tr) >> 1;
70     return f(query(lef(p), tl, tmid, l, min(r,tmid)),
71             query(rig(p), tmid+1, tr, max(l,tmid+1), r));
72 }
73 int walk(int u,int v,int lc,int plc,int d,int tl,int tr){
74     if(tl==tr)
75         return tl;
76     int at=0;
77     int mid=(tl+tr)/2;
78     at+=tree[lef(u)].v;
79     at+=tree[lef(v)].v;
80     at-=tree[lef(lc)].v;
81     at-=tree[lef(plc)].v;
82     // cout<<at<<" "<<tl<<" "<<tr<<"\n";
83     if(at>=d)
84         return walk(tree[u].l,tree[v].l,tree[lc].l,tree[plc].l,d,tl,mid);
85     if(at<d)
86         return walk(tree[u].r,tree[v].r,tree[lc].r,tree[plc].r,d-at,mid+1,tr);
87 }

```

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:

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

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

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int,int> pii;
5
6 #define lef(x) (x << 1)
7 #define rig(x) (lef(x) | 1)
8
9 const int N = 200100;
10 const ll emp = 0; //NULL VALUE
11
12 ll a[N],tree[N << 2];
13 pair<ll,ll> lazy[N << 2];
14
15 ll f(ll a, ll b){return a+b;}//function
16
17 pair<ll,ll> ff(pair<ll,ll> a, pair<ll,ll> b){return pair<ll,ll>{a.first+
18     b.first,a.second+b.second};}//function
19
20 //node p is always ok
21
22 // PA FORMULAS
23 // an = a1 + r * (n-1)
24 // sn = (a1 + an) * (n) / 2
25 void push(int p, int tl, int tr)
26 {
27     if(tl == tr || lazy[p] == pair<ll,ll>{emp,emp})
28         return;
29     int tmid = (tl+tr) >> 1;
30
31     //solve propagations
32     ll at = lazy[p].first + lazy[p].second * (tmid-tl);
33     ll sum = (at + lazy[p].first) * (tmid-tl+1) >> 1;
34     tree[lef(p)] = f(tree[lef(p)], sum);

```

```

35     at += lazy[p].second;
36     ll at2 = lazy[p].first + lazy[p].second * (tr - tl);
37     sum = (at + at2) * (tr-tmid) >> 1;
38     tree[rig(p)] = f(tree[rig(p)], sum);
39
40     //update lazy
41     lazy[lef(p)] = ff(lazy[lef(p)], lazy[p]);
42     lazy[rig(p)] = ff(lazy[rig(p)], pair<ll,ll>{at,lazy[p].second});
43
44     //clean lazy
45     lazy[p] = {emp,emp};
46 }
47
48 void build(int p, int tl, int tr)
49 {
50     if(tl == tr)
51     {
52         tree[p] = a[tl];
53         return;
54     }
55     int tmid = (tl + tr) >> 1;
56     build(lef(p), tl, tmid);
57     build(rig(p), tmid+1, tr);
58     tree[p] = f(tree[lef(p)], tree[rig(p)]);
59 }
60
61 void updt(int p, int tl, int tr, int l, int r, ll v)
62 {
63     if(l > r)
64         return ;
65     if(tl == l && tr == r){
66         ll at = tl-v+1;
67         ll at2 = tr -v+1;
68         ll sum = (at + at2) * (tr-tl+1) >> 1;
69         tree[p] = f(tree[p], sum);
70
71         lazy[p] = ff(lazy[p], {tl-v+1,1});
72         return;
73     }
74     //push if go down
75     push(p,tl,tr);
76
77     int tmid = (tl+tr) >> 1;
78     updt(lef(p), tl, tmid, l, min(r,tmid), v);
79     updt(rig(p), tmid+1, tr, max(l,tmid+1), r, v);
80     tree[p] = f(tree[lef(p)], tree[rig(p)]);
81 }
82
83
84
85
86
87 ll query(int p, int tl, int tr, int l, int r)
88 {
89     if(l > r)
90         return emp;
91     if(tl == l && tr == r)
92         return tree[p];
93     //push if go down
94     push(p,tl,tr);
95     int tmid = (tl+tr) >> 1;
96     return f(query(lef(p), tl, tmid, l, min(r,tmid)),
97             query(rig(p), tmid+1, tr, max(l,tmid+1), r));
98 }

```

4.6 Sub-segmento contíguo 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.

```

1  const int N = 200100;
2  const ll emp = -2000000000000000LL; //NULL VALUE
3
4  struct dt
5  {
6      ll l, r, mx, tot;
7      dt(){};
8      dt(ll a):l(a), r(a), mx(max(a,0LL)), tot(a){};
9  };
10 ll a[N];
11 dt tree[N << 2];
12
13 dt f(dt a, dt b){
14     dt x;
15     x.mx = max(a.mx, b.mx);
16     x.mx = max(x.mx, a.r + b.l);
17     x.r = max(a.r + b.tot, b.r);
18     x.l = max(b.l + a.tot, a.l);
19     x.tot = a.tot + b.tot;
20     return x;
21 }//function
22
23
24 void build(int p, int tl, int tr)
25 {
26     if(tl == tr)
27     {
28         tree[p] = dt(a[tl]);
29         return;
30     }
31     int tmid = (tl + tr) >> 1;
32     build(lef(p), tl, tmid);
33     build(rig(p), tmid+1, tr);
34     tree[p] = f(tree[lef(p)], tree[rig(p)]);
35 }
36
37
38
39 void updt_point(int p, int tl, int tr, int k, ll v)
40 {
41     if(tl == tr){
42         tree[p] = dt(v);
43         return;
44     }
45
46     int tmid = (tl+tr) >> 1;
47     if (tmid < k)
48         updt_point(rig(p), tmid+1, tr, k, v);
49     else
50         updt_point(lef(p), tl, tmid, k, v);
51     tree[p] = f(tree[lef(p)], tree[rig(p)]);
52     // cout << tl << " " << tr << ' ' << tree[p].mx << '\n';
53 }

```

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 occurrences 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 (*val*, *cnt*), where *val* is the minimum value and *cnt* is the number occurrences of the minimum value.

If node *c* has two children *a* and *b*, then

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

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

```

1  const int N = 1000001;
2
3  vector<pii> event[2*N + 100][2];
4
5  pii stree[8*N+ 100];
6  int lazy[8*N+ 100];
7
8  pii merge(pii a, pii b)
9  {
10     if(a.ft == b.ft)
11         a.sd+=b.sd;
12     if(a.ft <= b.ft)
13         return a;
14     return b;
15 }
16
17 void push(int p, int tl, int tr)
18 {
19     if(tl != tr && lazy[p] != 0)
20     {
21         stree[lef(p)].ft += lazy[p];
22         stree[rig(p)].ft += lazy[p];
23         lazy[lef(p)] += lazy[p];
24         lazy[rig(p)] += lazy[p];
25     }
26     lazy[p] = 0;
27 }
28
29 void build(int p, int tl, int tr)
30 {
31     if(tl == tr)
32     {
33         stree[p] = {0,1};
34         return;
35     }
36     int tmid = (tl+tr) >> 1;
37     build(lef(p), tl, tmid);
38     build(rig(p), tmid+1, tr);
39     stree[p] = merge(stree[lef(p)],stree[rig(p)]);
40 }
41
42
43 void updt(int p, int tl, int tr,int l, int r, int x)
44 {
45     if(l > r)
46         return;
47     if(tl == l && tr == r)
48     {
49         stree[p].ft+=x;
50         lazy[p]+=x;
51         return;
52     }
53     int tmid = (tl+tr) >> 1;
54     push(p, tl, tr);
55     updt(lef(p), tl, tmid, l, min(r,tmid), x);
56     updt(rig(p), tmid+1, tr, max(l, tmid+1), r, x);
57     stree[p] = merge(stree[lef(p)],stree[rig(p)]);
58 }
59

```

```

60
61 void test()
62 {
63     int n;
64     cin >> n;
65     pii pt;
66     rep(i, 1, n+1)
67     {
68         // cout << i << endl;
69         int x1,y1,x2,y2;
70         cin >> x1 >> y1 >> x2 >> y2;
71         pii at = pii{y1+N,y2+N-1};
72         // cout << x1+N << ' ' << x2+N << endl;
73         event[x1+N][0].pb(at);
74         event[x2+N][1].pb(at);
75     }
76     build(1,1,2*N);
77     ll rs = 0;
78     rep(i, 0, 2*N)
79     {
80         for(auto [l,r]:event[i][0])
81         {
82             // cout << l << ' ' << r << '\n';
83             updt(1, 1, 2*N, l, r, 1);
84         }
85         for(auto [l,r]:event[i][1])
86         {
87             // cout << l << ' ' << r << '\n';
88             updt(1, 1, 2*N, l, r, -1);
89         }
90         // if(stree[1].sd != 2*N)
91         //     cout << i << ' ' << stree[1] << '\n';
92         if(stree[1].ft != 0)
93             rs += 2*N;
94         else
95             rs += 2*N-stree[1].sd;
96     }
97     cout << rs << '\n';
98 }

```

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 $\mathcal{O}(n \log^2(n))$

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef long long ll;
5
6 #define all(x) x.begin(), x.end()
7 #define lef(x) (x << 1)
8 #define rig(x) (lef(x) | 1)
9 #define pb push_back
10
11 const int N = 30010;
12
13 int a[N];
14 vector<int> tree[N << 2];

```

```
15
16 void build(int x, int tl, int tr){
17     if(tl == tr){
18         tree[x].pb(a[tl]);
19     }
20     else{
21         int tm = tl + tr >> 1;
22         build(lef(x), tl, tm);
23         build(rig(x), tm+1, tr);
24         tree[x].resize(tr-tl+1);
25         merge(all(tree[lef(x)]), all(tree[rig(x)]), tree[x].begin());
26     }
27 }
28
29 int get(int x, int tl, int tr, int l, int r, int k){
30     if(tl > r || tr < l)
31         return 0;
32     else if(tl >= l && tr <= r)
33         return (int)tree[x].size() - (upper_bound(all(tree[x]), k) -
34 tree[x].begin());
35     else{
36         int tm = tl + tr >> 1;
37         return get(lef(x),tl,tm,l,r,k) + get(rig(x),tm+1,tr,l,r,k);
38     }
39 }
40
41 int32_t main(){
42     ios::sync_with_stdio(false);
43     cin.tie(nullptr);
44
45     int n;
46     cin>>n;
47     for(int i = 1; i <= n; i++){
48         cin>>a[i];
49     }
50
51     build(1,1,n);
52
53     int q; cin >> q;
54     while(q--){
55         int l, r, x;
56         cin >> l >> r >> x;
57         cout<<get(1,1,n,l,r,x)<<"\n";
58     }
59
60     return 0;
61 }
```

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 a_i - range $l-r$

$O(N * \log(N)^2)$

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int MAXN = 200001; // 1-based
6 const ll llINF=1e18;
7 #define lef(x) (x<<1)
```

```

8  #define rig(x) (lef(x) | 1)
9  int N;
10 ll A[MAXN];
11
12 struct Node {
13     ll sum;    // Sum tag
14     ll max1;   // Max value
15     ll max2;   // Second Max value
16     ll maxc;   // Max value count
17     ll min1;   // Min value
18     ll min2;   // Second Min value
19     ll minc;   // Min value count
20     ll lazy;   // Lazy tag
21 } T[MAXN * 4];
22
23 void merge(int t) {
24     // sum
25     T[t].sum = T[lef(t)].sum + T[rig(t)].sum;
26
27     // max
28     if (T[lef(t)].max1 == T[rig(t)].max1) {
29         T[t].max1 = T[lef(t)].max1;
30         T[t].max2 = max(T[lef(t)].max2, T[rig(t)].max2);
31         T[t].maxc = T[lef(t)].maxc + T[rig(t)].maxc;
32     } else {
33         if (T[lef(t)].max1 > T[rig(t)].max1) {
34             T[t].max1 = T[lef(t)].max1;
35             T[t].max2 = max(T[lef(t)].max2, T[rig(t)].max1);
36             T[t].maxc = T[lef(t)].maxc;
37         } else {
38             T[t].max1 = T[rig(t)].max1;
39             T[t].max2 = max(T[lef(t)].max1, T[rig(t)].max2);
40             T[t].maxc = T[rig(t)].maxc;
41         }
42     }
43
44     // min
45     if (T[lef(t)].min1 == T[rig(t)].min1) {
46         T[t].min1 = T[lef(t)].min1;
47         T[t].min2 = min(T[lef(t)].min2, T[rig(t)].min2);
48         T[t].minc = T[lef(t)].minc + T[rig(t)].minc;
49     } else {
50         if (T[lef(t)].min1 < T[rig(t)].min1) {
51             T[t].min1 = T[lef(t)].min1;
52             T[t].min2 = min(T[lef(t)].min2, T[rig(t)].min1);
53             T[t].minc = T[lef(t)].minc;
54         } else {
55             T[t].min1 = T[rig(t)].min1;
56             T[t].min2 = min(T[lef(t)].min1, T[rig(t)].min2);
57             T[t].minc = T[rig(t)].minc;
58         }
59     }
60 }
61
62 void push_add(int t, int tl, int tr, ll v) {
63     if (v == 0) {
64         return;
65     }
66     T[t].sum += (tr - tl + 1) * v;
67     T[t].max1 += v;
68     if (T[t].max2 != -11INF) {
69         T[t].max2 += v;
70     }
71     T[t].min1 += v;
72     if (T[t].min2 != 11INF) {
73         T[t].min2 += v;

```

```

74     }
75     T[t].lazy += v;
76 }
77
78 // corresponds to a chmin update
79 void push_max(int t, ll v, bool l) {
80     if (v >= T[t].max1) {
81         return;
82     }
83     T[t].sum -= T[t].max1 * T[t].maxc;
84     T[t].max1 = v;
85     T[t].sum += T[t].max1 * T[t].maxc;
86     if (l) {
87         T[t].min1 = T[t].max1;
88     } else {
89         if (v <= T[t].min1) {
90             T[t].min1 = v;
91         } else if (v < T[t].min2) {
92             T[t].min2 = v;
93         }
94     }
95 }
96
97 // corresponds to a chmax update
98 void push_min(int t, ll v, bool l) {
99     if (v <= T[t].min1) {
100         return;
101     }
102     T[t].sum -= T[t].min1 * T[t].minc;
103     T[t].min1 = v;
104     T[t].sum += T[t].min1 * T[t].minc;
105     if (l) {
106         T[t].max1 = T[t].min1;
107     } else {
108         if (v >= T[t].max1) {
109             T[t].max1 = v;
110         } else if (v > T[t].max2) {
111             T[t].max2 = v;
112         }
113     }
114 }
115
116 void pushdown(int t, int tl, int tr) {
117     if (tl == tr)
118         return;
119     // sum
120     int tm = (tl + tr) >> 1;
121     push_add(lef(t), tl, tm, T[t].lazy);
122     push_add(rig(t), tm + 1, tr, T[t].lazy);
123     T[t].lazy = 0;
124
125     // max
126     push_max(lef(t), T[t].max1, tl == tm);
127     push_max(rig(t), T[t].max1, tm + 1 == tr);
128
129     // min
130     push_min(lef(t), T[t].min1, tl == tm);
131     push_min(rig(t), T[t].min1, tm + 1 == tr);
132 }
133
134 void build(int t=1, int tl=0, int tr=N-1) {
135     T[t].lazy = 0;
136     if (tl == tr) {
137         T[t].sum = T[t].max1 = T[t].min1 = A[tl];
138         T[t].maxc = T[t].minc = 1;
139         T[t].max2 = -111INF;

```

```

140     T[t].min2 = llINF;
141     return;
142 }
143
144 int tm = (tl + tr) >> 1;
145 build(lef(t), tl, tm);
146 build(rig(t), tm + 1, tr);
147 merge(t);
148 }
149
150 void update_add(int l, int r, ll v, int t=1, int tl=0, int tr=N-1) {
151     if (r < tl || tr < l) {
152         return;
153     }
154     if (l <= tl && tr <= r) {
155         push_add(t, tl, tr, v);
156         return;
157     }
158     pushdown(t, tl, tr);
159
160     int tm = (tl + tr) >> 1;
161     update_add(l, r, v, lef(t), tl, tm);
162     update_add(l, r, v, rig(t), tm + 1, tr);
163     merge(t);
164 }
165
166 void update_chmin(int l, int r, ll v, int t=1, int tl=0, int tr=N-1) {
167     if (r < tl || tr < l || v >= T[t].max1) {
168         return;
169     }
170     if (l <= tl && tr <= r && v > T[t].max2) {
171         push_max(t, v, tl == tr);
172         return;
173     }
174     pushdown(t, tl, tr);
175
176     int tm = (tl + tr) >> 1;
177     update_chmin(l, r, v, lef(t), tl, tm);
178     update_chmin(l, r, v, rig(t), tm + 1, tr);
179     merge(t);
180 }
181
182 void update_chmax(int l, int r, ll v, int t=1, int tl=0, int tr=N-1) {
183     if (r < tl || tr < l || v <= T[t].min1) {
184         return;
185     }
186     if (l <= tl && tr <= r && v < T[t].min2) {
187         push_min(t, v, tl == tr);
188         return;
189     }
190     pushdown(t, tl, tr);
191
192     int tm = (tl + tr) >> 1;
193     update_chmax(l, r, v, lef(t), tl, tm);
194     update_chmax(l, r, v, rig(t), tm + 1, tr);
195     merge(t);
196 }
197
198 ll query_sum(int l, int r, int t=1, int tl=0, int tr=N-1) {
199     if (r < tl || tr < l) {
200         return 0;
201     }
202     if (l <= tl && tr <= r) {
203         return T[t].sum;
204     }
205     pushdown(t, tl, tr);

```

```
206
207     int tm = (tl + tr) >> 1;
208     return query_sum(l, r, lef(t), tl, tm) + query_sum(l, r, rig(t), tm +
209     1, tr);
210 }
211
212 int main() {
213     int Q;
214
215     cin >> N >> Q;
216     for (int i = 0; i < N; i++) {
217         cin >> A[i];
218     }
219     build();
220     for (int q = 0; q < Q; q++) {
221         int t; cin >> t;
222         if (t == 0) {
223             int l, r;
224             ll x;
225             cin >> l >> r >> x;
226             update_chmin(l, r - 1, x);
227         } else if (t == 1) {
228             int l, r;
229             ll x;
230             cin >> l >> r >> x;
231             update_chmax(l, r - 1, x);
232         } else if (t == 2) {
233             int l, r;
234             ll x;
235             cin >> l >> r >> x;
236             update_add(l, r - 1, x);
237         } else if (t == 3) {
238             int l, r;
239             cin >> l >> r;
240             cout << query_sum(l, r - 1) << '\n';
241         }
242     }
243 }
```

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

```
1 /*
2 WAVELET TREE, encontra o kth menor valor numa range [L,R] com queries em
3    $O(\log n)$ 
4 Espaço e pre-processamento é em  $O(n \log n)$ 
5 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
6 artigo : https://users.dcc.uchile.cl/~jperez/papers/ioiconf16.pdf
7
8 */
9 #include <bits/stdc++.h>
10 using namespace std;
11 #define MAXN 10000
12 int N;
13 struct elem {
14     int val, pos;
15     bool operator< (elem b) const {
16         return val < b.val;
17     }
18 }
```

```

17     }
18 };
19 int* tree[4*MAXN+10];
20 elem temp[MAXN], arr[MAXN], sorted[MAXN];
21 int* merge(int e, int d) {
22     int* num_left = (int*) malloc(sizeof(int) * (d - e + 1));
23     int left = e, right = (e+d)/2+1;
24     int i = 0, cnt = 0;
25     while (left <= (e+d)/2 && right <= d) {
26         if (arr[left].pos <= arr[right].pos) {
27             num_left[i] = ++cnt;
28             temp[i] = arr[left++];
29         }
30         else {
31             num_left[i] = cnt;
32             temp[i] = arr[right++];
33         }
34         i++;
35     }
36     while (left <= (e+d)/2) {
37         num_left[i] = ++cnt;
38         temp[i] = arr[left++];
39         i++;
40     }
41     while (right <= d) {
42         num_left[i] = cnt;
43         temp[i] = arr[right++];
44         i++;
45     }
46     for (int j = 0; j < (d-e+1); j++) {
47         arr[e+j]=temp[j];
48     }
49     return num_left;
50 }
51 void create_tree (int i=1, int e=1, int d=N) {
52     if (e == d) return;
53     else {
54         create_tree(2*i, e, (e+d)/2);
55         create_tree(2*i+1, (e+d)/2 + 1, d);
56         tree[i] = merge(e-1, d-1);
57     }
58 }
59 int query(int p, int q, int k, int i=1, int st=1, int end=N) {
60     if (st == end) return sorted[st-1].val;
61     int left = (p!=1 ? tree[i][p-2] : 0);
62     int right = tree[i][q-1];
63     int diff = right - left;
64     if (diff >= k)
65         return query(left+1, right, k, 2*i, st, st+(end-st)/2);
66     else
67         return query(p-left, q-right, k-diff, 2*i+1, st+(end-st)/2+1, end);
68 }
69 int main() {
70     scanf("%d",&N);
71     for (int i = 0; i < N; i++) {
72         scanf("%d", &sorted[i].val);
73         sorted[i].pos = i;
74     }
75     sort(sorted, sorted+N);
76     memcpy(arr,sorted,sizeof(sorted));
77     create_tree();
78     printf("%d\n",query(1,N,3));
79 }

```


Capítulo V

Dynammmic 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 possíveis 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.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 //template
5
6 const int N = 100010;
7
8 vector<pii> obj;
9 int dp[N], vh[N], vs[N], vk[N];
10 void test()
11 {
12     int n, x;
13     cin >> n >> x;
14
15     rep(i, 0, n)
16         cin >> vh[i];
17     rep(i, 0, n)
18         cin >> vs[i];
19     rep(i, 0, n)
20         cin >> vk[i];
21     rep(i, 0, n)
22     {
23         int h = vh[i], s = vs[i], k = vk[i];
24         int sum = 1, pg = s, cost = h;
25         while(sum <= k)
26         {
27             k-=sum;
28             obj.eb(pg, cost);
29             pg+=pg;
30             cost+=cost;
31             sum+=sum;
32         }
33         if(k != 0)
34         {
35             obj.eb(s*k, h*k);
36         }
37     }
38     int mx = 0;
39     dp[0] = 0;
```

```

40     for(auto [p,c]:obj)
41     {
42         // cout << p << ' ' << c << '\n';
43         rep(i, mx+1, 0)
44         {
45             if(i+c <= x)
46                 dp[i+c] = max(dp[i]+p,dp[i+c]);
47         }
48         mx+=c;
49         mx = min(mx, x);
50     }
51     int rs = 0;
52     rep(i, 0, x+1)
53         rs = max(rs, dp[i]);
54     cout << rs << '\n';
55 }
56
57
58 int32_t main()
59 {
60     ios::sync_with_stdio(false);
61     cin.tie(NULL);
62     int t = 1;
63     // cin >> t;
64     while(t--)
65         test();
66     return 0;
67 }

```

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.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 //template
5
6 const int N = 100100;
7
8 vector<int> adj[N];
9 bitset<N> bt;
10 int vis[N], cont[N];
11 int dfs(int v)
12 {
13     vis[v] = 1;
14     int qt = 1;
15     for(int u: adj[v])
16     {
17         if(vis[u])
18             continue;
19         qt+=dfs(u);
20     }
21     return qt;
22 }
23 vector<int> movs;
24 void test()
25 {
26     int n, m;
27     cin >> n >> m;
28     while(m--)
29     {
30         int a, b;
31         cin >> a >> b;

```

```

32     adj[a].pb(b);
33     adj[b].pb(a);
34 }
35 rep(i, 1, n+1)
36     if(!vis[i])
37         cont[dfs(i)]++;
38
39 rep(i, 1, n+1)
40 {
41     int s = i, tot = cont[i]*i;
42     while(tot > 0)
43     {
44         movs.pb(min(tot,s));
45         tot-=s;
46         s+=s;
47     }
48 }
49 for(int x:movs)
50 {
51     bt |= bt<<x;
52     bt[x] = 1;
53 }
54 rep(i, 1, n+1)
55 {
56     cout << bt[i];
57 }
58 cout << '\n';
59 }
60
61
62 int32_t main()
63 {
64     ios::sync_with_stdio(false);
65     cin.tie(NULL);
66     int t = 1;
67     // cin >> t;
68     while(t--)
69         test();
70     return 0;
71 }

```

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 $O(\sum_0^n \binom{n}{k} \cdot 2^{n-k} = 3^n)$.

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 $SOS(mask, x)$ 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, $SOS(1001001, 3)$ includes the subsets 1001001, 1000001, 1001000, 1000000 which all have the same common prefix of 100.

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

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 greatest element in n .

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 //template
4
5 const int MAXN = 2e5+100;
6 const int MAXMASK = (1<<20)+100;
7 const int MAXB = 21;
8
9 int X[MAXN];
10
11 int dp[MAXMASK][MAXB];
12 int custo[MAXMASK];
13 int resp[MAXN][3];
14
15 void sosdp(){
16     rep(mask, 0, MAXMASK){
17         dp[mask][MAXB-1] = custo[mask];
18         for (int i = MAXB-2; i >= 0; i--){
19             // rep (i, MAXB-1, 0){
20                 dp[mask][i] = dp[mask][i+1];
21                 if (mask&(1 << i)){
22                     dp[mask][i] += dp[mask - (1 << i)][i+1];
23                 }
24             }
25         }
26     }
27
28 void test()
29 {
30     int n;
31     cin >> n;
32     rep(i, 0, n){
33         cin >> X[i];
34         custo[X[i]]++;
35     }
36     sosdp();
37     rep(i, 0, n){
38         resp[i][0] = dp[X[i]][0];
39         int mask = X[i]^((1<<(MAXB-1))-1);
40         resp[i][2] = dp[mask][0];
41         custo[X[i]]--;
42         custo[mask]++;
43         // cout << mask << endl;
44     }
45     sosdp();
46     rep(i, 0, n){
47         int mask = X[i]^((1<<(MAXB-1))-1);
48         resp[i][1] = dp[mask][0];
49     }
50     rep(i, 0, n){
51         cout << resp[i][0] << " " << resp[i][1] << " " << n-resp[i][2]
52         << "\n";
53     }
54 }
55 int32_t main()
56 {
57     ios::sync_with_stdio(false);
58     cin.tie(NULL);
59     int t = 1;
60     // cin >> t;
61     while(t--){
62         test();
63     }
64     return 0;

```

64 }

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 $1, 2, \dots, n$ 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.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 //template
5
6 const int N = 510;
7 const ll mod = 1000000007LL;
8
9 ll dp[2][N*N];
10
11 int main()
12 {
13     ios::sync_with_stdio(false);
14     cin.tie(NULL);
15     int n, k;
16
17     cin >> n >> k;
18
19     dp[0][0] = 1;
20     rep(j, 1, k + 1)
21         dp[0][j] = 1;
22
23     rep(i, 1, n+1)
24     {
25         rep(j, 0, k + 1)
26         {
27             int l = j-i;
28             dp[1][j] = dp[0][j];
29             if(l >= 0)
30                 dp[1][j] -= dp[0][l];
31             if(dp[1][j] < 0)
32                 dp[1][j] += mod;
33         }
34         // error(i);
35         // error(dp[1][k]);
36         dp[0][0] = dp[1][0];
37         rep(j, 1, k + 1){
38             dp[0][j] = dp[1][j] + dp[0][j-1];
39             // cout << dp[0][j] << '\n';
40             if(dp[0][j] >= mod)
41                 dp[0][j] -= mod;
42         }
43     }
44     cout << dp[1][k] << '\n';
45     return 0;
46 }
47 // 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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //template
5
6 const int N = 510;
7 const ll mod = 1000000007LL;
8
9 ll dp[2][N*N];
10
11 int main()
12 {
13     ios::sync_with_stdio(false);
14     cin.tie(NULL);
15     int n, k;
16
17     cin >> n >> k;
18
19     dp[0][0] = 1;
20     rep(j, 1, k + 1)
21         dp[0][j] = 1;
22
23     rep(i, 1, n+1)
24     {
25         rep(j, 0, k + 1)
26         {
27             int l = j-i;
28             dp[1][j] = dp[0][j];
29             if(l >= 0)
30                 dp[1][j] -= dp[0][l];
31             if(dp[1][j] < 0)
32                 dp[1][j] += mod;
33         }
34         // error(i);
35         // error(dp[1][k]);
36         dp[0][0] = dp[1][0];
37         rep(j, 1, k + 1){
38             dp[0][j] = dp[1][j] + dp[0][j-1];
39             // cout << dp[0][j] << '\n';
40             if(dp[0][j] >= mod)
41                 dp[0][j] -= mod;
42         }
43     }
44     cout << dp[1][k] << '\n';
45     return 0;
46 }
47 // 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 $\mathcal{O}(n \log(n))$ w/ line container
Divide and Conquer Optimization	$dp[i][j] = \min_{k < j} \{dp[i - 1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j + 1]$	$\mathcal{O}(kn^2)$	$\mathcal{O}(kn \log(n))$
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

```
1 //Optimization dp[i][j] = min{dp[i-1][k]+C[k][j]}
2 //O(n * k^2) -> O(n * k * log k)
3
4
5
6 int m, n;
7 vector<long long> dp_before(n), dp_cur(n);
8
9 long long C(int i, int j);
10
11 // compute dp_cur[l], ... dp_cur[r] (inclusive)
12 void compute(int l, int r, int optl, int optr) {
13     if (l > r)
14         return;
15
16     int mid = (l + r) >> 1;
17     pair<long long, int> best = {LLONG_MAX, -1};
18
19     for (int k = optl; k <= min(mid, optr); k++) {
20         best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
21     }
22
23     dp_cur[mid] = best.first;
24     int opt = best.second;
25
26     compute(l, mid - 1, optl, opt);
27     compute(mid + 1, r, opt, optr);
28 }
29
30 int solve() {
31     for (int i = 0; i < n; i++)
32         dp_before[i] = C(0, i);
33
34     for (int i = 1; i < m; i++) {
35         compute(0, n - 1, 0, n - 1);
36         dp_before = dp_cur;
37     }
38
39     return dp_before[n - 1];
40 }
```

5.5.2 Knuth Optimization

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 4005;
4 int dp[MAXN][MAXN];
5 int opt[MAXN][MAXN];
6 int sum[MAXN][MAXN];
7
8 int cost(int i, int j)
9 {
10     return sum[j][j] - sum[i][j] - sum[j][i] + sum[i][i];
11 }
12 int main()
13 {
14     int n,k;
15     n = get();
16     k = get();
17     for(int i = 1; i <= n ; i++)
18     {
19         for(int j = 1; j <= n ; j++)
20         {
21             sum[i][j] = get();
22             sum[i][j] += sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1];
23         }
24     }
25     for(int i = 1; i <= n ; i++)
26     {
27         dp[i][1] = cost(0, i);
28         opt[1][i] = 1;
29     }
30     int aux = 0;
31     for(int i = 2; i <= k ; i++)
32     {
33         for(int j = n; j >= 1; j--)
34         {
35             dp[j][i] = INT_MAX;
36             opt[n+1][i] = n;
37             for(int l = opt[j][i-1]; l <= opt[j+1][i]; l++)
38             {
39                 aux = cost(l,j);
40                 if( dp[j][i] > dp[l][i-1] + aux)
41                 {
42                     dp[j][i] = dp[l][i-1] + aux;
43                     opt[j][i] = l;
44                 }
45             }
46         }
47     }
48     printf("%d\n", dp[n][k] >> 1);
49
50     return 0;
51 }

```

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)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct line {

```



```

6      long long m, c;
7      long long eval(long long x) { return m * x + c; }
8      long double intersectX(line l) { return (long double)
9      (c - l.c) / (l.m - m); }
10 };
11
12 const int N = int(2e5);
13 typedef long long    ll;
14 ll a[N], b[N];
15
16 int main(){
17     int n;
18
19     scanf("%d", &n);
20
21     for(int i = 1 ; i <= n ; i++){
22         scanf("%lld", &a[i]);
23         // a[i] *= -1;
24     }
25
26     for(int i = 1 ; i <= n ; i++){
27         scanf("%lld", &b[i]);
28     }
29
30     deque<line> dq;
31     dq.push_front({b[1], 0LL}); // caso base
32
33     ll ans = b[1];
34
35     for (int i = 2; i <= n; i++) {
36
37         while (dq.size() >= 2 && dq.back().eval(a[i]) > dq[dq.size() -
38 2].eval(a[i])) //Inverta caso queira o máximo
39             dq.pop_back();
40
41         ll f = dq.back().eval(a[i]);
42
43         if(i == n){
44             ans = f;
45         }
46
47         line cur = {b[i], f};
48         while (dq.size() >= 2 && cur.intersectX(dq[0]) <= dq[0].
49 intersectX(dq[1]))
50             dq.pop_front();
51         dq.push_front(cur);
52
53     }
54
55     printf("%lld\n", ans);
56
57     return 0;
58 }

```

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:

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

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)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct line {
6     long long m, c;
7     long long eval(long long x) { return m * x + c; }
8     long double intersectX(line l) { return (long double) (c - l.c) / (l
9     .m - m); }
10 };
11
12 const int N = int(1e5 + 100);
13 const int K = 20;
14 typedef long long    ll;
15 ll x[N], w[N];
16 ll dp[N][K];
17 ll pref[N], g[N];
18
19 int main(){
20     int n, k;
21
22     while(scanf("%d %d", &n, &k) != EOF){
23         for(int i = 1 ; i <= n ; i++){
24             scanf("%lld", &w[i]);
25             x[i] = i;
26
27             // a[i] *= -1;
28         }
29
30         reverse(w + 1, w + 1 + n);
31
32         for(int i = 1 ; i <= n ; i++){
33             pref[i] = pref[i - 1] + w[i];
34             g[i] = w[i] * x[i] + g[i - 1];
35         }
36
37         for(int i = 1 ; i <= n ; i++){
38             dp[i][1] = x[i] * pref[i] - g[i];
39         }
40
41         deque<line> dq;
42
43         for(int j = 2 ; j <= k ; j++){
44             dq.push_front({0LL, 0LL});
45
46             for (int i = 1; i <= n; i++) {
47                 while (dq.size() >= 2 && dq.back().eval(x[i]) > dq[dq.size()
48                 - 2].eval(x[i])){ //Inverta caso queira o maximo
49                     dq.pop_back();
50                 }
51
52                 ll f = dq.back().eval(x[i]);
53                 dp[i][j] = x[i] * pref[i] - g[i] + f;
54                 line cur = {-pref[i], dp[i][j - 1] + g[i]};
55
56                 while (dq.size() >= 2 && cur.intersectX(dq[0]) <= dq[0].
57                 intersectX(dq[1]))
58                     dq.pop_front();
59
60                 dq.push_front(cur);
61             }
62
63             dq.clear();

```

```

61     }
62
63     printf("%lld\n", dp[n][k]);
64 }
65
66     return 0;
67 }

```

Convex Hull Trick 3 (Online Queries)

This solutions works without any assumption about the coefficients, in this case we will query is answered in $O(\log n)$.

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

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 typedef long long ll;
6
7 const int N = int(1e5 + 10);
8 int subtree[N];
9 ll a[N], b[N];
10 ll dp[N];
11 bool is_leaf[N];
12 vector< int > adj[N];
13
14 template <class TT = ll>
15 struct Line{
16     // coef angular, linear, criterio de comparacao
17     mutable TT k,m,p;
18     // aqui eu quero sempre deixar os com menor coef angular mais pra
19     // frente, conc pra baixo
20     bool operator <(const Line & o) const{
21         return k>o.k;
22     }
23     bool operator <(const TT o) const{
24         return p<o;
25     }
26 };
27 struct CHT : multiset<Line,less<>>{
28     static const TT inf = std::numerical_limit<TT>::max();
29     TT div(TT a, TT b){
30         return a/b-((a^b) < 0 && a%b);
31     }
32
33     // x eh melhor que y?
34     bool isect(iterator x, iterator y){
35         if(y == end()){
36             x->p=inf;
37             return false;
38         }
39         if(x->k == y->k)
40             x->p= x->m <= y->m ? inf : -inf;
41         else
42             x->p=div(y->m-x->m,x->k-y->k);
43         return x->p >= y->p;
44     }
45
46     void add(TT k, TT m){
47         auto z= insert({k,m,0});
48         auto y=z++;
49         auto x=y;
50         while(insect(y,z)) z=erase(z);
51         if(x != begin() && insect(--x,y))
52             insect(x,y=erase(y));
53     }
54 };

```

```

51         while((y=x) != begin() && (--x)->p >= y->p)
52             insect(x,erase(y));
53     }
54     TT query(TT x){
55     assert(!empty());
56         auto ans=lower_bound(x);
57         return ans->k*x+ans->m;
58     }
59 };
60
61 int dfs(int u, int ft){
62     is_leaf[u] = 1;
63
64     for(auto v: adj[u]){
65         if(v == ft){
66             continue;
67         }
68
69         is_leaf[u] = 0;
70     }
71
72     if(is_leaf[u]){
73         return subtree[u] = 1;
74     }
75
76     subtree[u] = 1;
77
78     for(auto v: adj[u]){
79         if(v == ft){
80             continue;
81         }
82
83         subtree[u] += dfs(v, u);
84     }
85
86     return subtree[u];
87 }
88
89 void dfs1(int u, int ft, CHT &cur){
90     if(is_leaf[u]){
91         dp[u] = 0LL;
92         cur.add(-b[u], -dp[u]);
93
94         return;
95     }
96
97     int mx = 0;
98     int big_son = 0;
99
100    for(auto v: adj[u]){
101        if(v == ft){
102            continue;
103        }
104
105        if(subtree[v] > mx){
106            big_son = v;
107            mx = subtree[v];
108        }
109    }
110
111    dfs1(big_son, u, cur);
112
113    for(auto v: adj[u]){
114        if(v == big_son || v == ft){
115            continue;
116        }

```

```

117
118     LineContainer tmp;
119     dfs1(v, u, tmp);
120
121     for(auto d: tmp){
122         //printf("aqui!\n");
123         cur.add(d.k, d.m);
124     }
125 }
126
127 dp[u] = -cur.query(a[u]);
128 cur.add(-b[u], -dp[u]);
129 }
130
131 int main(){
132     int n;
133
134     scanf("%d", &n);
135
136     for(int i = 1 ; i <= n ; i++){
137         scanf("%lld", &a[i]);
138     }
139
140     for(int i = 1 ; i <= n ; i++){
141         scanf("%lld", &b[i]);
142     }
143
144     for(int i = 1 ; i < n ; i++){
145         int ui, vi;
146
147         scanf("%d %d", &ui, &vi);
148
149         adj[ui].push_back(vi);
150         adj[vi].push_back(ui);
151     }
152
153     CHT cur;
154     dfs(1, 1);
155     dfs1(1, 1, cur);
156
157     for(int i = 1 ; i <= n ; i++){
158         printf("%lld ", dp[i]);
159     }
160
161     printf("\n");
162
163     return 0;
164 }

```

5.6 Steiner Tree DP

MST for a subset

```

1 #include <iostream>
2 using namespace std;
3 //5717 - Peach Blossom Spring
4 #define INF 1000000000
5
6 int dp[1<<10][64];
7
8 bool check(int m, int K) {
9     int i, tmp1=0, tmp2=0;
10    for (i=0; i<K; i++) tmp1 += ((m>>i)&1);
11    for (i=0; i<K; i++) tmp2 += ((m>>(K+i))&1);
12    return tmp1 == tmp2;

```

```

13 }
14
15 int main() {
16     int N, M, K, i, j, k, t, T, a, b, c, d[64][64], RES, tmp;
17
18     cin >> T;
19     for (t=0; t<T; t++) {
20         cin >> N >> M >> K;
21
22         for (i=0; i<N; i++) {
23             d[i][i] = 0;
24             for (j=i+1; j<N; j++) d[i][j] = d[j][i] = INF;
25         }
26
27         for (i=0; i<M; i++) {
28             cin >> a >> b >> c; a--; b--;
29             d[a][b] = d[b][a] = min(c, min(d[a][b], d[b][a]));
30         }
31
32         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]);
33
34         for (i=0; i<(1<<(2*K)); i++) for (j=0; j<N; j++) dp[i][j] = INF;
35         for (i=0; i<K; i++) for (j=0; j<N; j++) dp[1<<i][j] = d[i][j];
36         for (i=0; i<K; i++) for (j=0; j<N; j++) dp[1<<(K+i)][j] = d[N-1-
i][j];
37
38         for (i=1; i<(1<<(2*K)); i++) {
39             for (j=0; j<N; j++) {
40                 if (dp[i][j] != INF) continue;
41
42                 for (k=(i-1)&i; k; k=(k-1)&i) dp[i][j] = min(dp[i][j],
dp[k][j]+dp[i~k][j]);
43             }
44
45             if (check(i, K)) {
46                 tmp = INF;
47                 for (j=0; j<N; j++) tmp = min(tmp, dp[i][j]);
48                 for (j=0; j<N; j++) dp[i][j] = tmp;
49             } else {
50                 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]);
51             }
52         }
53
54         RES = INF;
55         for (i=0; i<N; i++) RES = min(RES, dp[(1<<(2*K))-1][i]);
56         if (RES < INF) cout << RES << endl;
57         else cout << "No solution" << endl;
58     }
59
60     return 0;
61 }

```

Capítulo VI

String

6.1 String Hash

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

6.1.1 Simple Hash

```
1 // String Hash - Polynomial rolling hash
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 //template
6
7 const int N = 200100;
8 const uint64_t p=33, mod=1000000007, p2=73, mod2=1000000009;
9 uint64_t ppow[N], ppow2[N];
10
11 void build()
12 {
13     ppow[0] = 1;
14     ppow2[0] = 1;
15     for (int i = 1; i < N; i++) {
16         ppow[i] = (ppow[i-1] * p) % mod;
17         ppow2[i] = (ppow2[i-1] * p2) % mod2;
18     }
19 }
20 struct Hash {
21     vector<pair<uint64_t, uint64_t>> h;
22
23     Hash () {}
24
25     void init(string &s) {
26         h.resize(s.size()+2);
27         h[0] = {5389ULL, 5389ULL};
28         for (size_t i = 0; i < s.size(); i++) {
29             int code = s[i];
30             h[i+1].first = (h[i].first * p + code) % mod;
31             h[i+1].second = (h[i].second * p2 + code) % mod2;
32         }
33     }
34
35     pair<uint64_t, uint64_t> get_hash(int i, int j) {
36         pair<uint64_t, uint64_t> r;
37         r.first = (h[j+1].first - (h[i].first * ppow[j-i+1]) % mod + mod
38 ) % mod;
39         r.second = (h[j+1].second - (h[i].second * ppow2[j-i+1]) % mod2
40 + mod2) % mod2;
41         return r;
42     }
43 };
44
45 uint64_t fastExp(uint64_t a, uint64_t p, uint64_t mod) {
```

```

44     return p == 0 ? 1 : ((fastExp((a * a) % mod, p / 2, mod) % mod) * (p &
45         1 ? a%mod : 1)) % mod;
46 }
47 pair<uint64_t, uint64_t> repeatHash(pair<uint64_t, uint64_t> r, int size
48     , int qtd){
49     uint64_t q = ppow[size], q2 = ppow2[size];
50     uint64_t num = fastExp(q, qtd, mod) - 1, num2 = fastExp(q2, qtd,
51         mod2) - 1;
52     uint64_t den = fastExp((q - 1) , mod - 2, mod), den2 = fastExp((q2
53         - 1), mod2 - 2, mod2);
54     return mp((num * den % mod) * r.ft % mod, (num2 * den2 % mod2) * r.
55         sd % mod2);
56 }
57
58 pair<uint64_t, uint64_t> getQtdCharsInACiclicSubstring(int l, int r, int
59     qtd, Hash &hat){
60     int size = r - l + 1;
61     if (qtd <= size){
62         return hat.get_hash(l, l+qtd-1);
63     }
64     pair<uint64_t, uint64_t> h = hat.get_hash(l, r);
65     int integerPart = qtd / size;
66     int rest = qtd - size * integerPart;
67     pair<uint64_t, uint64_t> repeatedPart = repeatHash(h, size,
68         integerPart, hat);
69     pair<uint64_t, uint64_t> restPart = (rest == 0 ? mp((uint64_t)0, (
70         uint64_t)0) : hat.get_hash(l, l+rest-1));
71     return mp(
72         (restPart.ft + repeatedPart.ft * ppow[rest] % mod) % mod,
73         (restPart.sd + repeatedPart.sd * ppow2[rest] % mod2) % mod2
74     );
75 }
76
77 */

```

6.1.2 Hash 2D_{ayllon}

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

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

```

1  const int N = 2010;
2  const uint64_t p[4] = {33, 73, 37, 93}, mod[2] = {1000000007,
3      1000000009};
4  uint64_t ppow[4][N];
5
6  void build()
7  {
8      for (int j = 0; j < 4; j++){
9          ppow[j][0] = 1;
10         for (int i = 1; i < N; i++) {
11             ppow[j][i] = (ppow[j][i-1] * p[j]) % mod[j>>1];
12         }
13     }
14 }
15 struct Hash {
16     int n, m;
17     vector<vector<pair<uint64_t, uint64_t>>>>h;
18
19     Hash () {}
20
21     void init(vector<string> &word2d) {
22         n = word2d.size();
23         m = word2d[0].size();

```



```

23     h.resize(n+1, vector(m+1, mp((uint64_t)0,(uint64_t)0)));
24
25     pair<uint64_t, uint64_t> sv;
26     for (int i = 1; i <= n; i++) {
27         sv = {0, 0};
28         for (int j = 1; j <= m; j++){
29             int code = word2d[i-1][j-1];
30             sv = {(sv.ft * p[0] + code)%mod[0],
31                 (sv.sd * p[2] + code)%mod[1]};
32             h[i][j] = {(h[i-1][j].ft * p[1])%mod[0] + sv.ft,
33                 (h[i-1][j].sd * p[3])%mod[1] + sv.sd};
34             if (h[i][j].ft >= mod[0]) h[i][j].ft -= mod[0];
35             if (h[i][j].sd >= mod[1]) h[i][j].sd -= mod[1];
36         }
37     }
38 }
39
40 pair<uint64_t, uint64_t> get_hash(int i, int j, int height, int
width) {
41     int i2 = i + height, j2 = j + width;
42     pair<uint64_t, uint64_t> r = {
43         (h[i2][j2].ft + mod[0] + mod[0]
44         - (h[i][j2].ft * ppow[1][i2-i])%mod[0]
45         - (h[i2][j].ft * ppow[0][j2-j])%mod[0]
46         + ((ppow[1][i2-i] * ppow[0][j2-j])%mod[0] * h[i][j].ft)%
mod[0]
47         )%mod[0],
48         (h[i2][j2].sd + mod[1] + mod[1]
49         - (h[i][j2].sd * ppow[3][i2-i])%mod[1]
50         - (h[i2][j].sd * ppow[2][j2-j])%mod[1]
51         + ((ppow[3][i2-i] * ppow[2][j2-j])%mod[1] * h[i][j].sd)%
mod[1]
52         )%mod[1]
53     };
54     return r;
55 }
56 };
57
58 Hash h1, h2;
59 vector<string> ss1, ss2;
60
61 void test()
62 {
63     int n,m,n2,m2;
64     string s;
65     cin >> n >> m;
66     rep(i, 0, n){
67         cin >> s;
68         ss1.pb(s);
69     }
70     cin >> n2 >> m2;
71     rep(i, 0, n2){
72         cin >> s;
73         ss2.pb(s);
74     }
75     build();
76     h1.init(ss1);
77     h2.init(ss2);
78
79     auto target = h1.get_hash(0, 0, n, m);
80     for(int i = 0; i < n2 - n + 1; i++){
81         for(int j = 0; j < m2 - m + 1; j++){
82             if (target == h2.get_hash(i, j, n, m)){
83                 cout << "MATCH " << i << " " << j << "\n";
84             }
85         }
86     }

```

```

86     }
87 }

```

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.

```

1 // Palindrome Queries - CSES Solution
2 // link: https://cses.fi/problemset/task/2420
3
4 #include <bits/stdc++.h>
5 using namespace std;
6
7 typedef long long ll;
8
9 #define all(x) x.begin(), x.end()
10 #define ft first
11 #define sd second
12 #define mp make_pair
13
14 const int N = 2e5+3;
15 const ll mod = 1e9+7, mod2 = 1e9+9;
16 const ll p = 33, p2 = 73;
17
18 struct updtHash{
19     vector<ll> ppow, ppow2;
20     vector<pair<ll,ll>> h;
21     updtHash(){
22         ppow.resize(N);
23         ppow2.resize(N);
24         ppow[0] = 1;
25         ppow2[0] = 1;
26         for (int i = 1; i < N; i++) {
27             ppow[i] = (ppow[i-1] * p) % mod;
28             ppow2[i] = (ppow2[i-1] * p2) % mod2;
29         }
30     }
31
32     ll inv(ll x, int w){
33         ll md = (w ? mod2 : mod);
34         ll b = md-2;
35         ll rs = 1;
36
37         while(b){
38             if(1&b) rs = rs*x%md;
39             x = x*x%md;
40             b >>= 1;
41         }
42         return rs;
43     }
44
45     void updt(int x, pair<ll,ll> v){
46         while(x < N){
47             h[x].ft += v.ft;
48             h[x].sd += v.sd;
49             if(h[x].ft > mod) h[x].ft -= mod;
50             if(h[x].sd > mod2) h[x].sd -= mod2;
51             x += (x&-x);
52         }
53     }
54
55     pair<ll,ll> get(int x){
56         pair<ll,ll> r = {0ll, 0ll};
57         while(x){

```

```

58         r.ft += h[x].ft;
59         r.sd += h[x].sd;
60         if(r.ft > mod) r.ft -= mod;
61         if(r.sd > mod2) r.sd -= mod2;
62         x -= (x&-x);
63     }
64
65     return r;
66 }
67
68 void init(string &s) {
69     h.assign(N, mp(0, 0));
70     for (int i = 0; i < s.size(); i++) {
71         int code = s[i];
72         pair<ll, ll> v;
73         v.first = code * ppow[i+1] % mod;
74         v.second = code * ppow2[i+1] % mod2;
75         updt(i+1, v);
76     }
77 }
78
79 pair<ll, ll> get_hash(int i, int j){
80     pair<ll, ll> r, l;
81     r = get(j+1);
82     l = get(i);
83     r.ft = (r.ft - l.ft + mod) * inv(ppow[i], 0) % mod;
84     r.sd = (r.sd - l.sd + mod2) * inv(ppow2[i], 1) % mod2;
85
86     return r;
87 }
88
89 void updt_char(int i, int code){
90     pair<ll, ll> l, r, v;
91     r = get(i+1);
92     l = get(i);
93     v.ft = (code * ppow[i+1] - r.ft + l.ft + mod) % mod;
94     v.sd = (code * ppow2[i+1] - r.sd + l.sd + mod2) % mod2;
95     updt(i+1, v);
96 }
97 };
98
99 updtHash h1, h2;
100
101 void test(){
102     int n, q;
103     cin >> n >> q;
104     string s;
105     cin >> s;
106
107     h1.init(s);
108     reverse(all(s));
109     h2.init(s);
110
111     while(q--){
112         int t;
113         cin >> t;
114         if(t == 1){
115             int x;
116             char c;
117             cin >> x >> c;
118             h1.updt_char(x-1, c);
119             h2.updt_char(n-x, c);
120         }
121         else{
122             int l, r;
123             cin >> l >> r;

```

```

124         pair<ll,ll> v1, v2;
125         v1 = h1.get_hash(l-1, r-1);
126         v2 = h2.get_hash(n-r, n-1);
127         if(v1 == v2){
128             cout<<"YES\n";
129         }
130         else{
131             cout<<"NO\n";
132         }
133     }
134 }
135 }
136
137 int32_t main(){
138     ios::sync_with_stdio(false);
139     cin.tie(nullptr);
140
141     int teste = 1;
142     // cin >> teste;
143
144     while(teste--){
145         test();
146     }
147
148     return 0;
149 }

```

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 $lcp(i, j) = \min_{k=i}^{j-1} lcp[k]$.

```

1 vector<int> sort_cyclic_shifts(string const& s) {
2     int n = s.size();
3     const int alphabet = 256;
4     vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
5     for (int i = 0; i < n; i++)
6         cnt[s[i]]++;
7     for (int i = 1; i < alphabet; i++)
8         cnt[i] += cnt[i-1];
9     for (int i = 0; i < n; i++)
10        p[--cnt[s[i]]] = i;
11    c[p[0]] = 0;
12    int classes = 1;
13    for (int i = 1; i < n; i++) {
14        if (s[p[i]] != s[p[i-1]])
15            classes++;
16        c[p[i]] = classes - 1;
17    }
18    vector<int> pn(n), cn(n);
19    for (int h = 0; (1 << h) < n; ++h) {
20        for (int i = 0; i < n; i++) {
21            pn[i] = p[i] - (1 << h);
22            if (pn[i] < 0)
23                pn[i] += n;
24        }
25        fill(cnt.begin(), cnt.begin() + classes, 0);
26        for (int i = 0; i < n; i++)
27            cnt[c[pn[i]]]++;
28        for (int i = 1; i < classes; i++)
29            cnt[i] += cnt[i-1];
30        for (int i = n-1; i >= 0; i--)
31            p[--cnt[c[pn[i]]]] = pn[i];
32        cn[p[0]] = 0;
33        classes = 1;

```

```

34     for (int i = 1; i < n; i++) {
35         pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
36         pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
37     };
38     if (cur != prev)
39         ++classes;
40     cn[p[i]] = classes - 1;
41 }
42 c.swap(cn);
43 return p;
44 }
45
46
47
48 vector<int> lcp_construction(string const& s, vector<int> const& p) {
49     int n = s.size();
50     vector<int> rank(n, 0);
51     for (int i = 0; i < n; i++)
52         rank[p[i]] = i;
53
54     int k = 0;
55     vector<int> lcp(n-1, 0);
56     for (int i = 0; i < n; i++) {
57         if (rank[i] == n - 1) {
58             k = 0;
59             continue;
60         }
61         int j = p[rank[i] + 1];
62         while (i + k < n && j + k < n && s[i+k] == s[j+k])
63             k++;
64         lcp[rank[i]] = k;
65         if (k)
66             k--;
67     }
68     return lcp;
69 }
70
71
72 void test()
73 {
74     string s;
75     cin >> s;
76     s.pb('#');
77     vector<int> sa = sort_cyclic_shifts(s);
78     vector<int> lcp = lcp_construction(s, sa);
79 }
80 }

```

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.

```

1  const int N = 200100;
2  struct state {
3      int len, link, qt = 0;
4      map<char, int> next;
5  };
6  state st[N * 2];
7  int sz, last;
8  void sa_init() {
9      st[0].len = 0;

```

```

10     st[0].link = -1;
11     sz++;
12     last = 0;
13 }
14 void sa_extend(char c) {
15     int cur = sz++;
16     st[cur].len = st[last].len + 1;
17     st[cur].qt = 1;
18     int p = last;
19     while (p != -1 && !st[p].next.count(c)) {
20         st[p].next[c] = cur;
21         p = st[p].link;
22     }
23     if (p == -1) {
24         st[cur].link = 0;
25     } else {
26         int q = st[p].next[c];
27         if (st[p].len + 1 == st[q].len) {
28             st[cur].link = q;
29         } else {
30             int clone = sz++;
31             st[clone].len = st[p].len + 1;
32             st[clone].next = st[q].next;
33             st[clone].link = st[q].link;
34             while (p != -1 && st[p].next[c] == q) {
35                 st[p].next[c] = clone;
36                 p = st[p].link;
37             }
38             st[q].link = st[cur].link = clone;
39         }
40     }
41     last = cur;
42 }

```

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 it's a DAG) then just do a walk. Easier version doesn't need to count repetitions, so just a path.

```

1 ll dp[N];
2 string bfs(ll k)
3 {
4     vector<pair<int,int>> ids;
5     for(int pos = 1; pos < sz; pos++)
6         ids.push_back({st[pos].len,pos});
7     sort(ids.begin(), ids.end());
8     reverse(ids.begin(), ids.end());
9     for(auto [l,x]:ids){
10         dp[x] = st[x].qt;
11         // cout << x << ' ' << st[x].qt << '\n';
12         for(auto [c,y]:st[x].next)
13         {
14             dp[x] += dp[y];
15         }
16         if(st[x].link != -1){
17             // cout << x << " " << st[x].link << " " << link\n";
18             st[st[x].link].qt += st[x].qt;
19         }
20     }
21     reverse(ids.begin(), ids.end());
22     int pos = 0;
23     string resp;
24     while(k>0){
25         for(auto [c,y]:st[pos].next)
26         {
27             if(dp[y] < k){

```

```

28         // cout << c << ' ' << dp[y] << ' ' << y << "\n";
29         k -= dp[y];
30     }
31     else{
32         k -= st[y].qt;
33         // cout << c << ' ' << dp[y] << " " << y << " << go\n";
34         resp.push_back(c);
35         pos = y;
36         break;
37     }
38 }
39 }
40 return resp;
41 }

```

Example 2:

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

```

1 int getlen(int x)
2 {
3     if(x <= 0)
4         return 0;
5     return st[x].len;
6 }
7 int rs[N];
8 void bfs(int n)
9 {
10     vector<pair<int,int>> ids;
11     for(int pos = 0; pos < sz; pos++){
12         rs[getlen(st[pos].link)+1]++;
13         rs[getlen(pos)+1]--;
14     }
15     for(int i = 1; i <= n; i++){
16         rs[i] += rs[i-1];
17         cout << rs[i] << ' ';
18     }
19     cout << '\n';
20 }

```

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.

```

1 int best = 0, dep[N], pai[N];
2 string bfs()
3 {
4     vector<pair<int,int>> ids;
5     for(int pos = 1; pos < sz; pos++)
6         ids.push_back({st[pos].len, pos});
7     sort(ids.begin(), ids.end());
8     reverse(ids.begin(), ids.end());
9     for(auto [l,x]:ids){
10         if(st[x].link != -1){
11             // cout << x << " " << st[x].link << " << link\n";
12             st[st[x].link].qt += st[x].qt;
13         }
14     }
15     reverse(ids.begin(), ids.end());
16     int at = -1;
17     for(auto [l,pos]:ids){
18         if(st[pos].qt < 2)
19             continue;
20         if(dep[pos] == 0){
21             dep[pos] = 1;
22         }
23         if(dep[pos] > best){
24             best = dep[pos];

```

```

25         at = pos;
26     }
27     for(auto [x,y]:st[pos].next)
28     {
29         if(st[y].qt>1){
30             if(dep[y] < dep[pos]+1)
31             {
32                 dep[y] = dep[pos]+1;
33                 pai[y] = pos;
34             }
35             dep[y] = max(dep[y],dep[pos]+1);
36         }
37     }
38 }
39 if(at == -1)
40     return "-1";
41 string resp;
42 while(at)
43 {
44     for(auto [x,y]:st[pai[at]].next)
45     {
46         if(y == at){
47             resp.push_back(x);
48         }
49     }
50     at = pai[at];
51 }
52 reverse(all(resp));
53 return resp;
54
55 }

```

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).

```

1  const int NS =200100 ;
2  const int N =  NS*2;
3
4  int cn, cd, ns, en = 1, lst;
5  vector<int>/*string*/ S[NS]; int si = -1;
6
7
8  vector<int> sufn[N];
9
10 struct node {
11     int l, r, si;
12     int p, suf;
13     //map<char, int> adj;
14     map<int, int> adj;
15     node() : l(0), r(-1), suf(0), p(0) {}
16     node(int L, int R, int S, int P) : l(L), r(R), si(S), p(P) {}
17     inline int len() { return r - l + 1; }
18     inline int operator[](int i) { return S[si][l + i]; }
19     inline int& operator()(int c) { return adj[c]; }
20 } t[N];
21
22 inline int new_node(int L, int R, int S, int P) {
23     t[en] = node(L, R, S, P);
24     return en++;
25 }
26
27 void add_string(vector<int>/*string*/ &s) {

```



```

28 //s += '$';
29 S[++si] = s;
30 sufn[si].resize(s.size() + 1);
31 cn = cd = 0;
32 int i = 0; const int n = s.size();
33 for(int j = 0; j < n; j++)
34     for(; i <= j; i++) {
35         if(cd == t[cn].len() && t[cn](s[j]))
36             cn = t[cn](s[j]), cd = 0;
37         if(cd < t[cn].len() && t[cn][cd] == s[j]) {
38             cd++;
39             if(j < s.size() - 1) break;
40             else {
41                 if(i) t[lst].suf = cn;
42                 for(; i <= j; i++) {
43                     sufn[si][i] = cn;
44                     cn = t[cn].suf;
45                 }
46             }
47         } else if(cd == t[cn].len()) {
48             sufn[si][i] = en;
49             if(i) t[lst].suf = en; lst = en;
50             t[cn](s[j]) = new_node(j, n - 1, si, cn);
51             cn = t[cn].suf;
52             cd = t[cn].len();
53         } else {
54             int mid = new_node(t[cn].l, t[cn].l + cd - 1, t[cn].si, t[cn].p)
55             ;
56             t[t[cn].p](t[cn][0]) = mid;
57             if(ns) t[ns].suf = mid;
58             if(i) t[lst].suf = en; lst = en;
59             sufn[si][i] = en;
60             t[mid](s[j]) = new_node(j, n - 1, si, mid);
61             t[mid](t[cn][cd]) = cn;
62             t[cn].p = mid; t[cn].l += cd;
63             cn = t[mid].p;
64             int g = cn? j - cd : i + 1;
65             cn = t[cn].suf;
66             while(g < j && g + t[t[cn](S[si][g])].len() <= j)
67                 cn = t[cn](S[si][g]), g += t[cn].len();
68             if(g == j)
69                 ns = 0, t[mid].suf = cn, cd = t[cn].len();
70             else
71                 ns = mid, cn = t[cn](S[si][g]), cd = j - g;
72         }
73     }

```

6.5 Palindromic Tree

```

1 //UVALive - 6072
2 //http://adilet.org/blog/palindromic-tree/
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 const int N=200010;
7 typedef long long ll;
8
9 struct node {
10     int next[26];
11     int len;
12     int sufflink;
13     int num;
14     int lazy;

```

```

15 };
16
17 string s;
18 node tree[N];
19 int num;           // node 1 - root with len -1, node 2 - root with len
20                    // 0
21 int suff;          // max suffix palindrome
22
23 bool addLetter(int pos) {
24     int cur = suff, curlen = 0;
25     int let = s[pos] - 'a';
26
27     while (true) {
28         curlen = tree[cur].len;
29         if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
30             break;
31         cur = tree[cur].sufflink;
32     }
33     if (tree[cur].next[let]) {
34         suff = tree[cur].next[let];
35         tree[suff].lazy++;
36         return false;
37     }
38
39     num++;
40     suff = num;
41     tree[num].len = tree[cur].len + 2;
42     tree[cur].next[let] = num;
43     tree[num].lazy=1;
44     // cout<<num<<" "<<let<<"\n";
45     if (tree[num].len == 1) {
46         tree[num].sufflink = 2;
47         tree[num].num = 1;
48         return true;
49     }
50
51     while (true) {
52         cur = tree[cur].sufflink;
53         curlen = tree[cur].len;
54         if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
55             tree[num].sufflink = tree[cur].next[let];
56             break;
57         }
58     }
59
60     tree[num].num = 1 + tree[tree[num].sufflink].num;
61
62     return true;
63 }
64
65 void initTree() {
66     num = 2; suff = 2;
67     tree[1].len = -1; tree[1].sufflink = 1;
68     tree[2].len = 0; tree[2].sufflink = 1;
69 }
70 ll value[26], pot[N];
71 vector<pair<ll, ll>> pl;
72 const ll mod=777777777LL;
73 void dfs(int t)
74 {
75     for (int i = 0; i < 26; ++i)
76     {
77         // cout<<t<<" "<<tree[t].next[i]<<'\n';
78         if (tree[t].next[i])
79         {
80             dfs(tree[t].next[i]);

```

```

80     }
81 }
82 }
83 void dfs1(int t,ll at,int pt)
84 {
85     // cout<<t<<' '<<at<<" "<<tree[t].lazy<<'\\n';
86     if(t>2)
87         pl.push_back({at,tree[t].lazy});
88     for (int i = 0; i < 26; ++i)
89     {
90         if(tree[t].next[i])
91         {
92             // cout<<t<<" --- "<<(char)(i+'a')<<" --> "<<tree[t].next[i
93             ]<<"\\n";
94             dfs1(tree[t].next[i],(at+value[i]*pot[pt])%mod,pt+1);
95         }
96     }
97 }
98 int main() {
99     ios::sync_with_stdio(false);
100     cin.tie(NULL);
101     pot[0]=1;
102     for (int i = 1; i < N; ++i)
103     {
104         pot[i] = (pot[i-1]*26LL)%mod;
105     }
106     int q;
107     cin>>q;
108     while(q--)
109     {
110
111         int n,m;
112         cin>>n>>m;
113         cin>>s;
114
115         initTree();
116
117         for (int i = 0; i < n; i++) {
118             addLetter(i);
119         }
120         for (int i = num; i >0 ; --i)
121         {
122             if(tree[i].sufflink)
123                 tree[tree[i].sufflink].lazy+=tree[i].lazy;
124         }
125         // cout<<num<<" "<<s<<'\\n';
126         // dfs(1);
127         for (int i = 0; i < m; ++i)
128         {
129             ll k;
130             cin>>k;
131             for (int j = 0; j < 26; ++j)
132             {
133                 cin>>value[j];
134                 value[j]%=mod;
135             }
136             pl.clear();
137             dfs1(1,0,0);
138             dfs1(2,0,0);
139             sort(pl.begin(),pl.end());
140             ll now=0;
141             int j=0;
142             while(j<pl.size() && now+pl[j].second<k)
143             {
144                 now+=pl[j++].second;

```

```

145         }
146         cout<<pl[j].first<<"\n";
147     }
148     cout<<"\n";
149     for (int i = 0; i <= num; ++i)
150     {
151         tree[i].len=tree[i].sufflink=tree[i].num=tree[i].lazy=0;
152         for (int j = 0; j < 26; ++j)
153         {
154             tree[i].next[j]=0;
155         }
156     }
157 }
158
159 return 0;
160 }

```

6.6 Minimal Rotation

Find the minimal cyclic shift of an string in $O(N)$.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define all(x) x.begin(), x.end()
5 #define lef(x) (x<<1)
6 #define rig(x) (lef(x)|1)
7 int main()
8 {
9     ios::sync_with_stdio(false);
10    cin.tie(NULL);
11    string s;
12    cin >> s;
13    s += s;
14    int n = s.size();
15    int i = 0, ans = 0;
16    while (i < n / 2) {
17        ans = i;
18        int j = i + 1, k = i;
19        while (j < n && s[k] <= s[j]) {
20            if (s[k] < s[j])
21                k = i;
22            else
23                k++;
24            j++;
25        }
26        while (i <= k)
27            i += j - k;
28    }
29    cout << s.substr(ans, n / 2) << '\n';
30    return 0;
31 }

```

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).

```

1 const int K = 26;
2 struct node {
3     int fim = 0,p,link = 0,elink = -1,len;
4     int go[K] = {0},nxt[K] = {0};
5     node(int p=0,int len = 0) : p(p),len(len) {};
6 };
7 vector<node> t(1);

```

```

8 void add_string(string const& s) {
9     int v = 0;
10    for (char ch : s) {
11        ch-='a';
12        if (!t[v].nxt[ch]) {
13            t[v].nxt[ch] = t.size();
14            t.eb(v,t[v].len+1);
15        }
16        v = t[v].nxt[ch];
17    }
18    // maybe change this to a vector with some ids
19    t[v].fim = 1;
20 }
21
22 void calc_link()
23 {
24     queue<int> q;
25     q.push(0);
26     while(!q.empty())
27     {
28         int v = q.front();
29         q.pop();
30         int lk = t[v].link;
31         rep(i, 0, K)
32         {
33             if(!t[v].nxt[i])
34                 t[v].go[i] = t[lk].go[i];
35             else{
36                 t[v].go[i] = t[v].nxt[i];
37                 t[t[v].go[i]].link = (!v?0:t[lk].go[i]);
38                 q.push(t[v].nxt[i]);
39             }
40         }
41         //exit link is the first link with fim != 0
42         if(t[lk].fim)
43             t[v].elink = lk;
44         else
45             t[v].elink = t[lk].elink;
46     }
47 }

```

6.8 Trie

Tá safe

```

1 const int N = 1000500;
2 const int mod=1e9+7;
3 typedef long long ll;
4 struct node{
5     bool leaf;
6     map<int,int> prox;
7     node(){
8         leaf = false;
9     }
10 };
11 int id=0;
12 node nodes[N];
13 void insert_trie(string s){
14     int curr=0;
15     for(char c: s){
16         if(nodes[curr].prox[c-'a']==0)
17             nodes[curr].prox[c-'a']=++id;
18         curr=nodes[curr].prox[c-'a'];
19     }
20     nodes[curr].leaf=true;

```

```
21 }
22 bool find_trie(string s){
23     int curr=0;
24     for(char c: s){
25         if(!nodes[curr].prox[c-'a'])
26             return false;
27         curr=nodes[curr].prox[c-'a'];
28     }
29     if(nodes[curr].leaf) return true;
30     return false;
31 }
```

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, $|N(x)| = \{y, \text{ para todo } y \text{ tal que existe } (x, y) \in E\}$.
 - $N(S) = \bigcup_{s \in S} N(s)$.
- **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 König-Egerváry (1931):**
 - Seja $G = (V, E)$ um grafo bipartido. Então $\text{cobertura}(G) = \text{emparelhamento}(G)$.
 - 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 $\text{independência}(G) = n - \text{cobertura}(G)$.
- **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:
 - * $\text{val}(f^*) =$ número máximo de caminho disjuntos nas arestas.
 - * $\text{cal}(K^*) =$ 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

```

1 struct FlowEdge {
2     int v, u;
3     long long cap, flow = 0;
4     FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
5 };
6
7 struct Dinic {
8     const long long flow_inf = 1e18;
9     vector<FlowEdge> edges;
10    vector<vector<int>> adj;
11    int n, m = 0;
12    int s, t;
13    vector<int> level, ptr;
14    queue<int> q;
15
16    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
17        adj.resize(n);
18        level.resize(n);
19        ptr.resize(n);
20    }
21
22    void add_edge(int v, int u, long long cap) {
23        edges.emplace_back(v, u, cap);
24        edges.emplace_back(u, v, 0);
25        adj[v].push_back(m);
26        adj[u].push_back(m + 1);
27        m += 2;
28    }
29
30    bool bfs() {
31        while (!q.empty()) {
32            int v = q.front();
33            q.pop();
34            for (int id : adj[v]) {
35                if (edges[id].cap - edges[id].flow < 1)
36                    continue;
37                if (level[edges[id].u] != -1)
38                    continue;
39                level[edges[id].u] = level[v] + 1;
40                q.push(edges[id].u);
41            }
42        }
43        return level[t] != -1;
44    }
45
46    long long dfs(int v, long long pushed) {
47        if (pushed == 0)
48            return 0;
49        if (v == t)
50            return pushed;
51        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
52            int id = adj[v][cid];
53            int u = edges[id].u;

```



```

54         if (level[v] + 1 != level[u] || edges[id].cap - edges[id].
flow < 1)
55             continue;
56         long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].
flow));
57         if (tr == 0)
58             continue;
59         edges[id].flow += tr;
60         edges[id ^ 1].flow -= tr;
61         return tr;
62     }
63     return 0;
64 }
65
66 long long f = 0;
67 long long flow() {
68     //long long f = 0;
69     while (true) {
70         fill(level.begin(), level.end(), -1);
71         level[s] = 0;
72         q.push(s);
73         if (!bfs())
74             break;
75         fill(ptr.begin(), ptr.end(), 0);
76         while (long long pushed = dfs(s, flow_inf)) {
77             f += pushed;
78         }
79     }
80     return f;
81 }
82 vector<pair<int, int>> mincut()
83 {
84     fill(level.begin(), level.end(), -1);
85     level[s] = 0;
86     q.push(s);
87     vector<pair<int, int>> cut;
88     bfs();
89     for (auto & e: edges) {
90         if (e.flow == e.cap && level[e.v] != -1 && level[e.u] == -1
&& e.cap > 0) {
91             cut.eb(e.v, e.u);
92         }
93     }
94     return cut;
95 }
96 };

```

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.

```

1 template <class T = int>
2 class MCMF {
3 public:
4     struct Edge {
5         Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
6         int to;
7         T cap, cost;
8     };
9
10    MCMF(int size) {
11        n = size;
12        edges.resize(n);
13        pot.assign(n, 0);
14        dist.resize(n);

```

```

15     visit.assign(n, false);
16 }
17
18 pair<T, T> mcmf(int src, int sink) {
19     pair<T, T> ans(0, 0);
20     if(!SPFA(src, sink)) return ans;
21     fixPot();
22     // can use dijkstra to speed up depending on the graph
23     while(SPFA(src, sink)) {
24         auto flow = augment(src, sink);
25         ans.first += flow.first;
26         ans.second += flow.first * flow.second;
27         fixPot();
28     }
29     return ans;
30 }
31
32 void addEdge(int from, int to, T cap, T cost) {
33     edges[from].push_back(list.size());
34     list.push_back(Edge(to, cap, cost));
35     edges[to].push_back(list.size());
36     list.push_back(Edge(from, 0, -cost));
37 }
38 private:
39     int n;
40     vector<vector<int>> edges;
41     vector<Edge> list;
42     vector<int> from;
43     vector<T> dist, pot;
44     vector<bool> visit;
45
46     /*bool dij(int src, int sink) {
47         T INF = numeric_limits<T>::max();
48         dist.assign(n, INF);
49         from.assign(n, -1);
50         visit.assign(n, false);
51         dist[src] = 0;
52         for(int i = 0; i < n; i++) {
53             int best = -1;
54             for(int j = 0; j < n; j++) {
55                 if(visit[j]) continue;
56                 if(best == -1 || dist[best] > dist[j]) best = j;
57             }
58             if(dist[best] >= INF) break;
59             visit[best] = true;
60             for(auto e : edges[best]) {
61                 auto ed = list[e];
62                 if(ed.cap == 0) continue;
63                 T toDist = dist[best] + ed.cost + pot[best] - pot[ed.to];
64
65                 assert(toDist >= dist[best]);
66                 if(toDist < dist[ed.to]) {
67                     dist[ed.to] = toDist;
68                     from[ed.to] = e;
69                 }
70             }
71             return dist[sink] < INF;
72         }*/
73
74     pair<T, T> augment(int src, int sink) {
75         pair<T, T> flow = {list[from[sink]].cap, 0};
76         for(int v = sink; v != src; v = list[from[v]^1].to) {
77             flow.first = min(flow.first, list[from[v]].cap);
78             flow.second += list[from[v]].cost;
79         }

```

```

80     for(int v = sink; v != src; v = list[from[v]^1].to) {
81         list[from[v]].cap -= flow.first;
82         list[from[v]^1].cap += flow.first;
83     }
84     return flow;
85 }
86
87 queue<int> q;
88 bool SPFA(int src, int sink) {
89     T INF = numeric_limits<T>::max();
90     dist.assign(n, INF);
91     from.assign(n, -1);
92     q.push(src);
93     dist[src] = 0;
94     while(!q.empty()) {
95         int on = q.front();
96         q.pop();
97         visit[on] = false;
98         for(auto e : edges[on]) {
99             auto ed = list[e];
100             if(ed.cap == 0) continue;
101             T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
102             if(toDist < dist[ed.to]) {
103                 dist[ed.to] = toDist;
104                 from[ed.to] = e;
105                 if(!visit[ed.to]) {
106                     visit[ed.to] = true;
107                     q.push(ed.to);
108                 }
109             }
110         }
111     }
112     return dist[sink] < INF;
113 }
114
115 void fixPot() {
116     T INF = numeric_limits<T>::max();
117     for(int i = 0; i < n; i++) {
118         if(dist[i] < INF) pot[i] += dist[i];
119     }
120 }
121 };

```

7.2 Matching

7.2.1 Hopcroft Karp

Complexity for sparse graphs $O(E \log(V))$ Complexity for dense graphs $O(E \sqrt{V})$

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define MAX_V1 1000000
5 #define MAX_V2 1000000
6 #define MAX_E 8000000
7
8 int V1 = 0, V2 = 0, l[MAX_V2], r[MAX_V1];
9 int E, to[MAX_E], nex[MAX_E], last[MAX_V1];
10
11 void hk_init(){
12     memset(last, -1, sizeof last);
13     E=0;
14 }
15
16 void hk_add_edge(int u, int v){
17     to[E] = v; nex[E] = last[u]; last[u] = E++;

```

```

18 }
19
20 bool visited[MAX_V1];
21
22 bool hk_dfs(int u){
23     if(visited[u]) return false;
24     visited[u] = true;
25
26     for(int e = last[u],v; e != -1; e = nex[e]){
27         v = to[e];
28
29         if(l[v] == -1 || hk_dfs(l[v])){
30             r[u] = v;
31             l[v] = u;
32             return true;
33         }
34     }
35
36     return false;
37 }
38
39 int hk_match(){
40     memset(l,-1,sizeof l);
41     memset(r,-1,sizeof r);
42     bool change = true;
43
44     while(change){
45         change = false;
46         memset(visited,false,sizeof visited);
47
48         for(int i = 0; i < V1; ++i)
49             if(r[i] == -1)
50                 change |= hk_dfs(i);
51     }
52
53     int ret = 0;
54
55     for(int i = 0; i < V1; ++i)
56         if(r[i] != -1) ++ret;
57
58     return ret;
59 }
60
61 int n,m;
62 bool valid(int x, int y)
63 {
64     return (x > 0 && x <= n && y > 0 && y <= m);
65 }
66 int id[1005][1005];
67 int dx[] = {-1,-1,1,1,2,2,-2,-2};
68 int dy[] = {-2,2,-2,2,-1,1,-1,1};
69 int main()
70 {
71     int e;
72     scanf("%d %d %d",&V1,&V2,&e);
73     hk_init();
74     for (int i = 0; i < e; ++i)
75     {
76         int a,b;
77         cin>>a>>b;
78         hk_add_edge(a,b);
79     }
80
81     printf("%d\n",hk_match());
82     for (int i = 0; i < V1; ++i)
83     {

```

```

84         if(r[i]!=-1)
85             cout<<i<< ' ' <<r[i]<<"\n";
86     }
87     return 0;
88 }

```

7.2.2 Bipartite Weighted Hungarian Method

Complexity $O(V^3)$

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 typedef long long ll;
6
7 template<typename T>
8 class hungarian {
9 public:
10     int n;
11     int m;
12     vector< vector< T > >a;
13     vector< T > u;
14     vector< T > v;
15     vector< int > pa;
16     vector< int > pb;
17     vector< int > way;
18     vector< T > minv;
19     vector< bool > used;
20     T inf;
21
22     hungarian(int _n, int _m) : n(_n), m(_m) {
23         assert(n <= m);
24         a = vector< vector< T > >(n, vector< T >(m));
25         u = vector< T >(n + 1);
26         v = vector< T >(m + 1);
27         pa = vector< int >(n + 1, -1);
28         pb = vector< int >(m + 1, -1);
29         way = vector< int >(m, -1);
30         minv = vector< T >(m);
31         used = vector< bool >(m + 1);
32         inf = numeric_limits< T >::max();
33     }
34
35     inline void add_row(int i){
36         fill(minv.begin(), minv.end(), inf);
37         fill(used.begin(), used.end(), false);
38         pb[m] = i;
39         pa[i] = m;
40         int j0 = m;
41         do {
42             used[j0] = true;
43             int i0 = pb[j0];
44             T delta = inf;
45             int j1 = -1;
46             for(int j = 0 ; j < m ; j++){
47                 if(!used[j]){
48                     T cur = a[i0][j] - u[i0] - v[j];
49                     if(cur < minv[j]){
50                         minv[j] = cur;
51                         way[j] = j0;
52                     }
53                     if(minv[j] < delta) {
54                         delta = minv[j];
55                         j1 = j;
56                     }
57                 }
58             }
59             j0 = j1;
60         } while(j0 != m);
61     }
62
63     inline void add_col(int j){
64         u[j] = inf;
65         for(int i = 0 ; i < n ; i++){
66             if(!used[i]){
67                 T cur = a[i][j] - u[i] - v[j];
68                 if(cur < minv[j]){
69                     minv[j] = cur;
70                     way[j] = i;
71                 }
72             }
73         }
74     }
64
65     inline void solve(){
66         for(int i = 0 ; i < n ; i++){
67             add_row(i);
68         }
69         for(int j = 0 ; j < m ; j++){
70             add_col(j);
71         }
72         int i = way[0];
73         int j = 0;
74         while(j < m){
75             int i0 = way[j];
76             int j0 = j;
77             while(i0 != -1){
78                 int i1 = way[i0];
79                 int j1 = j0;
80                 j0 = j1;
81                 i0 = i1;
82             }
83             j = j0;
84             i = i0;
85         }
86     }
87
88     inline void print(){
89         for(int i = 0 ; i < n ; i++){
90             for(int j = 0 ; j < m ; j++){
91                 cout << a[i][j] << " ";
92             }
93             cout << endl;
94         }
95     }
96
97     inline void print_minv(){
98         for(int j = 0 ; j < m ; j++){
99             cout << minv[j] << " ";
100         }
101         cout << endl;
102     }
103
104     inline void print_used(){
105         for(int j = 0 ; j < m ; j++){
106             cout << used[j] << " ";
107         }
108         cout << endl;
109     }
110
111     inline void print_pa(){
112         for(int i = 0 ; i < n ; i++){
113             cout << pa[i] << " ";
114         }
115         cout << endl;
116     }
117
118     inline void print_pb(){
119         for(int j = 0 ; j < m ; j++){
120             cout << pb[j] << " ";
121         }
122         cout << endl;
123     }
124
125     inline void print_way(){
126         for(int j = 0 ; j < m ; j++){
127             cout << way[j] << " ";
128         }
129         cout << endl;
130     }
131
132     inline void print_u(){
133         for(int i = 0 ; i < n ; i++){
134             cout << u[i] << " ";
135         }
136         cout << endl;
137     }
138
139     inline void print_v(){
140         for(int j = 0 ; j < m ; j++){
141             cout << v[j] << " ";
142         }
143         cout << endl;
144     }
145
146     inline void print_inf(){
147         cout << inf << endl;
148     }
149
150     inline void print_all(){
151         print_minv();
152         print_used();
153         print_pa();
154         print_pb();
155         print_way();
156         print_u();
157         print_v();
158         print_inf();
159     }
160
161     inline void reset(){
162         a.clear();
163         u.clear();
164         v.clear();
165         pa.clear();
166         pb.clear();
167         way.clear();
168         minv.clear();
169         used.clear();
170         inf = numeric_limits< T >::max();
171     }
172
173     inline void set_n(int _n){
174         n = _n;
175     }
176
177     inline void set_m(int _m){
178         m = _m;
179     }
180
181     inline void set_a(vector< vector< T > > _a){
182         a = _a;
183     }
184
185     inline void set_u(vector< T > _u){
186         u = _u;
187     }
188
189     inline void set_v(vector< T > _v){
190         v = _v;
191     }
192
193     inline void set_pa(vector< int > _pa){
194         pa = _pa;
195     }
196
197     inline void set_pb(vector< int > _pb){
198         pb = _pb;
199     }
200
201     inline void set_way(vector< int > _way){
202         way = _way;
203     }
204
205     inline void set_minv(vector< T > _minv){
206         minv = _minv;
207     }
208
209     inline void set_used(vector< bool > _used){
210         used = _used;
211     }
212
213     inline void set_inf(T _inf){
214         inf = _inf;
215     }
216
217     inline void set_all(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){
218         set_a(_a);
219         set_u(_u);
220         set_v(_v);
221         set_pa(_pa);
222         set_pb(_pb);
223         set_way(_way);
224         set_minv(_minv);
225         set_used(_used);
226         set_inf(_inf);
227     }
228
229     inline void solve_and_print(){
230         solve();
231         print_all();
232     }
233
234     inline void reset_and_solve(){
235         reset();
236         solve();
237     }
238
239     inline void reset_and_solve_and_print(){
240         reset();
241         solve();
242         print_all();
243     }
244
245     inline void reset_and_solve_and_print_minv(){
246         reset();
247         solve();
248         print_minv();
249     }
250
251     inline void reset_and_solve_and_print_used(){
252         reset();
253         solve();
254         print_used();
255     }
256
257     inline void reset_and_solve_and_print_pa(){
258         reset();
259         solve();
260         print_pa();
261     }
262
263     inline void reset_and_solve_and_print_pb(){
264         reset();
265         solve();
266         print_pb();
267     }
268
269     inline void reset_and_solve_and_print_way(){
270         reset();
271         solve();
272         print_way();
273     }
274
275     inline void reset_and_solve_and_print_u(){
276         reset();
277         solve();
278         print_u();
279     }
280
281     inline void reset_and_solve_and_print_v(){
282         reset();
283         solve();
284         print_v();
285     }
286
287     inline void reset_and_solve_and_print_inf(){
288         reset();
289         solve();
290         print_inf();
291     }
292
293     inline void reset_and_solve_and_print_all(){
294         reset();
295         solve();
296         print_all();
297     }
298
299     inline void reset_and_solve_and_print_minv_and_used(){
300         reset();
301         solve();
302         print_minv();
303         print_used();
304     }
305
306     inline void reset_and_solve_and_print_minv_and_pa(){
307         reset();
308         solve();
309         print_minv();
310         print_pa();
311     }
312
313     inline void reset_and_solve_and_print_minv_and_pb(){
314         reset();
315         solve();
316         print_minv();
317         print_pb();
318     }
319
320     inline void reset_and_solve_and_print_minv_and_way(){
321         reset();
322         solve();
323         print_minv();
324         print_way();
325     }
326
327     inline void reset_and_solve_and_print_minv_and_u(){
328         reset();
329         solve();
330         print_minv();
331         print_u();
332     }
333
334     inline void reset_and_solve_and_print_minv_and_v(){
335         reset();
336         solve();
337         print_minv();
338         print_v();
339     }
340
341     inline void reset_and_solve_and_print_minv_and_inf(){
342         reset();
343         solve();
344         print_minv();
345         print_inf();
346     }
347
348     inline void reset_and_solve_and_print_minv_and_used_and_pa(){
349         reset();
350         solve();
351         print_minv();
352         print_used();
353         print_pa();
354     }
355
356     inline void reset_and_solve_and_print_minv_and_used_and_pb(){
357         reset();
358         solve();
359         print_minv();
360         print_used();
361         print_pb();
362     }
363
364     inline void reset_and_solve_and_print_minv_and_used_and_way(){
365         reset();
366         solve();
367         print_minv();
368         print_used();
369         print_way();
370     }
371
372     inline void reset_and_solve_and_print_minv_and_used_and_u(){
373         reset();
374         solve();
375         print_minv();
376         print_used();
377         print_u();
378     }
379
380     inline void reset_and_solve_and_print_minv_and_used_and_v(){
381         reset();
382         solve();
383         print_minv();
384         print_used();
385         print_v();
386     }
387
388     inline void reset_and_solve_and_print_minv_and_used_and_inf(){
389         reset();
390         solve();
391         print_minv();
392         print_used();
393         print_inf();
394     }
395
396     inline void reset_and_solve_and_print_minv_and_used_and_pa_and_pb(){
397         reset();
398         solve();
399         print_minv();
400         print_used();
401         print_pa();
402         print_pb();
403     }
404
405     inline void reset_and_solve_and_print_minv_and_used_and_pa_and_way(){
406         reset();
407         solve();
408         print_minv();
409         print_used();
410         print_pa();
411         print_way();
412     }
413
414     inline void reset_and_solve_and_print_minv_and_used_and_pa_and_u(){
415         reset();
416         solve();
417         print_minv();
418         print_used();
```

```

57         }
58     }
59     for(int j = 0 ; j <= m ; j++){
60         if(used[j]){
61             u[pb[j]] += delta;
62             v[j] -= delta;
63         }else{
64             minv[j] -= delta;
65         }
66     }
67     j0 = j1;
68     }while(pb[j0] != -1);
69
70     do{
71         int j1 = way[j0];
72         pb[j0] = pb[j1];
73         pa[pb[j0]] = j0;
74         j0 = j1;
75     }while(j0 != m);
76 }
77
78 inline T current_score(){
79     return -v[m];
80 }
81
82 inline pair< long long, vector< int > > solve() {
83     vector< int > ans(n, 0);
84     for(int i = 0 ; i < n ; i++){
85         add_row(i);
86     }
87     for(int i = 0 ; i < n ; i++){
88         ans[pb[i]] = i;
89     }
90
91     return make_pair(current_score(), ans);
92 }
93 };
94
95 const int N = 510;
96 const ll mx = ll(1e4) + 10;
97
98 ll prefx[N], sufx[N];
99 ll prefy[N], sufy[N];
100 ll px[N], py[N];
101 ll vx[N], vy[N];
102 vector< pair< ll, int > > x, y;
103
104 int main(){
105
106     x.push_back({-mx, -1});
107     y.push_back({-mx, -1});
108
109     int n;
110
111     scanf("%d", &n);
112
113     for(int i = 0 ; i < n ; i++){
114         scanf("%lld %lld", &px[i], &py[i]);
115         x.push_back({px[i], i});
116         y.push_back({py[i], i});
117     }
118
119     for(int i = 0 ; i < n ; i++){
120         scanf("%lld %lld", &vx[i], &vy[i]);
121     }
122

```

```

123     sort(x.begin(), x.end());
124     sort(y.begin(), y.end());
125
126     for(int i = 1 ; i < x.size() ; i++){
127         prefx[i] = x[i].first + prefx[i - 1];
128         prefy[i] = y[i].first + prefy[i - 1];
129     }
130
131     for(int i = int(x.size()) - 1 ; i >= 1 ; i--){
132         sufx[i] = sufx[i + 1] + x[i].first;
133         sufy[i] = sufy[i + 1] + y[i].first;
134     }
135
136     hungarian< ll > hg(n, n);
137     const ll INF = ll(1e14);
138
139     for(int i = 0 ; i < n ; i++){
140         for(int j = 0 ; j < n ; j++){
141             int idx = lower_bound(x.begin(), x.end(), make_pair(px[i], i
142 )) - x.begin();
143             int idy = lower_bound(y.begin(), y.end(), make_pair(py[i], i
144 )) - y.begin();
145
146             ll wx = 2LL * px[i] * vx[j] + vx[j] * vx[j] * ll(n);
147             ll wy = 2LL * py[i] * vy[j] + vy[j] * vy[j] * ll(n);
148
149             hg.a[i][j] = INF -(wx + wy);
150         }
151     }
152
153     auto u = hg.solve();
154
155     printf("Yes\n");
156
157     for(int i = 0 ; i < n ; i++){
158         printf("%d ", u.second[i] + 1);
159     }
160     printf("\n");
161     return 0;
162 }

```

7.2.3 General Graph (Edmonds Blossom)

Complexity $O(VE^2)$

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int M=500;
6 struct struct_edge{int v;struct_edge* n;};
7 typedef struct_edge* edge;
8 struct_edge pool[M*M*2];
9 edge top=pool,adj[M];
10 int V,E,match[M],qh,qt,q[M],father[M],base[M];
11 bool inq[M],inb[M],ed[M][M];
12 int a[M][M];
13 int n;
14
15 void add_edge(int u,int v){
16     top->v=v,top->n=adj[u],adj[u]=top++;
17     top->v=u,top->n=adj[v],adj[v]=top++;
18 }
19
20 int LCA(int root,int u,int v){

```

```

21  static bool inp[M];
22  memset(inp,0,sizeof(inp));
23
24  while(1){
25      inp[u=base[u]]=true;
26      if (u==root) break;
27      u=father[match[u]];
28  }
29  while(1){
30      if (inp[v=base[v]]) return v;
31      else v=father[match[v]];
32  }
33 }
34
35 void mark_blossom(int lca,int u)
36 {
37     while (base[u]!=lca)
38     {
39         int v=match[u];
40         inb[base[u]]=inb[base[v]]=true;
41         u=father[v];
42         if (base[u]!=lca) father[u]=v;
43     }
44 }
45 void blossom_contraction(int s,int u,int v){
46     int lca=LCA(s,u,v);
47     memset(inb,0,sizeof(inb));
48     mark_blossom(lca,u);
49     mark_blossom(lca,v);
50     if (base[u]!=lca)
51         father[u]=v;
52     if (base[v]!=lca)
53         father[v]=u;
54     for (int u=0;u<V;u++){
55         if (inb[base[u]]){
56             base[u]=lca;
57             if (!inq[u])
58                 inq[q[++qt]=u] = true;
59         }
60     }
61 }
62 int find_augmenting_path(int s)
63 {
64     memset(inq,0,sizeof(inq));
65     memset(father,-1,sizeof(father));
66
67     for (int i=0;i<V;i++) base[i]=i;
68     inq[q[qh=qt=0]=s]=true;
69
70     while (qh<=qt){
71         int u = q[qh++];
72         for (edge e=adj[u];e;e=e->n){
73             int v = e->v;
74             if (base[u]!=base[v]&&match[u]!=v){
75                 if ((v==s) || (match[v]!=-1 && father[match[v]]!=-1)){
76                     blossom_contraction(s,u,v);
77                 }else if (father[v]==-1){
78                     father[v] = u;
79                     if (match[v]==-1){
80                         return v;
81                     }
82                     else if (!inq[match[v]]){
83                         inq[q[++qt] = match[v]]=true;
84                     }
85                 }
86             }

```



```

87     }
88 }
89 return -1;
90 }
91
92 int augment_path(int s,int t)
93 {
94     int u=t,v,w;
95     while (u!=-1){
96         v=father[u];
97         w=match[v];
98         match[v]=u;
99         match[u]=v;
100        u=w;
101    }
102    return t!=-1;
103 }
104
105 int edmonds()
106 {
107     int matchc=0;
108     memset(match,-1,sizeof(match));
109     for (int u=0;u<V;u++){
110         if (match[u]==-1){
111             matchc+=augment_path(u,find_augmenting_path(u));
112         }
113     }
114
115     top = pool;
116     memset(adj, 0, sizeof adj);
117
118     return matchc;
119 }
120
121 bool can(int mid){
122     for(int i = 0 ; i < n ; i++){
123         for(int j = i + 1 ; j < n ; j++){
124             if(a[i][j] >= mid){
125                 add_edge(i, j);
126                 // printf("edge(%d, %d)\n", i, j);
127             }
128         }
129     }
130
131     int r = edmonds();
132
133     // printf("para %d temos %d\n", mid, r);
134
135     return r == (n / 2);
136 }
137
138 int main(){
139     int t;
140     int cs = 1;
141
142     scanf("%d", &t);
143
144     while(t--){
145         scanf("%d", &n);
146         n = 1 << n;
147         V = n;
148
149         for(int i = 0 ; i < n ; i++){
150             for(int j = i + 1 ; j < n ; j++){
151                 scanf("%d", &a[i][j]);
152                 a[j][i] = a[i][j];

```

```

153     }
154 }
155
156 int lo = 0, hi = 1e9;
157 int r = -1;
158
159 while(lo <= hi){
160     int mid = (lo + hi) >> 1;
161
162     if(can(mid)){
163         lo = mid + 1;
164         r = mid;
165     }else{
166         // printf("nao consegue!\n");
167         hi = mid - 1;
168     }
169 }
170
171 printf("Case %d: %d\n", cs++, r);
172 }
173
174 return 0;
175 }

```

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.

```

1 #include<bits/stdc++.h>
2 typedef long long ll;
3 using namespace std;
4 typedef pair<int,int> pii;
5 const int N = 3100;
6 const int M = 5100;
7 const ll inf = -1e16;
8
9 pair<pii,ll> edges[M];
10 ll dist[N];
11 int mark[N];
12 int main()
13 {
14     ios::sync_with_stdio(false);
15     cin.tie(NULL);
16     int n,m;
17     cin>>n>>m;
18     for (int i = 1; i <= n; ++i)
19     {
20         dist[i] = inf;
21     }
22     for (int i = 0; i < m; ++i)
23     {
24         int a,b;
25         cin>>a>>b>>edges[i].second;
26         edges[i].first = {a,b};
27     }
28     dist[1] = 0;
29     mark[1] = 0;
30     for (int i = 0; i <= n; ++i)
31     {
32         for (int j = 0; j < m ; ++j)
33         {
34             int a,b;

```

```

35         tie(a,b) = edges[j].first;
36         if(dist[a] > inf)
37         {
38             if(dist[b] < dist[a] + edges[j].second)
39             {
40                 mark[b] = i;
41                 dist[b] = dist[a] + edges[j].second;
42             }
43         }
44     }
45 }
46 for (int i = 0; i <= n; ++i)
47 {
48     for (int j = 0; j < m ; ++j)
49     {
50         int a,b;
51         tie(a,b) = edges[j].first;
52         mark[b] = max(mark[b],mark[a]);
53     }
54 }
55 if(mark[n]==n)
56     cout<<"-1\n";
57 else
58     cout<<dist[n]<<"\n";
59 return 0;
60 }

```

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.

- $a \vee b = a \vee b$
- $a \wedge b = (a \vee a) \wedge (b \vee b)$
- $\neg a = \neg a \vee \neg a$
- $a = a \vee a$
- $a \oplus b = (a \vee b) \wedge (\neg a \vee \neg b)$
- $\neg(a \oplus b) = (\neg a \vee b) \wedge (\neg b \vee a)$
- $a \Rightarrow b = \neg a \vee b$

```

1  int n;
2  vector<vector<int>> adj, adj_t;
3  vector<bool> used;
4  vector<int> order, comp;
5  vector<bool> assignment;
6
7  void dfs1(int v) {
8      used[v] = true;
9      for (int u : adj[v]) {
10         if (!used[u])
11             dfs1(u);
12     }
13     order.push_back(v);
14 }
15
16 void dfs2(int v, int cl) {
17     comp[v] = cl;
18     for (int u : adj_t[v]) {
19         if (comp[u] == -1)
20             dfs2(u, cl);
21     }
22 }

```

```

23
24 bool solve_2SAT() {
25     order.clear();
26     used.assign(n, false);
27     for (int i = 0; i < n; ++i) {
28         if (!used[i])
29             dfs1(i);
30     }
31
32     comp.assign(n, -1);
33     for (int i = 0, j = 0; i < n; ++i) {
34         int v = order[n - i - 1];
35         if (comp[v] == -1)
36             dfs2(v, j++);
37     }
38
39     assignment.assign(n / 2, false);
40     for (int i = 0; i < n; i += 2) {
41         if (comp[i] == comp[i + 1])
42             return false;
43         assignment[i / 2] = comp[i] > comp[i + 1];
44     }
45     return true;
46 }
47
48 #define neg(x) (x^1)
49
50 void add_disjunction(int a, bool na, int b, bool nb) {
51     // na and nb signify whether a and b are to be negated
52     a = 2*a ^ na;
53     b = 2*b ^ nb;
54     int neg_a = neg(a);
55     int neg_b = neg(b);
56     adj[neg_a].push_back(b);
57     adj[neg_b].push_back(a);
58     adj_t[b].push_back(neg_a);
59     adj_t[a].push_back(neg_b);
60 }

```

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.

```

1 struct SCC {
2     int n; // tamanho do grafo.
3     int n_cond; // tamanho do grafo condensado.
4
5     vector<vector<int>> edges, rEdges; // arestas e arestas reversas.
6     vector<vector<int>> cond_graph; // grafo condensado em uma SCC.
7     vector<int> order, comp;
8     vector<int> root_nodes, roots; // representantes de cada componente
9     e raizes
10    vector<bool> vis; // marcar os caras como visitado ou ão.
11    vector<int> in, out; // indegree e outdegree do grafo condensado.
12
13    SCC () {}
14
15    // áj inicializar todos os caras.
16    SCC(int n) {
17        this->n = n;
18        edges.resize(n);

```

```

18         rEdges.resize(n);
19         roots.resize(n);
20         in.resize(n);
21         out.resize(n);
22     }
23
24     // ou se não quiser pode usar init. Fica a gosto do usuário.
25     void init(int n) {
26         this->n = n;
27         edges.resize(n);
28         rEdges.resize(n);
29         roots.resize(n);
30         in.resize(n);
31         out.resize(n);
32     }
33
34     // dfs para andar nas arestas e arestas reversas do grafo, coletando
35     // o que é necessário.
36     void dfs(int u, vector<int> &vet, vector<vector<int>> &graph) {
37         vis[u] = true;
38         for(auto v: graph[u]) {
39             if(!vis[v]) dfs(v, vet, graph);
40         }
41         vet.push_back(u);
42     }
43
44     // adicionar arestas u->v e v->u.
45     void add_edge(int u, int v) {
46         edges[u].push_back(v);
47         rEdges[v].push_back(u);
48     }
49
50     // função para criar a Componente Fortemente Conexa.
51     void initSCC() {
52         vis.assign(n, false);
53         for(int i=0; i<n; i++){
54             if(!vis[i]) dfs(i, order, edges);
55         }
56
57         reverse(order.begin(), order.end());
58
59         vis.assign(n, false);
60         int color = 0;
61
62         for(auto v: order) {
63             if(!vis[v]) {
64                 dfs(v, comp, rEdges);
65                 root_nodes.push_back(color);
66                 for(auto u: comp){
67                     roots[u] = color;
68                 }
69                 color++;
70                 comp.clear();
71             }
72         }
73         this->n_cond = color;
74
75         cond_graph.resize(n_cond);
76
77         for(int v=0; v<n; v++) {
78             for(auto u: edges[v]) {
79                 if(roots[u] != roots[v]) {
80                     cond_graph[roots[v]].push_back(roots[u]);
81                     in[roots[u]]++;
82                     out[roots[v]]++;
83                 }
84             }
85         }
86     }

```

```

83         }
84     }
85 }
86
87 // çãfuno de solve para resolver algo usando a scc.
88 void solve() {}
89 };

```

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 $\mathcal{O}(n^2)$.

```

1  const int MAXN = 510;
2
3  queue<int> menPref[MAXN], Q;
4  int womenPref[MAXN][MAXN]; //womenPref[a][b] = c, men b is the c-th
   preference of women a
5  int womenMarryWith[MAXN];
6
7  void test()
8  {
9      int n, in;
10     int men, women, other;
11     cin >> n;
12     rep(i, 0, n)
13         womenMarryWith[i] = -1;
14
15     rep(i, 0, n){
16         cin >> men;
17         men--;
18         rep(j, 0, n){
19             cin >> in;
20             in--;
21             menPref[men].push(in);
22         }
23     }
24     rep(i, 0, n){
25         cin >> women;
26         women--;
27         rep(j, 0, n){
28             cin >> in;
29             in--;
30             womenPref[women][in] = j;
31         }
32     }
33
34     rep(i, 0, n){
35         Q.push(i);
36     }
37
38     while(!Q.empty()){
39         men = Q.front();
40         Q.pop();
41
42         if (menPref[men].empty())
43             continue;
44
45         women = menPref[men].front();
46         menPref[men].pop();

```

```

47
48     if (womenMarryWith[women] == -1){
49         womenMarryWith[women] = men;
50     }
51     else if (womenPref[women][womenMarryWith[women]] > womenPref[
women][men]){
52         other = womenMarryWith[women];
53         womenMarryWith[women] = men;
54         Q.push(other);
55     }
56     else{
57         Q.push(men);
58     }
59 }
60
61 rep(i, 0, n){
62     cout << i+1 << " " << womenMarryWith[i]+1 << "\n";
63 }
64
65 rep(i, 0, n){
66     while(!menPref[i].empty())
67         menPref[i].pop();
68 }
69 }

```

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 $\{(idom(w), w) \mid w \in V - \{S\}\}$ 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 $S = 1$.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //template
4
5 const int N = 200100;
6
7 vector<int> adj[N], tree[N], r_adj[N], bucket[N];
8 int sdom[N], par[N], dom[N], dsu[N], rot[N];
9 int in[N], rev[N], cs = 1;
10 //1-Based directed graph input

```

```

11 int findx(int u,int x=0)
12 {
13     if(u==dsu[u])return x?-1:u;
14     int v = findx(dsu[u],x+1);
15     if(v<0)return u;
16     if(sdom[rot[dsu[u]]]<sdom[rot[u]])
17         rot[u] = rot[dsu[u]];
18     dsu[u] = v;
19     return x?v:rot[u];
20 }
21 void merge(int u,int v){ //Add an edge u-->v
22     dsu[v]=u;
23 }
24 void dfs0(int u)
25 {
26     rev[cs]=u;
27     in[u] = rot[cs] = sdom[cs] = dsu[cs] = cs;
28     cs++;
29     for(int v:adj[u])
30     {
31         if(!in[v]){
32             dfs0(v);
33             par[in[v]]=in[u];
34         }
35         r_adj[in[v]].pb(in[u]);
36     }
37 }
38
39 void build_dominator_tree(int n, int s)
40 {
41     dfs0(s);
42     rep(u, n+1, 1)
43     {
44         for(int v:r_adj[u])
45             sdom[u] = min(sdom[u], sdom[findx(v)]);
46         if(u > 1)
47             bucket[sdom[u]].pb(u);
48
49         for(int v:bucket[u]){
50             int w = findx(v);
51             if(sdom[v] == sdom[w])
52                 dom[v] = sdom[v];
53             else
54                 dom[v] = w;
55         }
56         if(u > 1)
57             merge(par[u],u);
58     }
59     rep(i, 2, n+1)
60     {
61         if(dom[i]!=sdom[i])
62             dom[i]=dom[dom[i]];
63         tree[rev[i]].pb(rev[dom[i]]);
64         tree[rev[dom[i]]].pb(rev[i]);
65     }
66 }
67
68 int pai[N];
69
70 void dfs(int u, int p)
71 {
72     for(int v:tree[u])
73     {
74         if(v == p)
75             continue;
76         pai[v] = u;

```



```

77         dfs(v,u);
78     }
79 }
80
81 void test(){
82     int n, m;
83     cin >> n >> m;
84     while(m--)
85     {
86         int u, v;
87         cin >> u >> v;
88         adj[u].pb(v);
89     }
90     build_dominator_tree(n,1);
91     dfs(1,1);
92     int at = n;
93     vector<int> resp;
94     while(at)
95     {
96         resp.pb(at);
97         at = pai[at];
98     }
99     cout << resp.size() << '\n';
100    sort(all(resp));
101    for(int x:resp)
102        cout << x << ' ';
103    cout << '\n';
104 }
105
106 int32_t main(){
107     ios_base::sync_with_stdio(false);
108     cin.tie(NULL);
109
110     int t = 1;
111     // cin >> t;
112     while(t--) test();
113     return 0;
114 }

```

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. $O(2^n \cdot n)$ where n is the number of vertices. (Original Problem - UVa 10296 - Jogging Trails)

```

1 #include <stdio.h>
2 #include <string.h>
3 #define min(x, y) ((x) < (y) ? (x) : (y))
4 int map[16][16], odd[16];
5 void floyd(int n) {
6     int i, j, k;
7     for(k = 1; k <= n; k++)
8         for(i = 1; i <= n; i++)
9             for(j = 1; j <= n; j++)
10                map[i][j] = min(map[i][j], map[i][k]+map[k][j]);
11 }
12 int dp[1<<16];
13 int build(int pN, int ot) {
14     if(pN == 0)
15         return 0;
16     if(dp[pN] != -1)
17         return dp[pN];
18     int i, j, tmp;
19     dp[pN] = 0xffffffff;
20     for(i = 0; i < ot; i++) {

```

```

21         if(pN&(1<<i)) {
22             for(j = i+1; j < ot; j++) {
23                 if(pN&(1<<j)) {
24                     tmp = build(pN-(1<<i)-(1<<j), ot);
25                     dp[pN] = min(dp[pN], tmp+map[odd[i]][odd[j]]);
26                 }
27             }
28             break;
29         }
30     }
31     return dp[pN];
32 }
33 int main() {
34     int n, m, x, y, w;
35     while(scanf("%d", &n) == 1 && n) {
36         scanf("%d", &m);
37         memset(map, 63, sizeof(map));
38         memset(dp, -1, sizeof(dp));
39         int sum = 0, deg[16] = {};
40         while(m--) {
41             scanf("%d %d %d", &x, &y, &w);
42             map[x][y] = min(map[x][y], w);
43             map[y][x] = min(map[y][x], w);
44             deg[x]++, deg[y]++;
45             sum += w;
46         }
47         floyd(n);
48         int ot = 0;
49         for(int i = 1; i <= n; i++)
50             if(deg[i]&1)
51                 odd[ot++] = i;
52         printf("%d\n", sum+build((1<<ot)-1, ot));
53     }
54     return 0;
55 }

```

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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //template
5
6 const int N = 200100;
7
8 int cs = 1;
9 int id[N], low[N];
10 int vis[N];
11 vector<int> adj[N];
12 vector<pii> bridges;
13 void dfs(int u, int p) {
14     vis[u] = 1;
15     id[u] = low[u] = cs++;
16
17     int sz = 0;
18     for (int v : adj[u]) {
19         if (v == p)
20             continue;
21
22         if (!vis[v]) {
23             dfs(v, u);

```

```

24     low[u] = min(low[u], low[v]);
25     sz++;
26
27     // Check if the current u-v is an bridge
28     if (low[v] > id[u])
29         bridges.eb(u,v);
30 } else {
31     low[u] = min(low[u], id[v]);
32 }
33 }
34 }
35
36 int main() {
37     ios::sync_with_stdio(false);
38     cin.tie(NULL);
39     int n,m;
40     cin >> n >> m;
41     rep(i, 1, n+1){
42         adj[i].clear();
43         vis[i] = 0;
44     }
45     bridges.clear();
46     cs = 1;
47     rep(i, 0, m)
48     {
49         int u, v;
50         cin >> u >> v;
51         adj[u].pb(v);
52         adj[v].pb(u);
53     }
54
55     dfs(1, 1);
56
57     cout << bridges.size() << "\n";
58     for(auto [u,v]:bridges)
59         cout << u << ' ' << v << '\n';
60 }

```

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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 //template
5
6 const int N = 200100;
7
8 int cs = 1;
9 int id[N],low[N];
10 int vis[N],ap[N];
11 vector<int> adj[N];
12
13 void dfs(int u, int p) {
14     vis[u] = 1;
15     id[u] = low[u] = cs++;
16
17     int sz = 0;
18     for (int v : adj[u]) {
19         if (v == p)
20             continue;
21
22         if (!vis[v]) {
23             dfs(v, u);

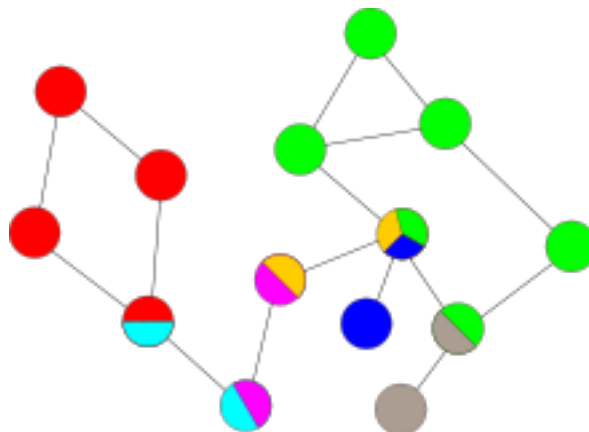
```

```

24     low[u] = min(low[u], low[v]);
25     sz++;
26
27     // Check if the current u is an articulation point
28     if (low[v] >= id[u] && p != u)
29         ap[u] = 1;
30 } else {
31     low[u] = min(low[u], id[v]);
32 }
33 }
34 if (p == u && sz > 1)
35     ap[u] = 1;
36 }
37
38 int main() {
39     int n,m;
40     cin >> n >> m;
41     rep(i, 1, n+1){
42         adj[i].clear();
43         ap[i] = vis[i] = 0;
44     }
45     cs = 1;
46     rep(i, 0, m)
47     {
48         int u, v;
49         cin >> u >> v;
50         adj[u].pb(v);
51         adj[v].pb(u);
52     }
53
54     dfs(1, 1);
55
56     int rs = 0;
57     rep(i, 1, n+1)
58         rs += ap[i];
59     cout << rs << '\n';
60 }

```

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 .

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //template
4 const int N = 200100;
5 const int K = 20;
6 int cs = 1, qt_comp = 0;
7 int id[N], low[N], rot[N];

```

```

8 int vis[N], ap[N];
9 vector<int> adj[N], tree[N];
10 vector<int> comp[N], stk;
11 void dfs0(int u, int p) {
12     vis[u] = 1;
13     id[u] = low[u] = cs++;
14     stk.pb(u);
15     for (int v : adj[u]) {
16         if (v == p)
17             continue;
18
19         if (!vis[v]) {
20             dfs0(v, u);
21             low[u] = min(low[u], low[v]);
22             // Check if the current u is an articulation point
23             if (low[v] >= id[u] && p != u)
24                 {
25                     ap[u] = (id[u] > 1 || id[v] > 2);
26                     comp[qt_comp].pb(u);
27                     while (comp[qt_comp].back() != v) {
28                         comp[qt_comp].pb(stk.back());
29                         stk.pop_back();
30                     }
31                     qt_comp++;
32                 }
33         } else {
34             low[u] = min(low[u], id[v]);
35         }
36     }
37 }
38 int n_tree;
39 // return the size of the tree
40 int build_block(int n)
41 {
42     dfs0(1, -1);
43     // Build the block-cut tree
44     int node_id = 1;
45     rep(i, 1, n+1)
46         if (ap[i])
47             rot[i] = node_id++;
48
49     rep(j, 0, qt_comp) {
50         int node = node_id++;
51         for (int u : comp[j])
52             if (!ap[u]) {
53                 rot[u] = node;
54             } else {
55                 tree[node].pb(rot[u]);
56                 tree[rot[u]].pb(node);
57             }
58     }
59     return node_id-1;
60 }
61
62 int up[N][K], in[N], out[N], tmp = 0;;
63 void dfs(int u, int p)
64 {
65     in[u] = tmp++;
66     up[u][0] = p;
67     rep(j, 1, K) {
68         up[u][j] = up[up[u][j-1]][j-1];
69     }
70     for (auto v : tree[u]) {
71         if (v == p) continue;
72         dfs(v, u);
73     }

```

```

74     out[u] = tmp-1;
75 }
76 bool is_ancestor(int u, int v)
77 {
78     return (in[u] <= in[v] && out[v] <= out[u]);
79 }
80 int calc_lca(int u, int v)
81 {
82     if(is_ancestor(u,v))
83         return u;
84
85     if(is_ancestor(v,u))
86         return v;
87     rep(j, (int)K, 0)
88     {
89         if(!is_ancestor(up[u][j],v))
90             u = up[u][j];
91     }
92     return up[u][0];
93 }
94 int main() {
95     int n, m, q;
96     cin >> n >> m >> q;
97
98     for (int i = 0; i < m; i++) {
99         int u, v;
100         cin >> u >> v;
101         adj[u].pb(v);
102         adj[v].pb(u);
103     }
104
105     n_tree = build_block(n);
106
107     // Check whether the node z is on path (x, y)
108     auto on_path = [&](int x, int y, int z) {
109         int lca = calc_lca(x, y);
110         int lca1 = calc_lca(x, z);
111         int lca2 = calc_lca(y, z);
112
113         if (lca == z || (lca1 == lca && lca2 == z) ||
114             (lca2 == lca && lca1 == z)) {
115             return true;
116         }
117         return false;
118     };
119
120     dfs(1, 1);
121
122     for (int i = 0; i < q; i++) {
123         int a, b, c;
124         cin >> a >> b >> c;
125         // The path does not exist in two cases:
126         // 1) a == c or b == c
127         // 2) c is a cutpoint and it lies on path between a and b in block-
128         cut
129         // tree
130         if (a == c || b == c ||
131             (ap[c] && on_path(rot[a], rot[b], rot[c]))) {
132             cout << "NO\n";
133         } else {
134             cout << "YES" << '\n';
135         }
136     }

```

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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // template
5
6 typedef long long ll;
7
8 const int N = 200100;
9
10 int cs = 1; // Time of entry in u
11 int qt_comp; // Number of strongly connected components
12 int id[N];
13 int low[N], rot[N]; // Lowest ID in u's subtree in DFS tree
14 vector<int> adj[N];
15 vector<int> comp[N], stk;
16 void dfs(int u, int p = -1) {
17     id[u] = low[u] = cs++;
18     stk.pb(u);
19     // careful with multiple edges
20     int did = 0;
21
22     for (int v : adj[u]) {
23         if (v == p && !did) {
24             did = 1;
25             continue;
26         }
27         if (!id[v]) {
28             dfs(v, u);
29             low[u] = min(low[u], low[v]);
30         } else {
31             low[u] = min(low[u], id[v]);
32         }
33     }
34
35     if (low[u] == id[u]) {
36         while (stk.back() != u) {
37             rot[stk.back()] = qt_comp;
38             comp[qt_comp].pb(stk.back());
39             stk.pop_back();
40         }
41         rot[stk.back()] = qt_comp;
42         comp[qt_comp++].pb(stk.back());
43         stk.pop_back();
44     }
45 }
46
47 int main() {
48     int n, m;
49     cin >> n >> m;
50     rep(i, 0, m) {
51         int x, y;
52         cin >> x >> y;
53         adj[x].push_back(y);
54         adj[y].push_back(x);
55     }
56
57     rep(u, 0, n)
58         if (!id[u])
59             dfs(u);
60
61
62     cout << qt_comp << '\n';

```

```
63     rep(i, 0, qt_comp) {
64         cout << comp[i].size() << ' ';
65         for (int u : comp[i]) { cout << u << ' '; }
66         cout << '\n';
67     }
68 }
```


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.

```
1 const int N = 200100;
2
3 vector<int> adj[N];
4
5 pair<int, int> dfs_diameter(int u, int p, int dep)
6 {
7     pair<int, int> best;
8     best = {dep, u};
9     for(int v : adj[u])
10         if(v != p)
11             best = max(best, dfs_diameter(v, u, dep+1));
12     return best;
13 }
14
15
16 void test()
17 {
18     int n;
19     cin >> n;
20     for(int i = 0, u, v, t; i < n-1; i++)
21     {
22         cin >> u >> v;
23         adj[u].emplace_back(v);
24         adj[v].emplace_back(u);
25     }
26     int at = dfs_diameter(1, 1, 1).second;
27     cout << dfs_diameter(at, at, 1).first << '\n';
28 }
```

8.2 LCA

A way to find the lowest common ancestor of two vertex (a, b) in a tree. Needs a Sparse Table.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 const int N = 300100;
5 const int K = 20;
6 vector<int> adj[N];
7 int pai[N][K], in[N], out[N], cs = 0;
8
9 void pre_dfs(int u, int p)
10 {
11     in[u] = cs++;
12     pai[u][0] = p;
13     for(int i = 1; i < K; i++){
14         pai[u][i] = pai[pai[u][i-1]][i-1];
15     }
```

```
16     for(int v : adj[u])
17         if(v != p)
18             pre_dfs(v, u);
19     out[u] = cs++;
20 }
21
22 // u is ancestor of v ?
23 bool is_ancestor(int u, int v)
24 {
25     return in[u] <= in[v] && out[v] <= out[u];
26 }
27
28 int calc_lca(int a, int b)
29 {
30     if(is_ancestor(a, b))
31         return a;
32     if(is_ancestor(b, a))
33         return b;
34     for(int i = K-1; i >=0; i--)
35     {
36         if(!is_ancestor(pai[b][i], a))
37             b = pai[b][i];
38     }
39     return pai[b][0];
40 }
41
42
43
44 void test()
45 {
46     int n, q;
47     cin >> n;
48     for(int i = 0, u, v; i < n-1; i++)
49     {
50         cin >> u >> v;
51         adj[u].emplace_back(v);
52         adj[v].emplace_back(u);
53     }
54     pre_dfs(1, 1);
55     cin >> q;
56     while(q--)
57     {
58         int a, b;
59         cin >> a >> b;
60         cout << calc_lca(a, b) << '\n';
61     }
62 }
63
64 int32_t main()
65 {
66     ios::sync_with_stdio(false);
67     cin.tie(NULL);
68
69     test();
70
71     return 0;
72 }
```

Version that Sabino likes

```
1  /*
2      Indexado de 1
3
4      Alteracoes:
5
6      As vezes mudar como calcula o calc do lca
7  */
```

```

8 struct LCA{
9     vector<vector<int>> graph;
10    int n;
11    // entrada, saida, e euler tour
12    int timer=0;
13    vector<int> tin,tout,euler;
14    // se vai diferenciar ou n o tin do tout
15    int flag;
16    // distancia de cada cara pra raiz
17    vector<int> dist;
18    // tabela de ancestrais
19    vector<vector<int>> pai;
20    // raiz
21    int r;
22    int TETO;
23
24    LCA(vector<vector<int>> &graph,int n,int flag=0,int r=1){
25        this->n = n;
26        this->graph = graph;
27        tin.resize(n+10);
28        tout.resize(n+10);
29        this->flag = flag;
30        euler.resize((flag ? 2*n+10 : n+10));
31        dist.resize(n+10);
32        int t=0;
33        int d=1;
34        while(d <= n){
35            t++;
36            d<=&1;
37        }
38        TETO = t;
39        pai.resize(n+10,vector<int>(TETO));
40        this->r = r;
41
42        build(r,r);
43    }
44
45    void build(int u,int ant){
46        if(u == ant) dist[u]=0;
47        else dist[u]=dist[ant]+1;
48        tin[u]=++timer;
49        pai[u][0]=ant;
50        // construir tabela de ancestrais
51        for(int i=1; i < TETO; i++) {
52            pai[u][i]=pai[pai[u][i-1]][i-1];
53        }
54        for(auto v: graph[u]){
55            if(v == ant) continue;
56            build(v,u);
57        }
58        tout[u]=(flag ? ++timer : timer);
59    }
60
61    // x eh lca de y?
62    bool lca(int x,int y){
63        return tin[x] <= tin[y] && tout[x] >= tout[y];
64    }
65
66    // calcula alguma coisa, deixei como se fosse dist, entre x e y
67    int calc(int x,int y){
68        if(lca(x,y)) return dist[y]-dist[x];
69        else if(lca(y,x)) return dist[x]-dist[y];
70        int z=x;
71        for(int i=TETO-1; i>=0; i--){
72            if(!lca(pai[z][i],y)) z=pai[z][i];
73        }

```

```

74         z=pai[z][0];
75         return dist[x]+dist[y]-2*dist[z];
76     }
77 };

```

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 $1, 2, \dots, n$. 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 .

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int,int> pii;
5 #define lef(x) (x << 1)
6 #define rig(x) (lef(x) | 1)
7 const int N = 200100;
8
9 class Segtree {
10 public:
11     const ll emp = 0; //NULL VALUE
12     vector<ll> tree;
13     Segtree(){}
14     ll f(ll a, ll b){return max(a,b);} //function
15
16     void updt(int p, int tl, int tr, int k, ll v)
17     {

```

```

18     if(tl == tr){
19         tree[p] = v;
20         return;
21     }
22     int tmid = (tl+tr) >> 1;
23     if (tmid < k)
24         updt(rig(p), tmid+1, tr, k, v);
25     else
26         updt(lef(p), tl, tmid, k, v);
27     tree[p] = f(tree[lef(p)], tree[rig(p)]);
28
29 }
30
31 ll query(int p, int tl, int tr, int l, int r)
32 {
33     if(l > r)
34         return emp;
35     if(tl == l && tr == r)
36         return tree[p];
37     int tmid = (tl+tr) >> 1;
38     return f(query(lef(p), tl, tmid, l, min(r,tmid)),
39             query(rig(p), tmid+1, tr, max(l,tmid+1), r));
40 }
41
42 };
43 Segtree tree[N];
44
45 vector<int> adj[N];
46 int depth[N], hd[N], sz[N], pai[N], sub_sz[N], pos[N];
47 ll a[N];
48
49 void dfs(int u,int p = -1)
50 {
51     pai[u] = p;
52     sub_sz[u] = 1;
53     if(p!=-1)
54         depth[u] = depth[p] + 1;
55
56     for(int v: adj[u])
57     {
58         if(v != p){
59             dfs(v, u);
60             sub_sz[u] += sub_sz[v];
61         }
62     }
63 }
64 void dfs2(int u,int p = 1)
65 {
66     for(int v: adj[u])
67     {
68         if(v == p)
69             continue;
70         if(sub_sz[u] <= 2*sub_sz[v]){
71             hd[v] = hd[u];
72             pos[v] = ++sz[hd[u]];
73             dfs2(v, u);
74         }else
75         {
76             hd[v] = v;
77             sz[v] = 1;
78             pos[v] = 1;
79             dfs2(v, u);
80         }
81     }
82     if(u == hd[u])
83         tree[u].tree.resize(sz[u] << 2,0);

```

```

84 }
85 void build_hld(int n)
86 {
87     dfs(1);
88     hd[1] = sz[1] = pos[1] = 1;
89     dfs2(1);
90     for(int i = 1; i <= n; i++)
91     {
92         tree[hd[i]].updt(1,1,sz[hd[i]],pos[i],a[i]);
93     }
94 }
95 ll query(int a,int b)
96 {
97     ll ans = 0;
98     while(hd[a] != hd[b])
99     {
100         // cout << a << ' ' << b << '\n';
101         if(depth[hd[a]] > depth[hd[b]])
102             swap(a,b);
103         ans = max(ans, tree[hd[b]].query(1,1,sz[hd[b]],pos[hd[b]],pos[b
104     ]));
105         b = pai[hd[b]];
106     }
107     if(pos[a] > pos[b])
108         swap(a,b);
109     ans = max(ans, tree[hd[b]].query(1,1,sz[hd[b]],pos[a],pos[b]));
110     return ans;
111 }
112 int main()
113 {
114     ios::sync_with_stdio(false);
115     cin.tie(NULL);
116     int n, q;
117     cin >> n >> q;
118
119     for(int i = 1; i <= n; i++)
120         cin >> a[i];
121     for(int i = 1; i < n; i++)
122     {
123         int u, v;
124         cin >> u >> v;
125         adj[u].push_back(v);
126         adj[v].push_back(u);
127     }
128
129     build_hld(n);
130     while(q-- )
131     {
132         int t;
133         cin >> t;
134         if(t == 2)
135         {
136             int u, v;
137             cin >> u >> v;
138             cout << query(u, v) << "\n";
139         }else
140         {
141             int u;
142             ll x;
143             cin >> u >> x;
144             tree[hd[u]].updt(1,1,sz[hd[u]],pos[u],x);
145             a[u] = x;
146         }
147     }
148     return 0;

```

8.5.1 Can you answer these queries VII

Given are a tree with n vertices, each vertex 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).

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int N = int(2e5 + 10);
6 typedef long long ll;
7
8 vector< vector< int > > adj;
9 vector< int > parent, depth, heavy, head, pos;
10 int cur_pos;
11 int n;
12 int carry[N];
13 int idx[N];
14 ll arr[N];
15
16 int dfs(int v){
17     int size = 1;
18     int max_c_size = 0;
19
20     for(int c: adj[v]){
21         if(c != parent[v]){
22             parent[c] = v, depth[c] = depth[v] + 1;
23             int c_size = dfs(c);
24             size += c_size;
25
26             if(c_size > max_c_size){
27                 max_c_size = c_size, heavy[v] = c;
28             }
29         }
30     }
31
32     return size;
33 }
34
35 void decompose(int v, int h){
36     idx[cur_pos] = v;
37     head[v] = h, pos[v] = cur_pos++;
38
39     if(heavy[v] != -1){
40         decompose(heavy[v], h);
41     }
42
43     for(int c: adj[v]){
44         if(c != parent[v] && c != heavy[v]){
45             decompose(c, c);
46         }
47     }
48 }
49
50 class Node{
51 public:
52     ll pref, suf, best, total;
53 };
54
55 const ll inf = ll(1e10);
56 ll lazy[N << 2];
57 int lazy1[N << 2];

```

```

58 Node segtree[N << 2];
59
60 void shift(int node, ll el){
61     int lf = node << 1;
62     int rg = lf + 1;
63
64     ll val = max(0LL, lazy[node] * el);
65     segtree[node].pref = val;
66     segtree[node].suf = val;
67     segtree[node].best = val;
68     segtree[node].total = lazy[node] * el;
69
70     lazy[lf] = lazy[node];
71     lazy[rg] = lazy[node];
72     lazy1[lf] = lazy1[rg] = 1;
73
74     lazy1[node] = 0;
75 }
76
77 Node merge(Node lf, Node rg){
78     Node r;
79
80     if(lf.total == -inf){
81         return rg;
82     }
83
84     if(rg.total == -inf){
85         return lf;
86     }
87
88     r.pref = max(lf.pref, lf.total + rg.pref);
89     r.suf = max(rg.suf, rg.total + lf.suf);
90     r.best = max({r.pref, r.suf, lf.best, rg.best, lf.suf + rg.pref});
91     r.total = lf.total + rg.total;
92
93     return r;
94 }
95
96 void build(int node, int i, int j){
97     if(i == j){
98         segtree[node].total = arr[idx[i] + 1];
99         segtree[node].suf = max(0LL, arr[idx[i] + 1]);
100        segtree[node].pref = max(0LL, arr[idx[i] + 1]);
101        segtree[node].best = max(0LL, arr[idx[i] + 1]);
102    }else{
103        int mid = (i + j) >> 1;
104        int lf = node << 1;
105        int rg = lf + 1;
106
107        build(lf, i, mid);
108        build(rg, mid + 1, j);
109
110        segtree[node] = merge(segtree[lf], segtree[rg]);
111    }
112 }
113
114 void update(int node, int i, int j, int l, int r, ll val){
115     if(lazy1[node]){
116         shift(node, j - i + 1);
117     }
118
119     if(i > r || j < l) return;
120
121     if(l <= i && j <= r){
122         lazy[node] = val;
123         lazy1[node] = 1;

```



```

124     shift(node, j - i + 1);
125 }else{
126     int mid = (i + j) >> 1;
127     int lf = node << 1;
128     int rg = lf + 1;
129
130     update(lf, i, mid, l, r, val);
131     update(rg, mid + 1, j, l, r, val);
132
133     segtree[node] = merge(segtree[lf], segtree[rg]);
134 }
135 }
136
137 Node segment_tree_query(int node, int i, int j, int l, int r){
138     if(lazy1[node]){
139         shift(node, j - i + 1);
140     }
141
142     if(l > r) return {-inf, -inf, -inf, -inf};
143
144     if(i > r || j < l){
145         return {-inf, -inf, -inf, -inf};
146     }
147
148     if(l <= i && j <= r){
149         return segtree[node];
150     }else{
151         int mid = (i + j) >> 1;
152         int lf = node << 1;
153         int rg = lf + 1;
154
155         return merge(segment_tree_query(lf, i, mid, l, r),
156                     segment_tree_query(rg, mid + 1, j, l, r));
157     }
158 }
159
160 void init(){
161     int n = adj.size();
162     parent = vector< int >(n);
163     depth = vector< int >(n);
164     heavy = vector< int >(n, -1);
165     head = vector< int >(n);
166     pos = vector< int >(n);
167
168     cur_pos = 1;
169     dfs(0);
170     decompose(0, 0);
171     build(1, 1, n);
172 }
173
174 Node merge2(Node lf, Node rg){
175     Node r;
176     r.best = max({lf.best, rg.best, rg.pref + lf.pref});
177
178     return r;
179 }
180
181 ll query(int a, int b){
182     Node res[2] = {{0LL, 0LL, 0LL, 0LL}, {0LL, 0LL, 0LL, 0LL}};
183     int c = 0;
184
185     for(; head[a] != head[b] ; b = parent[head[b]]){
186         if(depth[head[a]] > depth[head[b]]){
187             swap(a, b);
188             c = !c;
189         }

```

```

190     res[c] = merge(segment_tree_query(1, 1, n, pos[head[b]], pos[b]),
191     res[c]);
192 }
193 if(pos[a] > pos[b]){
194     c = !c;
195     swap(a, b);
196 }
197
198 res[c] = merge(segment_tree_query(1, 1, n, pos[a], pos[b]), res[c]);
199 res[c] = merge2(res[c], res[!c]);
200
201 return res[c].best;
202 }
203
204 void updatet(int a, int b, ll val){
205     for(; head[a] != head[b] ; b = parent[head[b]]){
206         if(depth[head[a]] > depth[head[b]]){
207             swap(a, b);
208         }
209         update(1, 1, n, pos[head[b]], pos[b], val);
210     }
211
212     if(pos[a] > pos[b]){
213         swap(a, b);
214     }
215     update(1, 1, n, pos[a], pos[b], val);
216 }
217
218 int main(){
219     scanf("%d", &n);
220     for(int i = 1 ; i <= n ; i++){
221         scanf("%lld", &arr[i]);
222     }
223
224     adj.resize(n);
225
226     for(int i = 1 ; i < n ; i++){
227         int ui, vi;
228         scanf("%d %d", &ui, &vi);
229         ui--, vi--;
230         adj[ui].push_back(vi);
231         adj[vi].push_back(ui);
232     }
233
234     int q;
235     init();
236     scanf("%d", &q);
237
238     while(q--){
239         int t;
240         scanf("%d", &t);
241
242         if(t == 1){
243             int a, b;
244             scanf("%d %d", &a, &b);
245             a--, b--;
246             printf("%lld\n", query(a, b));
247         }else{
248             int a, b;
249             ll c;
250             scanf("%d %d %lld", &a, &b, &c);
251             a--, b--;
252             updatet(a, b, c);
253         }
254     }

```

```

255
256     return 0;
257 }

```

8.5.2 Can you answer these queries VI

Given a tree with n vertices, each vertex has a color (0 or 1), and two type of operations. Operation 0 asks for how many vertices are connected to a vertex u , when one vertex 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 vertex u (0 to 1 and 1 to 0).

The problem can be solved with HLD in $O(Q(\log(n))^2)$. Original Problem (SPOJ - Can You Answer These Queries VI).

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 typedef pair< int, int > pii;
6 const int N = int(1e5 + 10);
7 vector< int > adj[N];
8 vector< int > parent, depth, heavy, head, pos;
9 int sz[N], idx[N];
10 pii lazy[N << 3];
11 int cur_pos;
12 pii segtree1[N << 3];
13 pii segtree2[N << 3];
14 int n;
15 int pivot = N - 5;
16 int cor[N];
17
18 int dfs(int v){
19     int size = 1;
20     int max_c_size = 0;
21
22     for(int c: adj[v]){
23         if(c != parent[v]){
24             parent[c] = v, depth[c] = depth[v] + 1;
25             int c_size = dfs(c);
26             size += c_size;
27
28             if(c_size > max_c_size){
29                 max_c_size = c_size, heavy[v] = c;
30             }
31         }
32     }
33
34     sz[v] = size;
35
36     return size;
37 }
38
39 void decompose(int v, int h){
40     idx[cur_pos] = v;
41     head[v] = h, pos[v] = cur_pos++;
42
43     // printf("head[%d] = %d, %d\n", v, h, heavy[v]);
44
45     if(heavy[v] != -1){
46         decompose(heavy[v], h);
47     }
48
49     for(int c: adj[v]){
50         if(c != parent[v] && c != heavy[v]){
51             decompose(c, c);
52         }
53     }

```

```

54 }
55
56 pii merge(pii lf, pii rg){
57     pii r = {max(lf.first, rg.first), max(lf.second, rg.second)};
58
59     return r;
60 }
61
62 pii merge2(pii lf, pii rg){
63     pii r = {max(lf.first, rg.first), max(lf.second, rg.second)};
64
65     return r;
66 }
67
68 void build1(int node, int i, int j){
69     if(i == j){
70         if(i != 1){
71             segtree1[node] = {i, -1};
72         }else{
73             segtree1[node] = {i, i}; //pivot is black and white
74         }
75     }else{
76         int mid = (i + j) >> 1;
77         int lf = node << 1;
78         int rg = lf + 1;
79
80         build1(lf, i, mid);
81         build1(rg, mid + 1, j);
82
83         segtree1[node] = merge(segtree1[lf], segtree1[rg]);
84     }
85 }
86
87 pii query1(int node, int i, int j, int l, int r){
88     if(i > r || j < l){
89         return {-1, -1};
90     }
91
92     if(l <= i && j <= r){
93         return segtree1[node];
94     }else{
95         int mid = (i + j) >> 1;
96         int lf = node << 1;
97         int rg = lf + 1;
98
99         return merge(
100             query1(lf, i, mid, l, r),
101             query1(rg, mid + 1, j, l, r)
102         );
103     }
104 }
105
106 void update1(int node, int i, int j, int pos){
107     if(i > pos || j < pos) return;
108
109     if(i == pos && j == pos){
110         swap(segtree1[node].first, segtree1[node].second);
111     }else{
112         int mid = (i + j) >> 1;
113         int lf = node << 1;
114         int rg = lf + 1;
115
116         update1(lf, i, mid, pos);
117         update1(rg, mid + 1, j, pos);
118
119         segtree1[node] = merge(segtree1[lf], segtree1[rg]);

```

```

120     }
121 }
122
123 void build2(int node, int i, int j){
124     if(i == j){
125         if(i != 1){
126             segtree2[node] = {sz[idx[i]], 0};
127         }else{
128             segtree2[node] = {sz[0], 0}; //pivot
129         }
130     }else{
131         int mid = (i + j) >> 1;
132         int lf = node << 1;
133         int rg = lf + 1;
134
135         build2(lf, i, mid);
136         build2(rg, mid + 1, j);
137
138         segtree2[node] = merge2(segtree2[lf], segtree2[rg]);
139     }
140 }
141
142 pii add(pii a, pii b){
143     return {a.first + b.first, a.second + b.second};
144 }
145
146 void shift(int node){
147     int lf = node << 1;
148     int rg = lf + 1;
149
150     segtree2[node].first += lazy[node].first;
151     segtree2[node].second += lazy[node].second;
152
153     lazy[lf] = add(lazy[lf], lazy[node]);
154     lazy[rg] = add(lazy[rg], lazy[node]);
155
156     lazy[node] = {0, 0};
157 }
158
159 pii query2(int node, int i, int j, int l, int r){
160     if(lazy[node].first || lazy[node].second){
161         shift(node);
162     }
163
164     if(j < l || i > r){
165         return {0, 0};
166     }
167
168     if(l <= i && j <= r){
169         return segtree2[node];
170     }else{
171         int mid = (i + j) >> 1;
172         int lf = node << 1;
173         int rg = lf + 1;
174
175         return merge2(query2(lf, i, mid, l, r), query2(rg, mid + 1, j, l, r)
176     );
177     }
178 }
179
180 void update2(int node, int i, int j, int l, int r, pii updt){
181     if(lazy[node].first || lazy[node].second){
182         shift(node);
183     }
184
185     if(j < l || i > r){

```

```

185     return;
186 }
187
188 if(l <= i && j <= r){
189     lazy[node] = updt;
190
191     shift(node);
192 }else{
193     int mid = (i + j) >> 1;
194     int lf = node << 1;
195     int rg = lf + 1;
196
197     update2(lf, i, mid, l, r, updt);
198     update2(rg, mid + 1, j, l, r, updt);
199
200     segtree2[node] = merge2(segtree2[lf], segtree2[rg]);
201 }
202 }
203
204 void init(){
205     parent = vector< int >(N, -1);
206     depth = vector< int >(N);
207     heavy = vector< int >(N, -1);
208     head = vector< int >(N);
209     pos = vector< int >(N);
210
211     cur_pos = 1;
212     dfs(pivot);
213
214     decompose(pivot, pivot);
215     build1(1, 1, n);
216     build2(1, 1, n);
217 }
218
219 int query(int a){
220     int st = cor[a];
221     int best = 0;
222
223     while(true){
224         if(cor[a] != st) return best;
225
226         pii r = query1(1, 1, n, pos[head[a]], pos[a]);
227         // printf("%d, %d\n", r.first, r.second);
228         if(st == 0){
229             if(r.second != -1){
230                 // printf("s: %d\n", idx[r.second]);
231                 // printf("1.%d->%d\n", idx[r.second], idx[r.second]);
232                 while(best > query2(1, 1, n, r.second + 1, r.second + 1).first);
233                 return query2(1, 1, n, r.second + 1, r.second + 1).first;
234             }else{
235                 // printf("2.%d->%d %d\n", head[a], a, query2(1, 1, n, pos[a],
pos[head[a]]).first);
236                 while(query2(1, 1, n, pos[head[a]], pos[a]).first < best);
237                 best = max(best, query2(1, 1, n, pos[head[a]], pos[a]).first);
238             }
239         }else{
240             if(r.first != -1){
241                 // printf("3.%d->%d %d\n", idx[r.first], idx[r.first + 1],
best);
242                 while(best > query2(1, 1, n, r.first + 1, r.first + 1).second);
243                 return query2(1, 1, n, r.first + 1, r.first + 1).second;
244             }else{
245                 // printf("4.%d->%d\n", head[a], a);
246                 while(best > query2(1, 1, n, pos[head[a]], pos[a]).second);
247                 best = max(best, query2(1, 1, n, pos[head[a]], pos[a]).second);
248             }

```

```

249     }
250
251     a = parent[head[a]];
252 }
253
254 return best;
255 }
256
257 void update(int a){
258     int st = cor[a];
259     pii insz;
260
261     update1(1, 1, n, pos[a]);
262
263     insz = query2(1, 1, n, pos[a], pos[a]);
264
265     if(st == 0){
266         update2(1, 1, n, pos[a], pos[a], {-1, 1});
267         // printf("updt: %d, %d (%d)\n", insz.first - 1, insz.second + 1, a)
268         ;
269     }else{
270         update2(1, 1, n, pos[a], pos[a], {1, -1});
271     }
272
273     cor[a] = !cor[a];
274
275     a = parent[a];
276     int tmp = a;
277
278     while(true){
279         pii r = query1(1, 1, n, pos[head[a]], pos[a]);
280
281         if(st == 1){
282             if(r.second != -1){
283                 update2(1, 1, n, r.second, pos[a], {insz.first + 1, 0});
284                 break;
285             }else{
286                 update2(1, 1, n, pos[head[a]], pos[a], {insz.first + 1, 0});
287             }
288         }else{
289             if(r.first != -1){
290                 update2(1, 1, n, r.first, pos[a], {0, insz.second + 1});
291                 break;
292             }else{
293                 update2(1, 1, n, pos[head[a]], pos[a], {0, insz.second + 1});
294             }
295         }
296
297         a = parent[head[a]];
298     }
299
300     a = tmp;
301
302     while(true){
303         pii r = query1(1, 1, n, pos[head[a]], pos[a]);
304
305         if(st == 0){
306             if(r.second != -1){
307                 update2(1, 1, n, r.second, pos[a], {-insz.first, 0});
308                 break;
309             }else{
310                 update2(1, 1, n, pos[head[a]], pos[a], {-insz.first, 0});
311             }
312         }else{
313             if(r.first != -1){
314                 update2(1, 1, n, r.first, pos[a], {0, -insz.second});

```

```

314         break;
315     }else{
316         update2(1, 1, n, pos[head[a]], pos[a], {0, -insz.second});
317     }
318 }
319
320 a = parent[head[a]];
321 }
322 }
323
324 int main(){
325     scanf("%d", &n);
326
327     for(int i = 1 ; i < n ; i++){
328         int ui, vi;
329
330         scanf("%d %d", &ui, &vi);
331
332         ui--, vi--;
333         adj[ui].push_back(vi);
334         adj[vi].push_back(ui);
335     }
336
337     adj[pivot].push_back(0);
338     n++;
339
340     init();
341
342     int q;
343
344     scanf("%d", &q);
345
346     while(q--){
347         int op, u;
348
349         scanf("%d %d", &op, &u);
350
351         u--;
352
353         if(op == 0){
354             printf("%d\n", query(u));
355         }else{
356             update(u);
357         }
358     }
359
360     return 0;
361 }

```

8.5.3 HLD with subtree queries

Slower version with subtree queries, uses lazy segtree and template

```

1 int VALS_IN_EDGES = 0;
2 // namespace HLD {
3     int n;
4     vector<int> adj[N];
5     int par[N], root[N], dep[N], sz[N], ti;
6     int pos[N];
7     vector<int> rpos; // rpos not used but could be useful
8     void add_edge(int x, int y) { adj[x].pb(y), adj[y].pb(x); }
9     void dfsSz(int x) {
10         sz[x] = 1;
11         for(auto y:adj[x]) {
12             par[y] = x; dep[y] = dep[x]+1;
13             adj[y].erase(find(all(adj[y]),x)); /// remove parent from adj list

```



```

14     dfsSz(y); sz[x] += sz[y];
15     if (sz[y] > sz[adj[x][0]]) swap(y, adj[x][0]);
16 }
17 }
18 void dfsHld(int x) {
19     pos[x] = ti++; rpos.pb(x);
20     for(auto y:adj[x]) {
21         root[y] = (y == adj[x][0] ? root[x] : y);
22         dfsHld(y);
23     }
24 }
25 void init(int nn, int r = 1) {
26     n = nn;
27     par[r] = dep[r] = ti = 0;
28     dfsSz(r);
29     root[r] = r;
30     dfsHld(r);
31     // build(1, 1, n);
32 }
33 int lca(int x, int y) {
34     for (; root[x] != root[y]; y = par[root[y]])
35         if (dep[root[x]] > dep[root[y]]) swap(x,y);
36     return dep[x] < dep[y] ? x : y;
37 }
38 int dist(int x, int y) { // # edges on path
39     return dep[x]+dep[y]-2*dep[lca(x,y)];
40 }
41 template <class BinaryOp>
42 void processPath(int x, int y, BinaryOp op) {
43     for (; root[x] != root[y]; y = par[root[y]]) {
44         if (dep[root[x]] > dep[root[y]])
45             swap(x,y);
46         op(pos[root[y]], pos[y]);
47     }
48     if (dep[x] > dep[y])
49         swap(x,y);
50     op(pos[x]+VALS_IN_EDGES, pos[y]);
51 }
52 void modifyPath(int x, int y, int v) {
53     processPath(x,y,[&v](int l, int r) {
54         updt(1, 1, n, l, r, v); });
55 }
56 void modifyVert(int x, int v) {
57     updt(1,1,n,pos[x],pos[x],v);
58 }
59 ll queryPath(int x, int y) {
60     ll res = 0; processPath(x,y,[&res](int l, int r) {
61         res = f(res,query(1, 1, n, l, r)); });
62     return res;
63 }
64 void modifySubtree(int x, int v) {
65     updt(1,1,n,pos[x]+VALS_IN_EDGES,pos[x]+sz[x]-1,v);
66 }

```

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 $O(\log(N))$.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef pair<int,int> pii;
5 const int N = 200100;

```

```

6 // KEEP
7 vector<int> adj[N];
8 int sz[N], did[N];
9 void dfs(int u,int p )
10 {
11     sz[u] = 1;
12     for(int v:adj[u])
13     {
14         if(v == p || did[v])
15             continue;
16         dfs(v, u);
17         sz[u] += sz[v];
18     }
19 }
20
21 int centroid(int u,int p, int sub_sz)
22 {
23     for(int v:adj[u])
24     {
25         if(v == p || did[v])
26             continue;
27         if(2*sz[v] >= sub_sz)
28             return centroid(v, u, sub_sz);
29     }
30     return u;
31 }
32 // SOLVE PROBLEM
33 int k, cs = 1;
34 ll mp[N], vis[N];
35 void push(int x)
36 {
37     if(vis[x] != cs)
38     {
39         vis[x] = cs;
40         mp[x] = 0;
41     }
42 }
43 ll calc(int u, int p,int dep)
44 {
45     ll rs = 0;
46     if(dep <= k){
47         push(k-dep);
48         rs = mp[k-dep];
49     }
50     for(int v:adj[u])
51     {
52         if(v == p || did[v])
53             continue;
54         rs += calc(v, u, dep+1);
55     }
56     return rs;
57 }
58 void put(int u, int p,int dep)
59 {
60     push(dep);
61     mp[dep]++;
62     for(int v:adj[u])
63     {
64         if(v == p || did[v])
65             continue;
66         put(v, u, dep+1);
67     }
68 }
69
70 //DECOMPOSITION
71 ll decompose(int u)

```

```

72 {
73     dfs(u, u);
74     u = centroid(u, u, sz[u]);
75
76     //SOLVE PROBLEM
77     cs++;
78     ll rs = 0;
79     for(int v:adj[u])
80     {
81         if(did[v])
82             continue;
83         rs += calc(v, u, 1);
84         put(v, u, 1);
85     }
86     push(k);
87     rs += mp[k];
88
89     //PROPAGATE DECOMPOSITION
90     did[u] = 1;
91
92     for(int v:adj[u])
93     {
94         if(did[v])
95             continue;
96         rs += decompose(v);
97     }
98     return rs;
99 }
100
101 int main()
102 {
103     ios::sync_with_stdio(false);
104     cin.tie(NULL);
105     int n;
106     cin >> n >> k;
107
108     for(int i = 1; i < n; i++)
109     {
110         int u, v;
111         cin >> u >> v;
112         adj[u].push_back(v);
113         adj[v].push_back(u);
114     }
115     cout << decompose(1) << '\n';
116     return 0;
117 }

```

Version that Sabino likes

```

1  /*
2      Indexado de 1
3  */
4  struct Centroid{
5      // o pai de cada cara no centroide
6      vector<int> p;
7      // visitado para construir o centroide
8      vector<bool> vis;
9      // tam da subarvore
10     vector<int> tam;
11     // grafo original
12     vector<vector<int>> graph;
13     int n;
14     // raiz do centroide
15     int r;
16
17     // passa com o grafo ja lido
18     Centroid(vector<vector<int>> &graph, int n){

```

```

19     this->graph = graph;
20     tam.resize(n+10);
21     vis.resize(n+10);
22     p.resize(n+10);
23     this->n = n;
24     r=build(1);
25 }
26
27 // constroi o centroide
28 int build(int u){
29     set_tam(u,u);
30     while(1){
31         int flag=1;
32         for(auto v: graph[u]){
33             if(vis[v]) continue;
34             if(tam[v]*2 > tam[u]){
35                 tam[u]-=tam[v];
36                 tam[v]+=tam[u];
37                 u=v;
38                 flag=0;
39                 break;
40             }
41         }
42         if(flag) break;
43     }
44     vis[u]=1;
45     p[u]=u;
46     for(auto v: graph[u]){
47         if(vis[v]) continue;
48         int x=build(v);
49         p[x]=u;
50     }
51     return u;
52 }
53
54 void set_tam(int u,int ant){
55     tam[u]=1;
56     for(auto v: graph[u]){
57         if(v == ant || vis[v]) continue;
58         set_tam(v,u);
59         tam[u]+=tam[v];
60     }
61 }
62 };

```

Example 1: Given a tree of n nodes, your task is to count the number of distinct paths that consist of exactly k edges.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 const int N = 400100;
5 vector<int> adj[N];
6 int sz[N], did[N];
7 int pass = 0;
8 void dfs(int u,int p )
9 {
10     sz[u] = 1;
11     // cout << pass++ << '\n';
12     for(int v:adj[u])
13     {
14         if(v == p || did[v])
15             continue;
16         dfs(v, u);
17         sz[u] += sz[v];
18     }
19 }

```

```

20
21 int centroid(int u,int p, int sub_sz)
22 {
23     for(int v:adj[u])
24     {
25         if(v == p || did[v])
26             continue;
27         if(2*sz[v] >= sub_sz)
28             return centroid(v, u, sub_sz);
29     }
30     return u;
31 }
32 int k, cs = 1;
33 int cont[N], vis[N];
34 void push(int x)
35 {
36     if(vis[x] != cs)
37     {
38         vis[x] = cs;
39         cont[x] = 0;
40     }
41 }
42 int calc(int u, int p,int dep)
43 {
44     ll rs = 0;
45     if(dep <= k){
46         push(k-dep);
47         rs = cont[k-dep];
48     }else
49         return 0;
50     for(int v:adj[u])
51     {
52         if(v == p || did[v])
53             continue;
54         rs += calc(v, u, dep+1);
55     }
56     return rs;
57 }
58 void put(int u, int p,int dep)
59 {
60     push(dep);
61     cont[dep]++;
62     if(dep > k)
63         return;
64     for(int v:adj[u])
65     {
66         if(v == p || did[v])
67             continue;
68         put(v, u, dep+1);
69     }
70 }
71 ll decomp(int u)
72 {
73     // cout << u << " <<< \n";
74     dfs(u, u);
75
76     // cout << u << " <<< \n";
77     u = centroid(u, u, sz[u]);
78     cs++;
79     ll rs = 0;
80     for(int v:adj[u])
81     {
82         if(did[v])
83             continue;
84         rs += calc(v, u, 1);
85         put(v, u, 1);

```

```

86     }
87     push(k);
88     rs += cont[k];
89     if(rs == 0)
90         return 0;
91     // cout << u << ' ' << rs << '\n';
92     did[u] = 1;
93     for(int v:adj[u])
94     {
95         if(did[v])
96             continue;
97         rs += decomp(v);
98     }
99     return rs;
100 }
101
102 int main()
103 {
104     ios::sync_with_stdio(false);
105     cin.tie(NULL);
106     int n;
107     cin >> n >> k;
108
109     for(int i = 1; i < n; i++)
110     {
111         int u, v;
112         cin >> u >> v;
113         adj[u].push_back(v);
114         adj[v].push_back(u);
115     }
116     cout << decomp(1) << '\n';
117     return 0;
118 }

```

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.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 const int N = 400100;
5 vector<int> adj[N];
6 int sz[N], did[N];
7 int pass = 0;
8 void dfs(int u,int p )
9 {
10     sz[u] = 1;
11     // cout << pass++ << '\n';
12     for(int v:adj[u])
13     {
14         if(v == p || did[v])
15             continue;
16         dfs(v, u);
17         sz[u] += sz[v];
18     }
19 }
20
21 int centroid(int u,int p, int sub_sz)
22 {
23     for(int v:adj[u])

```

```

24     {
25         if(v == p || did[v])
26             continue;
27         if(2*sz[v] >= sub_sz)
28             return centroid(v, u, sub_sz);
29     }
30     return u;
31 }
32 int k, cs = 1;
33 map<pair<int,int>, int> dist;
34 int best[N], pai[N];
35 void dfs_calc(int u, int p, int orig, int dep)
36 {
37     dist[{orig, u}] = dep;
38     for(int v:adj[u])
39     {
40         if(v == p || did[v])
41             continue;
42         dfs_calc(v, u, orig, dep+1);
43     }
44 }
45
46 void decomp(int u, int p)
47 {
48     dfs(u, u);
49     int at = centroid(u, u, sz[u]);
50     pai[at] = p;
51     dfs_calc(at, at, at, 0);
52     if(u == p)
53         pai[at] = at;
54     // cout << at << ' ' << pai[at] << "<<<\n";
55     did[at] = 1;
56     for(int v:adj[at])
57     {
58         if(did[v])
59             continue;
60         decomp(v, at);
61     }
62 }
63
64
65 int main()
66 {
67     ios::sync_with_stdio(false);
68     cin.tie(NULL);
69     int n, m;
70     cin >> n >> m;
71     for(int i = 1; i <= n; i++)
72         best[i] = n+1;
73     for(int i = 1; i < n; i++)
74     {
75         int u, v;
76         cin >> u >> v;
77         adj[u].push_back(v);
78         adj[v].push_back(u);
79     }
80     decomp(1, 1);
81     int u = 1;
82     best[u] = 0;
83     int x = u;
84     while(pai[u] != u)
85     {
86         u = pai[u];
87         best[u] = min(best[u], dist[{u, x}]);
88     }
89     while(m--)

```

```

90 {
91     int t, u;
92     cin >> t >> u;
93     // cout << t << ' ' << u << '\n';
94     if(t == 2)
95     {
96         int rs = best[u];
97         int x = u;
98         while(pai[u] != u)
99         {
100             u = pai[u];
101             rs = min(rs, best[u] + dist[{u,x}]);
102         }
103         cout << rs << '\n';
104     }else
105     {
106         best[u] = 0;
107         int x = u;
108         while(pai[u] != u)
109         {
110             u = pai[u];
111             best[u] = min(best[u], dist[{u, x}]);
112         }
113     }
114 }
115 return 0;
116 }

```

8.7 Tree Isomorphism

In this code, we generate an id for each tree. It is also possible to do with subtrees.

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 map<vector<int>, int> hasher;
5 int hashify(vector<int> x) {
6     sort(all(x));
7     if(!hasher[x]) {
8         hasher[x] = hasher.size();
9     }
10    return hasher[x];
11 }
12 class Tree
13 {
14     public:
15     vector<vector<int>> adj;
16     vector<int> sub, pai;
17     int r[2], hs[2], n, bt = 0;
18     Tree(int x): n(x)
19     {
20         adj.resize(n+1);
21         sub.resize(n+1, 0);
22         pai.resize(n+1);
23     }
24     void add_edge(int u, int v)
25     {
26         adj[u].pb(v);
27         adj[v].pb(u);
28     }
29     int dfs(int u, int p = -1)
30     {
31         vector<int> at;
32         for(int v:adj[u])
33         {

```



```

34         if(v == p)
35             continue;
36         at.pb(dfs(v, u));
37     }
38     return hashify(at);
39 }
40 void dfs_sub(int u, int p = -1)
41 {
42     pai[u] = p;
43     sub[u] = 1;
44     for(int v:adj[u])
45     {
46         if(v == p)
47             continue;
48         dfs_sub(v, u);
49         sub[u] += sub[v];
50     }
51 }
52 int find_centroid(int u, int p = -1)
53 {
54     // cout << u << '\n';
55     for(int v:adj[u])
56     {
57         if(v == p)
58             continue;
59         if(2*sub[v] > n)
60             return find_centroid(v, u);
61     }
62     return u;
63 }
64 void build()
65 {
66     bt = 1;
67     dfs_sub(1);
68     r[1] = r[0] = find_centroid(1);
69     for(int v:adj[r[0]])
70     {
71         if(v == pai[r[0]])
72         {
73             if(2*(n-sub[r[0]]) >= n)
74                 r[1] = v;
75         }else
76         {
77             if(2*sub[v] >= n)
78                 r[1] = v;
79         }
80     }
81     hs[0] = dfs(r[0]);
82     hs[1] = dfs(r[1]);
83 }
84
85 bool isIsomorphic(Tree t2)
86 {
87     if(!bt)
88         build();
89     if(!t2.bt)
90         t2.build();
91     rep(i, 0, 2)
92     {
93         rep(j, 0, 2)
94         {
95             if(hs[i] == t2.hs[j]){
96                 // cout << r[i] << ' ' << t2.r[j] << "<<<\n";
97                 return true;
98             }
99         }

```

```

100         }
101     }
102     return false;
103 }
104 };
105
106
107
108
109
110
111
112 void test()
113 {
114     int n;
115     cin >> n;
116     Tree t1(n), t2(n);
117     rep(i, 0, n-1)
118     {
119         int u, v;
120         cin >> u >> v;
121         t1.add_edge(u, v);
122     }
123     rep(i, 0, n-1)
124     {
125         int u, v;
126         cin >> u >> v;
127         t2.add_edge(u, v);
128     }
129
130     if(t1.isIsomorphic(t2))
131         cout << "YES\n";
132     else
133         cout << "NO\n";
134     // hasher.clear();
135 }
136
137
138 int32_t main()
139 {
140     ios::sync_with_stdio(false);
141     cin.tie(NULL);
142     int t = 1;
143     cin >> t;
144     while(t--)
145         test();
146     return 0;
147 }

```

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.

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 //template
6
7 const int N = 400100;
8 const int K = 20;
9 vector<int> adj[N];
10 int pai[N][K], in[N], out[N], cs = 0;
11

```

```

12 void pre_dfs(int u, int p)
13
14
15 // u is ancestor of v ?
16 bool is_ancestor(int u, int v)
17
18 int calc_lca(int a, int b)
19
20
21 int cor[N];
22 vector<int> c[N];
23 vector<int> nadj[N];
24
25 int lc[N];
26
27 ll dp[N][3];
28
29 ll calc(int u, int col, int p)
30 {
31     dp[u][1] = (cor[u] == col);
32     dp[u][2] = 0;
33     for(int v:nadj[u])
34     {
35         if(v == p)
36             continue;
37         calc(v, col, u);
38         dp[u][2] += dp[u][1]*dp[v][1];
39         if(cor[u] != col){
40             dp[u][1] += dp[v][1];
41         }
42         dp[u][2] += dp[v][2];
43     }
44     // cout << u << ' ' << dp[u][2] << '\n';
45     return dp[u][2];
46 }
47
48
49 ll solve(vector<int> &at, int j)
50 {
51     // solve for tree with one vertex
52     if(at.size() <= 1)
53         return 0;
54     sort(all(at), [&](int a, int b){
55         return in[a] < in[b];
56     });
57     int m = at.size();
58     rep(i, 0, m)
59         lc[i] = at[i];
60     rep(i, 0, m-1){
61         lc[i+m] = calc_lca(at[i], at[i+1]);
62     }
63     sort(lc, lc+m+m-1, [&](int a, int b){
64         return in[a] < in[b];
65     });
66     m = unique(lc, lc+m+m-1) - lc;
67     rep(i, 0, m)
68         nadj[lc[i]].clear();
69
70     vector<int> st;
71     st.pb(lc[0]);
72     rep(i, 1, m)
73     {
74         while(!is_ancestor(st.back(), lc[i]))
75             st.pop_back();
76         nadj[st.back()].pb(lc[i]);
77         nadj[lc[i]].pb(st.back());

```

```

78         st.pb(lc[i]);
79     }
80     // solve for virtual tree
81     return calc(lc[0],j,lc[0]);
82 }
83
84 void test()
85 {
86     int n;
87     cin >> n;
88
89     rep(i, 1, n+1){
90         adj[i].clear();
91         c[i].clear();
92     }
93     cs = 0;
94     rep(i, 1, n+1)
95     {
96         cin >> cor[i];
97         c[cor[i]].pb(i);
98     }
99     rep(i, 0, n-1)
100    {
101        int u, v;
102        cin >> u >> v;
103        adj[u].pb(v);
104        adj[v].pb(u);
105    }
106    pre_dfs(1,1);
107    ll rs = 0;
108    rep(i, 1, n+1)
109    {
110        rs += solve(c[i],i);
111    }
112    cout << rs << '\n';
113 }
114
115 int32_t main(){
116     ios_base::sync_with_stdio(false);
117     cin.tie(NULL);
118
119     int t = 1;
120     cin >> t;
121     while(t--){
122         test();
123     }
124     return 0;
125 }

```

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.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Node {

```

```

5   int x;
6   Node *l = 0;
7   Node *r = 0;
8   Node *p = 0;
9   bool rev = false;
10
11  Node() = default;
12
13  Node(int v) { x = v; }
14
15  void push() {
16      if (rev) {
17          rev = false;
18          swap(l, r);
19          if (l) l->rev ^= true;
20          if (r) r->rev ^= true;
21      }
22  }
23
24  bool is_root() { return p == 0 || (p->l != this && this != p->r); }
25 };
26
27 struct LCT {
28     vector<Node> a;
29
30     LCT(int n) {
31         a.resize(n + 1);
32         for (int i = 1; i <= n; ++i) a[i].x = i;
33     }
34
35     void rot(Node *c) {
36         auto p = c->p;
37         auto g = p->p;
38
39         if (!p->is_root()) (g->r == p ? g->r : g->l) = c;
40
41         p->push();
42         c->push();
43
44         if (p->l == c) { // rtr
45             p->l = c->r;
46             c->r = p;
47             if (p->l) p->l->p = p;
48         } else { // rtl
49             p->r = c->l;
50             c->l = p;
51             if (p->r) p->r->p = p;
52         }
53
54         p->p = c;
55         c->p = g;
56     }
57
58     void splay(Node *c) {
59         while (!c->is_root()) {
60             auto p = c->p;
61             auto g = p->p;
62             if (!p->is_root()) rot((g->r == p) == (p->r == c) ? p : c);
63             rot(c);
64         }
65         c->push();
66     }
67
68     Node *access(int v) {
69         Node *last = 0;
70         Node *c = &a[v];

```

```

71     for (Node *p = c; p; p = p->p) {
72         splay(p);
73         p->r = last;
74         last = p;
75     }
76     splay(c);
77     return last;
78 }
79
80 void make_root(int v) {
81     access(v);
82     auto *c = &a[v];
83     if (c->l) c->l->rev ^= true, c->l = 0;
84 }
85
86 void link(int u, int v) {
87     make_root(v);
88     Node *c = &a[v];
89     c->p = &a[u];
90 }
91
92 void cut(int u, int v) {
93     make_root(u);
94     access(v);
95     if (a[v].l) {
96         a[v].l->p = 0;
97         a[v].l = 0;
98     }
99 }
100
101 bool connected(int u, int v) {
102     access(u);
103     access(v);
104     return a[u].p;
105 }
106 };
107
108 int main() {
109     int n;
110     int m;
111     cin >> n >> m;
112     LCT lc(n);
113
114     for (int i = 0; i < m; i++) {
115         string a;
116         cin >> a;
117         int b, c;
118         cin >> b >> c;
119         if (a == "add") { lc.link(b, c); }
120         if (a == "rem") { lc.cut(b, c); }
121         if (a == "conn") { cout << (lc.connected(b, c) ? "YES" : "NO") << "\n"; }
122     }
123 }

```

Capítulo IX

Geometry

9.1 Simple Geometry

This section will have a lot 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. you can change the type from `ld` to `ll` and also change the `EPS`. The rest is pretty much fixed.

```
1  const ld EPS = 1e-9;
2  const ld PI = acosl(-1);
3
4  using T = ll;
5  // using T = ld;
6  typedef pair<T,T> pt;
7  typedef pair<pt,pt> line;
8  typedef vector<pt> vpt;
9  typedef pair<pt, T> circle;
10
11 pt operator-(pt a) { return mp(-a.ft,-a.sd); }
12 pt &operator+=(pt &a, pt b) {
13     a.ft += b.ft;
14     a.sd += b.sd;
15     return a;
16 }
17 pt &operator-=(pt &a, pt b) {
18     a.ft -= b.ft;
19     a.sd -= b.sd;
20     return a;
21 }
22 pt &operator*=(pt &a, T r) {
23     a.ft *= r;
24     a.sd *= r;
25     return a;
26 }
27 pt &operator/=(pt &a, T r) {
28     a.ft /= r;
29     a.sd /= r;
30     return a;
31 }
32
33 pt operator+(pt a, pt b) { return a += b; }
34 pt operator-(pt a, pt b) { return a -= b; }
35 pt operator*(pt a, T r) {return a*=r;}
36 pt operator*(T r, pt a) {return a*=r;}
37 pt operator/(pt a, T r) {return a/=r;}
38 int sgn(T a) { return (a>EPS)-(a<-EPS); }
39 T sq(T a) { return a*a; }
40 T abs2(pt p) { return sq(p.ft)+sq(p.sd); }
41 ld abs(pt p) { return sqrtl(abs2(p)); }
42 pt unit(pt p) { return p/abs(p); }
43 ld arg(pt p) { return atan2(p.sd,p.ft); }
```

```

44 pt conj(pt p) { return mp(p.ft,-p.sd); }
45 pt perp(pt p) { return mp(-p.sd,p.ft); }
46 pt dir(T ang) { return mp(cos(ang),sin(ang)); }
47
48
49 pt operator*(pt a, pt b) {return pt(a.ft*b.ft-a.sd*b.sd,a.sd*b.ft+a.ft*b
    .sd);}
50
51 pt operator/(pt a, pt b) {return a*conj(b)/abs2(b);}
52
53 pt reflect(const pt& p, const line& l) {
54     pt a = l.ft, d = l.sd-l.ft;
55     return a+conj((p-a)/d)*d; }
56 pt foot(const pt& p, const line& l) {
57     return (p+reflect(p,l))/(T)2; }
58
59 T dot(pt p1, pt p2)
60 {
61     return p1.ft*p2.ft+p1.sd*p2.sd;
62 }
63
64 T dot(pt p1, pt p2, pt p3)
65 {
66     return dot(p2-p1, p3-p1);
67 }
68
69 T cross(pt p1, pt p2)
70 {
71     return p1.ft*p2.sd-p1.sd*p2.ft;
72 }
73
74 T cross(pt p1, pt p2, pt p3)
75 {
76     return cross(p2-p1, p3-p1);
77 }
78
79 // rotate a 90 degrees centered in c
80 pt rotate90(pt a, pt c = mp(0,0))
81 {
82     a-=c;
83     swap(a.ft,a.sd);
84     a.ft = -a.ft;
85     a+=c;
86     return a;
87 }
88
89 // rotate a d degrees centered in c
90 pt rotate(pt a, long double d, pt c = mp(0,0))
91 {
92     a-=c;
93     pt b;
94     b.ft = a.ft*cos(d) - a.sd*sin(d);
95     b.sd = a.sd*cos(d) + a.ft*sin(d);
96     b+=c;
97     return b;
98 }
99
100 bool onseg(pt p, line l) {
101     return sgn(cross(l.ft,l.sd,p)) == 0 && sgn(dot(p,l.ft,l.sd)) <= 0;}
102
103 // distance between a line and a point, needs two points which are in
    the line
104 ld linedist(line lin, pt at)
105 {
106     ld rs = abs(cross(at, lin.ft, lin.sd));
107     rs/=abs(lin.ft-lin.sd);

```



```

108     return rs;
109 }
110
111 // distance between a segment and a point
112 ld point_seg(line seg, pt at)
113 {
114     if(dot(seg.ft, at, seg.sd) <= 0)
115         return abs(at-seg.ft);
116     if(dot(seg.sd, at, seg.ft) <= 0)
117         return abs(at-seg.sd);
118     return linedist(seg, at);
119 }
120
121 // Usage: vpt v; sort(all(v),angleCmp);
122 // used to sort points int ccw around the origin
123 int half(pt x) { return x.sd != 0 ? sgn(x.sd) : -sgn(x.ft); }
124 bool angleCmp(pt a, pt b) { int A = half(a), B = half(b);
125     return A == B ? cross(a,b) > 0 : A < B; }
126
127 // equivalent to: sort(all(v),[](pt a, pt b) {
128 //     return atan2(a.sd,a.ft) < atan2(b.sd,b.ft); });
129
130 // {unique intersection point} if it exists
131 // {b.f,b.s} if input lines are the same
132 // empty if lines do not intersect
133 vpt lineIsect(const line& a, const line& b) {
134     T a0 = cross(a.ft,a.sd,b.ft), a1 = cross(a.ft,a.sd,b.sd);
135     if (a0 == a1) return a0 == 0 ? vpt{b.ft,b.sd} : vpt{};
136     return {(b.sd*a0-b.ft*a1)/(a0-a1)};
137 }
138
139 // point in interior of both segments a and b, if it exists
140 vpt strictIsect(const line& a, const line& b) {
141     T a0 = cross(a.ft,a.sd,b.ft), a1 = cross(a.ft,a.sd,b.sd);
142     T b0 = cross(b.ft,b.sd,a.ft), b1 = cross(b.ft,b.sd,a.sd);
143     if (sgn(a0)*sgn(a1) < 0 && sgn(b0)*sgn(b1) < 0)
144         return {(b.sd*a0-b.ft*a1)/(a0-a1)};
145     return {};
146 }
147
148 // intersection of segments, a and b may be degenerate
149 vpt segIsect(const line& a, const line& b) {
150     vpt v = strictIsect(a,b); if (!v.empty()) return v;
151     set<pt> s;
152     #define i(x,y) if (onseg(x,y)) s.insert(x)
153     i(a.ft,b); i(a.sd,b); i(b.ft,a); i(b.sd,a);
154     return {all(s)};
155 }

```

9.1.2 Circles

This section will have a lot of algorithms related to circles and their description. A circle will have the following format:

```

1     typedef pair<pt, T> circle;
2

```

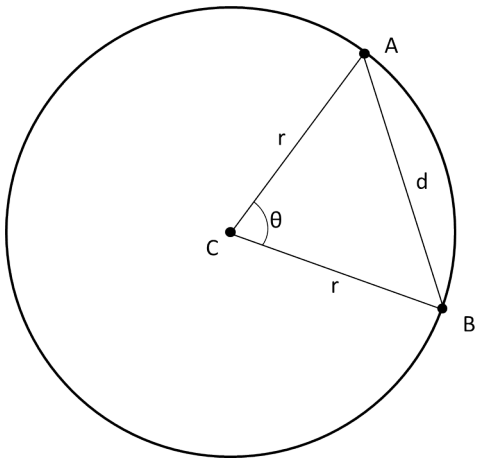
To see if a point is in/on/out a circle we can do:

```

1 // -1 if inside, 0 if in border, 1 if outside
2 int in(const circle& x, const pt& y) {
3     return sgn(abs(y-x.ft)-x.sd);
4 }
5

```

Arc length of two points:



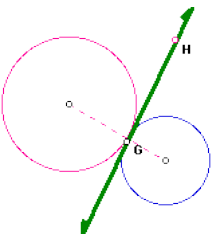
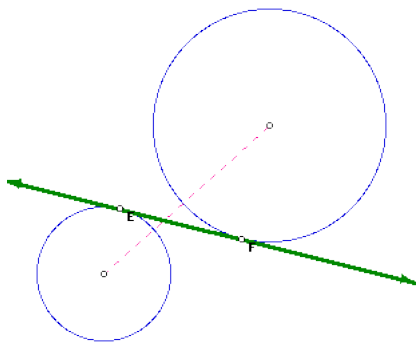
```
1 T arcLength(const circle& x, pt a, pt b) {
2     // precondition: a and b on x
3     pt d = (a-x.ft)/(b-x.ft); return x.sd*acos(d.ft);
4 }
5
```

Intersections related to circles:

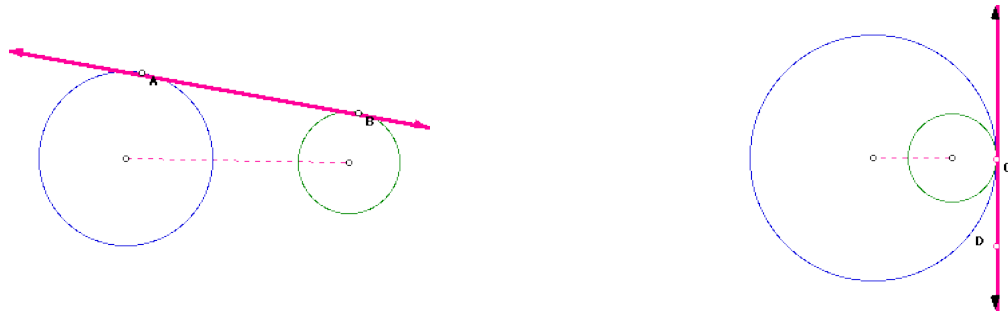
```
1 vpt isect(const circle& x, const circle& y) { // precondition: x!=y
2     T d = abs(x.ft-y.ft), a = x.sd, b = y.sd;
3     if (sgn(d) == 0) { assert(a != b); return {}; }
4     T C = (a*a+d*d-b*b)/(2*a*d);
5     if (abs(C) > 1+EPS) return {};
6     T S = sqrt(max(1-C*C,(T)0));
7     pt tmp = (y.ft-x.ft)/d*x.sd;
8     return {x.ft+tmp*pt(C,S),x.ft+tmp*pt(C,-S)};
9 }
10
11 vpt isect(const circle& x, const line& y) {
12     pt c = foot(x.ft,y);
13     T sq_dist = sq(x.sd)-abs2(x.ft-c);
14     if (sgn(sq_dist) < 0) return {};
15     pt offset = unit(y.sd-y.ft)*sqrt(max(sq_dist,T(0)));
16     return {c+offset,c-offset};
17 }
18
19 T isect_area(circle x, circle y) { // not thoroughly tested
20     T d = abs(x.ft-y.ft), a = x.sd, b = y.sd;
21     if (a < b) swap(a,b);
22     if (d >= a+b) return 0;
23     if (d <= a-b) return PI*b*b;
24     T ca = (a*a+d*d-b*b)/(2*a*d), cb = (b*b+d*d-a*a)/(2*b*d);
25     T s = (a+b+d)/2, h = 2*sqrt(s*(s-a)*(s-b)*(s-d))/d;
26     return a*a*acos(ca)+b*b*acos(cb)-d*h;
27 }
28
```

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.

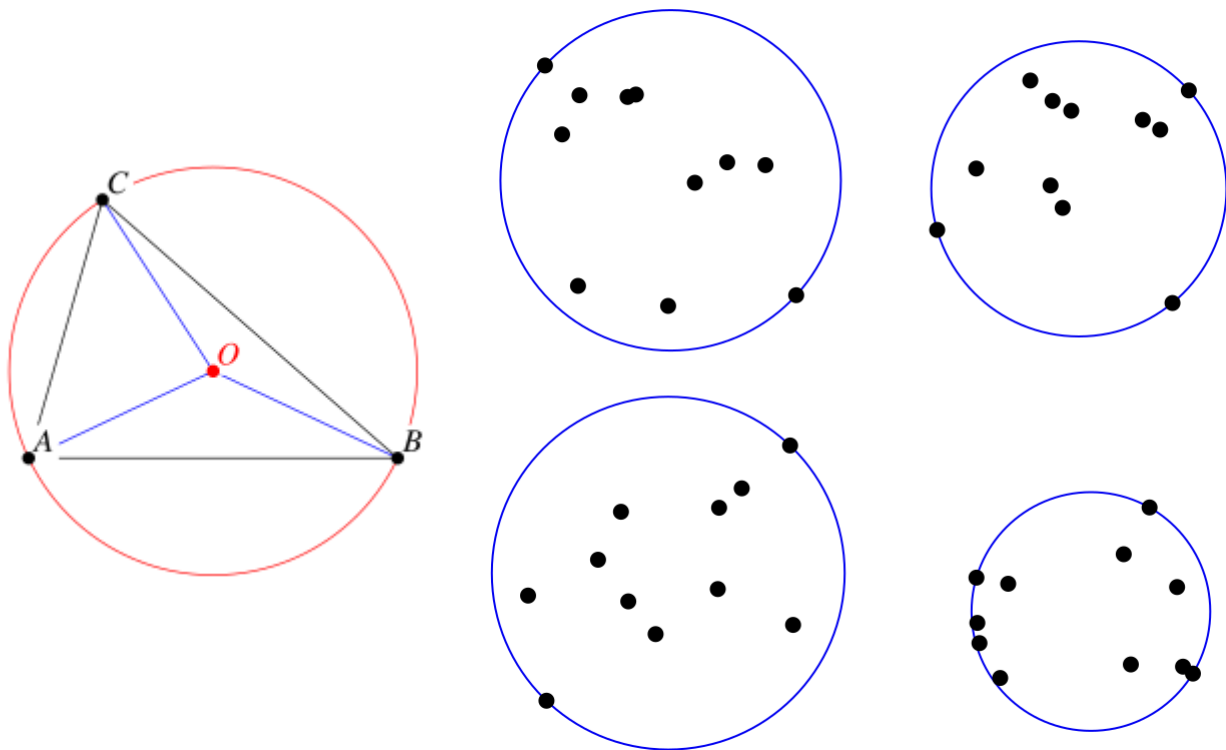


A tangent of two circles is a common external tangent if the intersection of the tangent and the line segment joining the centers is empty.



```
1 pt tangent(pt x, circle y, int t = 0) {
2   y.sd = abs(y.sd); // abs needed because internal calls y.s < 0
3   if (y.sd == 0) return y.ft;
4   T d = abs(x-y.ft);
5   pt a = pow(y.sd/d,2)*(x-y.ft)+y.ft;
6   pt b = sqrt(d*d-y.sd*y.sd)/d*y.sd*unit(x-y.ft)*dir(PI/2);
7   return t == 0 ? a+b : a-b;
8 }
9 vector<pair<pt,pt>> external(circle x, circle y) {
10  vector<pair<pt,pt>> v;
11  if (x.sd == y.sd) {
12    pt tmp = unit(x.ft-y.ft)*x.sd*dir(PI/2);
13    v.eb(x.ft+tmp,y.ft+tmp);
14    v.eb(x.ft-tmp,y.ft-tmp);
15  } else {
16    pt p = (y.sd*x.ft-x.sd*y.ft)/(y.sd-x.sd);
17    rep(i, 0, 2) v.eb(tangent(p,x,i),tangent(p,y,i));
18  }
19  return v;
20 }
21 vector<pair<pt,pt>> internal(circle x, circle y) {
22   return external({x.ft,-x.sd},y); }
23
```

The CircumCenter of a triangle is the minimum enclosing circle of a triangle. We can also calculated the minimum enclosing circle of a polygon.



```
1 // return the minimum enclosing circle of a triangle
2 circle ccCenter(pt a, pt b, pt c) {
3   b -= a; c -= a;
4   pt res = b*c*(conj(c)-conj(b))/(b*conj(c)-conj(b)*c);
```

```
5     return {a+res,abs(res)};
6 }
7
8 // return the minimum enclosing circle of a set of points O(N)
9 circle mec(vpt ps) {
10     shuffle(all(ps), rng);
11     pt o = ps[0];
12     T r = 0, EPS = 1+1e-8;
13     rep(i, 0, (int)ps.size())
14         if (abs(o-ps[i]) > r*EPS) {
15             o = ps[i], r = 0; // point is on MEC
16             rep(j, 0, i)
17                 if (abs(o-ps[j]) > r*EPS) {
18                     o = (ps[i]+ps[j])/2, r = abs(o-ps[i]);
19                     rep(k, 0, j) if (abs(o-ps[k]) > r*EPS)
20                         tie(o,r) = ccCenter(ps[i],ps[j],ps[k]);
21                 }
22         }
23     return {o,r};
24 }
25
```

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

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 typedef pair<int, int> Point;
5
6 int cross(Point a, Point b, Point c) {
7     return a.first * (b.second - c.second)
8         + b.first * (c.second - a.second)
9         + c.first * (a.second - b.second);
10 }
11
12 const int N = 100100;
13 pair<int, int> p[N], p2[N], upper[N], lower[N];
14 int uppersize = 0, lowersize = 0;
15
16 void convexHull(int n) {
```

```

17  sort(p, p+n);
18  uppersize = 0;
19  lowersize = 0;
20
21  for (int i = 0; i < n; i++) {
22      while (uppersize >= 2 && cross(upper[uppersize-2], upper[uppersize
23          -1], p[i]) >= 0) {
24          uppersize--;
25      }
26      upper[uppersize++] = p[i];
27  }
28
29  for (int i = 0; i < n; i++) {
30      while (lowersize >= 2 && cross(lower[lowersize-2], lower[lowersize
31          -1], p[i]) <= 0) {
32          lowersize--;
33      }
34      lower[lowersize++] = p[i];
35  }
36 }
37
38 double dist(Point a, Point b) {
39     return hypot(a.first - b.first, a.second - b.second);
40 }
41
42 int main() {
43     ios::sync_with_stdio(false);
44     cin.tie(nullptr);
45
46     int c;
47
48     cin >> c;
49
50     for (int i = 0; i < c; i++) {
51         cin >> p2[i].first >> p2[i].second;
52     }
53
54     sort(p2, p2+c);
55
56     int n = 0;
57
58     p[n++] = p2[0];
59
60     for (int i = 1; i < c; i++) {
61         if (p2[i] != p2[i-1]) {
62             // cout << "adding " << p2[i].first << " " << p2[i].second << endl
63             ;
64             p[n++] = p2[i];
65         }
66     }
67
68     // cout << "n: " << n << endl;
69     convexHull(n);
70
71     // cout << "lower: \n";
72     // for (int i = 0; i < lowersize; i++) {
73     //     cout << lower[i].first << ", " << lower[i].second << endl;
74     // }
75
76     // cout << "upper: \n";
77     // for (int i = 0; i < uppersize; i++) {
78     //     cout << upper[i].first << ", " << upper[i].second << endl;
79     // }

```

```

80  int i = 0, j = lowersize-1;
81  double ans = 0;
82
83
84  while (i < uppersize-1 || j > 0) {
85      // cout << "(" << i << ")" " << upper[i].first << ", " << upper[i].
      second << " dist to "
86      // << "(" << j << ")" " << lower[j].first << ", " << lower[j].second
87      // << " = " << dist(upper[i], lower[j]) << endl;
88
89      ans = max(ans, dist(upper[i], lower[j]));
90
91      if (i == uppersize-1) {
92          j--;
93      } else if (j == 0) {
94          i++;
95      } else {
96          if ((upper[i+1].second - upper[i].second) * (lower[j].first -
97              lower[j-1].first)
98              > (lower[j].second - lower[j-1].second) * (upper[i+1].first -
99              upper[i].first)) {
100              i++;
101          } else {
102              j--;
103          }
104      }
105
106      cout << setprecision(10) << fixed;
107      cout << ans << "\n";
108
109      return 0;
110  }

```

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. $O(n \log n)$. (Original Problem: CODECHEF - TRIANGULAR QUERIES)

```

1  #include <bits/stdc++.h>
2
3  #define getcx getchar_unlocked
4  #define pc(x) putchar_unlocked(x)
5
6  using namespace std;
7
8  typedef pair<int, int> pii;
9  typedef long double ld;
10 typedef long long ll;
11 typedef unsigned long long ull;
12
13 inline int readInt()
14 {
15     bool minus = false;
16     register ll result = 0;
17     register char ch = getchar_unlocked();
18     while (true)
19     {
20         if (ch == '-') break;
21         if (ch >= '0' && ch <= '9') break ;
22         ch = getchar_unlocked();
23     }
24     if (ch == '-') minus = true;

```

```

25     else result = ch-'0';
26     while (true)
27     {
28         ch = getchar_unlocked();
29         if (ch < '0' || ch > '9') break;
30         result = result*10LL + (ch - '0');
31     }
32     if (minus) return -result;
33     else return result;
34 }
35
36 inline void writeInt ( int n )
37 {
38     register ll N = n, rev, count = 0 ;
39     rev = N ;
40     if (N == 0)
41     {
42         pc('0'); return ;
43     }
44     while ((rev % 10LL) == 0LL)
45     {
46         count++; rev /= 10LL;
47     } //obtain the count of the number of 0s
48     rev = 0;
49     while (N != 0LL)
50     {
51         rev = (rev<<3LL) + (rev<<1LL) + N % 10LL; N /= 10LL;
52     } //store reverse of N in rev
53     while (rev != 0LL)
54     {
55         pc(rev % 10LL + '0'); rev /= 10LL ;
56     }
57     while (count--) pc('0') ;
58 }
59
60 inline void write_string(char *str)
61 {
62     register char c = 0;
63     register int i = 0;
64     while (c < 33)
65         c = getchar_unlocked() ;
66     while (c != '\n')
67     {
68         str[i] = c;
69         c = getchar_unlocked() ;
70         i = i + 1;
71     }
72     str[i] = '\0';
73 }
74
75 class Triangle{
76 public:
77     int x, y, d;
78     int idx;
79
80     bool operator<(Triangle &other){
81         return (x + y) < (other.x + other.y);
82     }
83 };
84
85 enum events_types{points_ap, beg_triangle, end_triangle};
86
87 class Event{
88 public:
89     int x, y;
90     int type;//0 -> POINT EVENT

```

```

91         //1 -> BEGIN TRIANGLE
92         //2 -> END OF A TRIANGLE
93     int idx;
94
95     bool operator<(Event &other){
96         int w1 = x + y;
97         int w2 = other.x + other.y;
98
99         if(w1 < w2){
100             return true;
101         }else if(w1 > w2){
102             return false;
103         }else{
104             return type < other.type;
105         }
106     }
107 };
108
109 class Point{
110     public:
111     int x, y;
112 };
113
114 const int N = int(3e5 + 10);
115 Point points[N];
116 Triangle triangles[N];
117 int bag[N];
118 int BIT_X[N], BIT_Y[N];
119 vector< Event > events;
120
121 int LSOne(int x){
122     return x & (-x);
123 }
124
125 void update(int BIT[], int x, int value){
126     while(x < N){
127         BIT[x] += value;
128         x += LSOne(x);
129     }
130 }
131
132 int query(int BIT[], int x){
133     int sum = 0;
134
135     while(x > 0){
136         sum += BIT[x];
137         x -= LSOne(x);
138     }
139
140     return sum;
141 }
142
143 int rsq(int BIT[], int l, int r){
144     r = min(r, N - 1);
145
146     return query(BIT, r) - query(BIT, l - 1);
147 }
148
149 void run_sweep(int n, int q){
150     for(int i = 0 ; i < n ; i++){
151         events.push_back({points[i].x, points[i].y, points_ap, i});
152     }
153
154     for(int i = 0 ; i < q ; i++){
155         events.push_back({triangles[i].x - 1, triangles[i].y - 1,
156             beg_triangle, i});

```



```

156     events.push_back({triangles[i].x + triangles[i].d, triangles[i].
157     y, end_triangle, i});
158 }
159 sort(events.begin(), events.end());
160
161 int size_active_set = 0;
162
163 for(int i = 0 ; i < events.size() ; i++){
164     if(events[i].type == points_ap || events[i].type == beg_triangle
165 ){
166         if(events[i].x == 0){
167             continue;
168         }
169
170         if(events[i].type == points_ap){
171             size_active_set++;
172             update(BIT_X, events[i].x, 1);
173             update(BIT_Y, events[i].y, 1);
174             // printf("adding %d\n", events[i].idx);
175         }else{
176             bag[events[i].idx] = rsq(BIT_X, 1, events[i].x) + rsq(
177             BIT_Y, 1, events[i].y) - size_active_set;
178             // printf("1: %d->%d\n", events[i].idx, bag[events[i].
179             idx]);
180         }
181         }else{
182             events[i].x -= triangles[events[i].idx].d;
183             bag[events[i].idx] = size_active_set - (rsq(BIT_X, 1, events
184             [i].x - 1) + rsq(BIT_Y, 1, events[i].y - 1) - bag[events[i].idx]);
185             // printf("2: %d->%d\n", events[i].idx, bag[events[i].idx]);
186         }
187     }
188 }
189
190 int main(){
191     ios::sync_with_stdio(false);
192     cin.tie(NULL);
193
194     int n, q;
195
196     n = readInt();
197     q = readInt();
198     // scanf("%d %d", &n, &q);
199
200     for(int i = 0 ; i < n ; i++){
201         points[i].x = readInt();
202         points[i].y = readInt();
203         // scanf("%d %d", &points[i].x, &points[i].y);
204     }
205
206     for(int i = 0 ; i < q ; i++){
207         triangles[i].x = readInt();
208         triangles[i].y = readInt();
209         triangles[i].d = readInt();
210         // scanf("%d %d %d", &triangles[i].x, &triangles[i].y, &
211         triangles[i].d);
212     }
213
214     run_sweep(n, q);
215
216     for(int i = 0 ; i < q ; i++){
217         writeInt(bag[i]);
218         pc('\n');
219         // printf("%d\n", bag[i]);
220     }

```

216

217

218

```
    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 than student b , if all ranks of student a in each contest is lesser than student b . A student is said to be excellent if no other student is better than him. How many excellent students there exists ?

This solution runs in $O(n \log n)$. (Original Problem: SPOJ: NICEDAY)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 typedef pair<int, int> pii;
5 typedef long double ld;
6 typedef long long ll;
7 typedef unsigned long long ull;
8
9 class Rank{
10     public:
11     int a, b, c;
12
13     bool operator <(Rank &other){
14         return this->a < other.a;
15     }
16 };
17
18 const int N = int(1e5 + 10);
19 Rank ranks[N];
20 int BIT[N];
21
22 int LSOne(int x){
23     return x & (-x);
24 }
25
26 int query(int x){
27     int best = INT32_MAX;
28
29     while(x != 0){
30         if(BIT[x] != -1)
31             best = min(best, BIT[x]);
32         x -= LSOne(x);
33     }
34
35     return best;
36 }
37
38 void update(int x, int val){
39     while(x < N){
40         if(BIT[x] == -1){
41             BIT[x] = val;
42         }
43
44         BIT[x] = min(BIT[x], val);
45         x += LSOne(x);
46     }
47 }
48
49 int main(){
50     ios::sync_with_stdio(false);
51     cin.tie(NULL);
52
53     int t;
54
55     cin >> t;
56
57     while(t--){
58         memset(BIT, -1, sizeof BIT);
59

```

```
60     int n;
61
62     cin >> n;
63
64     for(int i = 0 ; i < n ; i++){
65         cin >> ranks[i].a >> ranks[i].b >> ranks[i].c;
66     }
67
68     sort(ranks , ranks + n);
69
70     int cnt = n;
71
72     for(int i = 0 ; i < n ; i++){
73         int q = query(ranks[i].b - 1);
74         // cout << "q: " << q << endl;
75
76         if(q < ranks[i].c){
77             cnt--;
78         }
79
80         update(ranks[i].b, ranks[i].c);
81     }
82
83     cout << cnt << endl;
84 }
85
86 return 0;
87 }
```

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 $O(n \log n)$ if the number of balls is $O(n)$).

This solutions runs in $O(n \log n)$. Original Problem (NCPC 2013 - H).

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 typedef pair< int, int > pii;
6 typedef double ld;
7 typedef long long ll;
8 typedef unsigned long long ull;
9 typedef pair< ll, ll > pll;
10
11 const int N = int(1e5 + 10);
12 pll rectangle[N][2];
13
14 struct event{
15 public:
16     int type;
17     int idx;
18
19     event(){
20
21     }
22
23     event(int tpe, int id){
24         this->type = tpe;
25         this->idx = id;
26     }
27 };
28
29 ll cross(pll a, pll b, pll c){
30     ll dx1 = a.first - b.first, dy1 = a.second - b.second;
31     ll dx2 = c.first - b.first, dy2 = c.second - b.second;
32
33     return dx1 * dy2 - dx2 * dy1;
34 }
35
36 struct MNode{
37 public:
38     int idx;
39
40     MNode(){
41
42     }
43
44     MNode(int id){
45         this->idx = id;
46     }
47
48     bool operator <(MNode other) const{
49         if(rectangle[idx][0].first >= rectangle[other.idx][0].first){
50             ll a = cross(rectangle[other.idx][0], rectangle[idx][0],
51                 rectangle[other.idx][1]);
52
53             return cross(rectangle[other.idx][0], rectangle[idx][0],
54                 rectangle[other.idx][1]) > 0;
55         }else{
56             ll a = cross(rectangle[idx][0], rectangle[other.idx][0],
57                 rectangle[idx][1]);
58
59             return cross(rectangle[idx][0], rectangle[other.idx][0],
60                 rectangle[idx][1]) < 0;
61         }
62     }
63 };

```

```

57         }
58     }
59 };
60
61 vector< event > events;
62 pll balls[N];
63 int adj[N];
64 int adj_balls[N];
65 set< MNode > st;
66 ll resp[N];
67 int prio[] = {0, 2, 1};
68
69 bool compar(event &a, event &b){
70     ll xa, xb;
71     ll ya, yb;
72
73     if(a.type <= 1){
74         xa = rectangle[a.idx][a.type].first;
75         ya = rectangle[a.idx][a.type].second;
76     }else{
77         xa = balls[a.idx].first;
78         ya = balls[a.idx].second;
79     }
80
81     if(b.type <= 1){
82         xb = rectangle[b.idx][b.type].first;
83         yb = rectangle[b.idx][b.type].second;
84     }else{
85         xb = balls[b.idx].first;
86         yb = balls[b.idx].second;
87     }
88
89     if(xa < xb){
90         return true;
91     }else if(xa > xb){
92         return false;
93     }else{
94         if(prio[a.type] == prio[b.type]){
95             if(a.type == 1){
96                 return ya > yb;
97             }
98
99             return ya < yb;
100         }
101
102         return prio[a.type] < prio[b.type];
103     }
104 }
105
106 ll m_point(int u){
107     ll mx;
108
109     if(rectangle[u][0].second > rectangle[u][1].second){
110         mx = rectangle[u][1].first;
111     }else{
112         mx = rectangle[u][0].first;
113     }
114
115     return mx;
116 }
117
118 bool intersect(int u, int v){
119     ll mx = m_point(u);
120
121     return rectangle[v][0].first <= mx && mx <= rectangle[v][1].first;
122 }

```

```

123
124 void line_sweep(){
125     sort(events.begin(), events.end(), compar);
126
127     for(int i = 0 ; i < events.size() ; i++){
128         event u = events[i];
129
130         if(u.type == 2){
131             rectangle[N - 1][0] = rectangle[N - 1][1] = balls[u.idx];
132
133             if(st.size() == 0) continue;
134
135             adj_balls[u.idx] = st.begin()->idx;
136         }else{
137             if(u.type == 1){
138                 auto d = st.find(MNode(u.idx));
139
140                 if(rectangle[u.idx][0].second > rectangle[u.idx][1].
second){
141                     auto nxt = next(d);
142
143                     if(nxt != st.end()){
144                         adj[d->idx] = nxt->idx;
145                     }
146                 }
147                 st.erase(d);
148             }else{
149                 st.insert(MNode(u.idx));
150                 auto d = st.find(MNode(u.idx));
151
152                 if(rectangle[u.idx][0].second < rectangle[u.idx][1].second){
153                     auto nxt = next(d);
154
155                     if(nxt != st.end()){
156                         adj[d->idx] = nxt->idx;
157                     }
158                 }
159             }
160         }
161     }
162
163     st.clear();
164     events.clear();
165 }
166
167 int get_resp(int u){
168     if(resp[u] != -1){
169         return resp[u];
170     }
171
172     if(adj[u] == -1){
173         return m_point(u);
174     }
175
176     return resp[u] = get_resp(adj[u]);
177 }
178
179 int main(){
180     ios::sync_with_stdio(false);
181     cin.tie(NULL);
182
183     memset(adj_balls, -1, sizeof adj_balls);
184     memset(adj, -1, sizeof adj);
185     memset(resp, -1, sizeof resp);
186
187     int n;

```

```
188
189     cin >> n;
190
191     for(int i = 0 ; i < n ; i++){
192         cin >> rectangle[i][0].first >> rectangle[i][0].second >> rectangle[
193             i][1].first >> rectangle[i][1].second;
194
195         if(rectangle[i][1].first < rectangle[i][0].first){
196             swap(rectangle[i][1], rectangle[i][0]);
197         }
198     }
199
200     int m = 1; // m = number of balls
201
202     for(int i = 0 ; i < m ; i++){
203         cin >> balls[i].first;
204
205         balls[i].second = 11(1e9);
206     }
207
208     for(int i = 0 ; i < n ; i++){
209         events.push_back(event(0, i));
210         events.push_back(event(1, i));
211     }
212
213     for(int i = 0 ; i < m ; i++){
214         events.push_back(event(2, i));
215     }
216
217     line_sweep();
218
219     for(int i = 0 ; i < m ; i++){
220         if(adj_balls[i] == -1){
221             cout << balls[i].first << endl;
222         }else{
223             cout << get_resp(adj_balls[i]) << endl;
224         }
225     }
226
227     return 0;
228 }
```


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 $O(\log n)$.

```

1 struct pt{
2     ll x, y;
3
4     pt(){}
5
6     pt(ll _x, ll _y):x(_x), y(_y){}
7
8     pt operator+(pt p){
9         return pt(x + p.x, y + p.y);
10    }
11
12    pt operator-(pt p){
13        return pt(x - p.x, y - p.y);
14    }
15
16    ll cross(pt p){
17        return x * p.y - y * p.x;
18    }
19
20    ll dot(pt p){
21        return x * p.x + y * p.y;
22    }
23
24    ll cross(pt a, pt b){
25        return (a - *this).cross(b - *this);
26    }
27
28    ll dot(pt a, pt b){
29        return (a - *this).dot(b - *this);
30    }
31
32    ll sqrLen(){
33        return this->dot(*this);
34    }
35 };
36
37 bool lexComp(pt l, pt r){
38     return l.x < r.x || (l.x == r.x && l.y < r.y);
39 }
40
41 int sgn(ll val){
42     return val > 0 ? 1 : (val == 0 ? 0 : -1);
43 }
44
45 bool pointsInTriangle(pt a, pt b, pt c, pt point){
46     ll s1 = abs(a.cross(b, c));
47     ll s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs(point.
48         cross(c, a));
49
50     return s1 == s2;
51 }
52
53 bool pointInConvexPolygon(pt point, vector< pt > points){
54     int n = points.size();
55
56     pt p1 = points[1];
57     pt p0 = points[0];
58     pt pn = points[n - 1];

```

```

59     if((p1 - p0).cross(point - p0) != 0 && sgn((p1 - p0).cross(point -
60     p0)) != sgn((p1 - p0).cross(pn - p0))) {
61         return false;
62     }
63     if((pn - p0).cross(point - p0) != 0 && sgn((pn - p0).cross(point -
64     p0)) != sgn((pn - p0).cross(p1 - p0))) {
65         return false;
66     }
67     if((p1 - p0).cross(point - p0) == 0) {
68         return (p1 - p0).sqrLen() >= (point - p0).sqrLen();
69     }
70
71     int lo = 0;
72     int hi = n - 2;
73     int r = -1;
74
75     while(lo <= hi) {
76         int mid = (lo + hi) >> 1;
77
78         if((point - p0).cross(points[mid] - p0) <= 0) {
79             r = mid;
80             lo = mid + 1;
81         } else {
82             hi = mid - 1;
83         }
84     }
85
86     assert(r != -1);
87
88     return pointsInTriangle((points[r] - p0), points[(r + 1) % n] - p0,
89     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 solution runs in $O(n^2 \log n)$. Original Problem (Codechef, REDBLUE - DEC17)

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const double eps = 1e-10;
6
7  struct Point {
8      int x, y, color;
9  } points[2001];
10
11  int M, N;
12  double pi;
13
14  int ComputeMin(int a1, int b1, int a2, int b2) {
15      return min(b1 + a2, a1 + b2);
16  }
17
18  int cnt[2];
19  vector<pair<double, int>> vp;
20
21  int Solve(int idx) {
22      vp.clear();
23      for (int i = 0; i < M + N; ++i) {

```

```

24     if (i == idx) continue;
25     const int dx = points[i].x - points[idx].x, dy = points[i].y -
points[idx].y;
26     assert(dx != 0 || dy != 0);
27     const double u = atan2(dy, dx);
28     vp.push_back({u, i});
29     vp.push_back({u + 2.0 * pi, i});
30 }
31
32 sort(vp.begin(), vp.end());
33
34 int ans = M + N, color_idx = points[idx].color;
35
36 memset(cnt, 0, sizeof(cnt));
37 for (int i = 0, j = 0; i < M + N - 1; ++i) {
38     while (j < int(vp.size()) && vp[j].first - vp[i].first <= pi - eps)
39     {
40         ++cnt[points[vp[j].second].color];
41         ++j;
42     }
43     // idx inside, vp[i].second inside.
44     ++cnt[color_idx];
45     ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));
46
47     // idx inside, vp[i].second outside.
48     --cnt[points[vp[i].second].color];
49     ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));
50
51     // idx outside, vp[i].second outside.
52     --cnt[color_idx];
53     ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));
54
55     // idx outside, vp[i].second inside.
56     ++cnt[points[vp[i].second].color];
57     ans = min(ans, ComputeMin(cnt[0], cnt[1], M - cnt[0], N - cnt[1]));
58
59     // Move vp[i].second outside.
60     --cnt[points[vp[i].second].color];
61 }
62
63 return ans;
64 }
65
66 int main() {
67     pi = 2.0 * acos(0.0);
68     int T;
69     scanf("%d", &T);
70     assert(1 <= T && T <= 100);
71     while (T--) {
72         scanf("%d %d", &M, &N);
73
74         for (int i = 0; i < M; ++i) {
75             scanf("%d %d", &points[i].x, &points[i].y);
76
77             points[i].color = 0;
78         }
79
80         for (int i = M; i < M + N; ++i) {
81             scanf("%d %d", &points[i].x, &points[i].y);
82
83             points[i].color = 1;
84         }
85
86         int ans = M + N;
87         for (int idx = 0; idx < M + N; ++idx) ans = min(ans, Solve(idx));

```

```

88
89     printf("%d\n", ans);
90 }
91 return 0;
92 }

```

9.3.6 Radial Sweep sem Double

You are in a point inside a square $N \times N$. 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.

```

1 #include <bits/stdc++.h>
2 #define x first
3 #define y second
4 typedef long long ll;
5 using namespace std;
6
7 const double PI = 4 * atan(1);
8
9 struct Event {
10     short type, id;
11     pair<ll, ll> loc;
12 };
13
14 pair<ll, ll> origin, polygon[22];
15
16 // Cross product
17 ll cross(pair<ll, ll> a, pair<ll, ll> b) {
18     return (a.y - origin.y) * (b.x - origin.x) -
19           (a.x - origin.x) * (b.y - origin.y);
20 }
21
22 // Which half of the plane some point lies in
23 int half(pair<ll, ll> p) {
24     if (p.x != origin.x) return (p.x < origin.x) - (p.x > origin.x);
25     return (p.y < origin.y) - (p.y > origin.y);
26 }
27
28 // Custom comparator to sort by bearing
29 bool operator<(Event a, Event b) {
30     int ah = half(a.loc), bh = half(b.loc);
31     if (ah == bh) {
32         ll c = cross(a.loc, b.loc);
33         if (c == 0) return a.type > b.type;
34         return c > 0;
35     }
36     return ah < bh;
37 }
38
39 // Generates the next fence post in clockwise order
40 Event get_next_post(Event curr, int n) {
41     if (curr.loc.x == n) {
42         if (curr.loc.y) return {0, 0, {n, curr.loc.y - 1}};
43         return {0, 0, {n - 1, 0}};
44     } else if (!curr.loc.x) {
45         if (curr.loc.y != n) return {0, 0, {0, curr.loc.y + 1}};
46         return {0, 0, {1, n}};
47     } else if (curr.loc.y == n) {
48         if (curr.loc.x != n) return {0, 0, {curr.loc.x + 1, n}};

```

```

49     return {0, 0, {n, n - 1}};
50 } else {
51     if (curr.loc.x) return {0, 0, {curr.loc.x - 1, 0}};
52     return {0, 0, {0, 1}};
53 }
54 }
55
56 vector<Event> events;
57 bool before[44444];
58
59 int main() {
60     cin.tie(0)->sync_with_stdio(0);
61     int n, r;
62     cin >> n >> r >> origin.x >> origin.y;
63
64     for (int i = 0; i < r; i++) {
65         int m;
66         cin >> m;
67         for (int j = 0; j < m; j++) cin >> polygon[j].x >> polygon[j].y;
68         // Sort the polygon's vertices to find the 2 "tangents" from the
        // origin
69         sort(polygon, polygon + m,
70             [](pair<ll, ll> a, pair<ll, ll> b) { return cross(a, b) > 0; });
71         events.push_back({1, i, polygon[0]});
72         events.push_back({-1, i, polygon[m - 1]});
73     }
74     sort(events.begin(), events.end());
75
76     int active = 0;
77     // Do an initial sweep to handle rocks containing the ray with bearing
78     // 0
79     // This way, 'active' won't be messed up
80     for (Event i : events) {
81         if (i.type == 1) before[i.id] = true;
82         if (i.type == -1 && !before[i.id]) active++;
83     }
84
85     int ans = 0, ptr = 0;
86     Event curr_post = {0, 0, {origin.x, n}};
87     for (Event i : events) {
88         while (ptr != 4 * n && curr_post < i) {
89             // If there are no rocks that our current ray intersects...
90             if (!active) ans++;
91             ptr++;
92             curr_post = get_next_post(curr_post, n);
93         }
94         if (i.type == 1) active++;
95         else active--;
96     }
97     if (!active) ans += 4 * n - ptr;
98
99     cout << ans;
100     return 0;
101 }

```

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. $O(n \log n)$. (Original Problem - Google Code Jam WF 2009 - B)

```

1 #include <algorithm>
2 #include <cassert>
3 #include <cmath>

```

```

4 #include <cstdio>
5 #include <cstdlib>
6 #include <vector>
7 using namespace std;
8 #define REP(i,n) for(int i=0;i<(n);++i)
9 template<class T> inline int size(const T&c) { return c.size();}
10
11 const int BILLION = 1000000000;
12 const double INF = 1e20;
13 typedef long long LL;
14
15 struct Point {
16     int x,y;
17     Point() {}
18     Point(int x,int y):x(x),y(y) {}
19 };
20
21 inline Point middle(const Point &a, const Point &b) {
22     return Point((a.x+b.x)/2, (a.y+b.y)/2);
23 }
24
25 struct CmpX {
26     inline bool operator()(const Point &a, const Point &b) {
27         if(a.x != b.x) return a.x < b.x;
28         return a.y < b.y;
29     }
30 } cmpx;
31
32 struct CmpY {
33     inline bool operator()(const Point &a, const Point &b) {
34         if(a.y != b.y) return a.y < b.y;
35         return a.x < b.x;
36     }
37 } cmpy;
38
39 inline LL sqr(int x) { return LL(x) * LL(x); }
40
41 inline double dist(const Point &a, const Point &b) {
42     return sqrt(double(sqr(a.x-b.x) + sqr(a.y-b.y)));
43 }
44
45 inline double perimeter(const Point &a,
46                         const Point &b,
47                         const Point &c) {
48     return dist(a,b) + dist(b,c) + dist(c,a);
49 }
50
51 double calc(int n, const Point points[],
52             const vector<Point> &pointsByY) {
53     if(n<3) return INF;
54     int left = n/2;
55     int right = n-left;
56     Point split = middle(points[left-1], points[left]);
57     vector<Point> pointsByYLeft, pointsByYRight;
58     pointsByYLeft.reserve(left);
59     pointsByYRight.reserve(right);
60     REP(i,n) {
61         if(cmpx(pointsByY[i], split))
62             pointsByYLeft.push_back(pointsByY[i]);
63         else
64             pointsByYRight.push_back(pointsByY[i]);
65     }
66     double res = INF;
67     res = min(res, calc(left, points, pointsByYLeft));
68     res = min(res, calc(right, points+left, pointsByYRight));
69     static vector<Point> closeToTheLine;

```

```

70  int margin = (res > INF/2) ? 2*BILLION : int(res/2);
71  closeToTheLine.clear();
72  closeToTheLine.reserve(n);
73  int start = 0;
74  for(int i=0;i<n;++i) {
75      Point p = pointsByY[i];
76      if(abs(p.x - split.x) > margin) continue;
77      while(start < size(closeToTheLine) &&
78             p.y - closeToTheLine[start].y > margin) ++start;
79      for(int i=start;i<size(closeToTheLine);++i) {
80          for(int j=i+1;j<size(closeToTheLine);++j) {
81              res = min(res, perimeter(p, closeToTheLine[i],
82                                     closeToTheLine[j]));
83          }
84      }
85      closeToTheLine.push_back(p);
86  }
87  return res;
88 }
89
90 double calc(vector<Point> &points) {
91     sort(points.begin(), points.end(), cmpx);
92     vector<Point> pointsByY = points;
93     sort(pointsByY.begin(), pointsByY.end(), cmpy);
94     return calc(size(points), &points[0], pointsByY);
95 }
96
97 int main() {
98     assert(0==system("cat > Input.java"));
99     fprintf(stderr, "Compiling generator\n");
100    assert(0==system("javac Input.java"));
101    fprintf(stderr, "Running generator\n");
102    assert(0==system("java -Xmx512M Input > input.tmp"));
103    fprintf(stderr, "Solving\n");
104    FILE *f = fopen("input.tmp", "r");
105    int ntc; fscanf(f, "%d", &ntc);
106    REP(tc, ntc) {
107        int n; fscanf(f, "%d", &n);
108        vector<Point> points;
109        points.reserve(n);
110        REP(i, n) {
111            int x, y; fscanf(f, "%d%d", &x, &y);
112            points.push_back(Point(2*x-BILLION, 2*y-BILLION));
113        }
114        double res = calc(points);
115        printf("Case #%d: %.15e\n", tc+1, res/2);
116    }
117    fclose(f);
118 }

```


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.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int N = 1000010;
5 const ll mod = 1000000007LL;
6 int dp[N];
7 int mex(set<int> s)
8 {
9     int x = 0;
10    while(s.find(x) != s.end())
11        x++;
12    return x;
13 }
14 int grundy(int n)
15 {
16     if(n <= 2)
17         return 0;
18     if(dp[n] != -1)
19         return dp[n];
20     set<int> s;
21     for(int i = 1; i + i < n; i++)
22     {
23         s.insert(grundy(i)^grundy(n-i));
24     }
25     return dp[n] = mex(s);
26 }
27 int main()
28 {
29     ios::sync_with_stdio(false);
30     cin.tie(NULL);
31     memset(dp,-1,sizeof(dp));
32     // for(int i = 1; i <= 50000; i++){
33     //     if(grundy(i) == 0 )
```

```
34 //          cout << i << endl;
35 // }
36 // cout << '\n';
37 int q;
38 cin >> q;
39 while(q-->0)
40 {
41     int n;
42     cin >> n;
43     if(n > 1222)
44         cout << "first\n";
45     else
46         cout << (grundy(n)?"first\n":"second\n");
47 }
48 return 0;
49 }
```

Example 2:

There is a staircase consisting of n stairs, numbered $1, 2, \dots, n$. 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.

```
1 int main()
2 {
3     int q;
4     cin >> q;
5     while(q-->0)
6     {
7         int n;
8         cin >> n;
9         int rs = 0;
10        for(int i = 0; i < n; i++)
11        {
12            int x;
13            cin >> x;
14            if(i&1)
15                rs ^= x;
16        }
17        cout << (rs?"first":"second") << '\n';
18    }
19    return 0;
20 }
```

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))

```
1 #include <iostream>
2 #include <vector>
3 #include <map>
4 #include <algorithm>
5 #include <cassert>
6
7 using namespace std;
8
9 #define forn(i,n) for(int i=0;i<int(n);i++)
10 #define all(c) begin(c), end(c)
11
12 #define SIZE(c) int((c).size())
13
14 typedef unsigned long long Nimber;
15
```

```

16 vector<int> exponents(Nimber value)
17 {
18     vector<int> ret;
19     unsigned long long x = value;
20     for (; x; x = (x-1)&x)
21         ret.push_back(__builtin_ctzll(x));
22     return ret;
23 }
24
25 map<pair<Nimber,Nimber>, Nimber> cache;
26
27 Nimber nimProduct(Nimber a, Nimber b)
28 {
29     auto it = cache.find(make_pair(a,b));
30     if (it != cache.end())
31         return it->second;
32     Nimber &ret = cache[make_pair(a,b)];
33     if (a == 0 || b == 0)
34         ret = 0;
35     else if (a == 1 || b == 1)
36         ret = a^b^1;
37     else
38     {
39         vector<int> aExponents = exponents(a);
40         vector<int> bExponents = exponents(b);
41         if (aExponents.size() == 1 && bExponents.size() == 1)
42         {
43             // Computes nim product of 2^a and 2^b
44             // Decompose exponents = write as product of fermats
45             vector<int> aExpBits = exponents(aExponents[0]);
46             vector<int> bExpBits = exponents(bExponents[0]);
47             #define FERMAT(index) Nimber(1ULL<<(1ULL<<(index)))
48             ret = Nimber(1);
49             int i = 0, j = 0;
50             while (i < SIZE(aExpBits) && j < SIZE(bExpBits))
51             {
52                 if (aExpBits[i] < bExpBits[j])
53                     ret *= FERMAT(aExpBits[i++]);
54                 else if (aExpBits[i] > bExpBits[j])
55                     ret *= FERMAT(bExpBits[j++]);
56                 else
57                 {
58                     ret = (ret * FERMAT(aExpBits[i])) ^ nimProduct(ret,
FERMAT(aExpBits[i])/2);
59                     i++;
60                     j++;
61                 }
62             }
63             for (; i < SIZE(aExpBits); i++) ret = ret * FERMAT(aExpBits[
i]);
64             for (; j < SIZE(bExpBits); j++) ret = ret * FERMAT(bExpBits[
j]);
65         }
66         else
67         {
68             ret = 0;
69             for (int aExp : aExponents)
70                 for (int bExp : bExponents)
71                     ret ^= nimProduct(1ULL<<aExp, 1ULL<<bExp);
72         }
73     }
74     return ret;
75 }
76
77
78 int main()

```

```
79 {
80     forn(i,16)
81     forn(j,16)
82         cout << i << " " << j << " " << nimProduct(i,j) << endl;
83     return 0;
84 }
```

10.2 Binary Search

10.2.1 Parallel Binary Search

(Original Problem: Atcoder Grand Contest 2 D)

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define pb push_back
5 #define sd second
6 #define ft first
7 const int N=100100;
8 int pai[N],rk[N],que[N],lo[N],hi[N];
9 vector<int> lista[N],mid[N];
10
11 vector<pair<pair<int,int>,int>> v;
12
13
14
15
16 void init(){
17     for(int i=0;i<N;i++){
18         pai[i]=i,rk[i]=1;
19     }
20 }
21
22 int find(int x){
23     if(x==pai[x])
24         return x;
25     return pai[x]=find(pai[x]);
26 }
27
28
29
30 void join(int a,int b){
31     a=find(a);
32     b=find(b);
33     if(a!=b){
34         if(rk[a]<rk[b])
35             swap(a,b);
36         pai[b]=a;
37         rk[a]+=rk[b];
38     }
39 }
40
41 vector<pair<int,int>> p;
42 int main(){
43     ios::sync_with_stdio(false);
44     cin.tie(NULL);
45     int n,m;
46     int q,a,b,z;
47     cin>>n>>m;
48     int t=m;
49     while(t--){
50         cin>>a>>b;
51         p.pb({a,b});
52     }
53 }
```

```

54  cin>>q;
55  for(int i=0;i<q;i++){
56      cin>>a>>b>>z;
57      lo[i]=0,hi[i]=m;
58      v.pb({{a,b},z});
59  }
60  bool need =true;
61  while(need){
62      for(int i=0;i<m;i++)
63          mid[i].clear();
64      for (int i = 0; i < q; ++i)
65      {
66          if(lo[i]<=hi[i])
67          {
68              mid[(lo[i]+hi[i])/2].pb(i);
69          }
70      }
71      init();
72      need=false;
73      for (int i = 0; i < m; ++i)
74      {
75          join(p[i].ft,p[i].sd);
76          for(auto at:mid[i])
77          {
78              need=true;
79              int val;
80              if(find(v[at].ft.ft)==find(v[at].ft.sd))
81              {
82                  val=rk[find(v[at].ft.ft)];
83              }else{
84                  val=rk[find(v[at].ft.ft)]+rk[find(v[at].ft.sd)];
85              }
86              if(val>=v[at].sd)
87              {
88                  hi[at]=i-1;
89                  que[at]=i;
90              }else{
91                  lo[at]=i+1;
92              }
93          }
94      }
95  }
96  }
97  for (int i = 0; i < q; ++i)
98  {
99      cout<<que[i]+1<<"\n";
100  }
101
102 }
```

10.3 Big Num

Implementação de Big Number em C++.

```

1  const int DIG = 4;
2  const int BASE = 10000; // BASE**3 < 2**51
3  const int TAM = 2048;
4
5  int cmp(int a, int b) {
6      if(a < b) return -1;
7      else if (a == b) return 0;
8      else return 1;
9  }
10
11 struct bigint {
```

```

12  int v[TAM], n;
13  bigint(int x = 0): n(1) {
14      memset(v, 0, sizeof(v));
15      v[n++] = x; fix();
16  }
17  bigint(char *s): n(1) {
18      memset(v, 0, sizeof(v));
19      int sign = 1;
20      while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
21      char *t = strdup(s), *p = t + strlen(t);
22      while (p > t) {
23          *p = 0; p = max(t, p - DIG);
24          sscanf(p, "%d", &v[n]);
25          v[n++] *= sign;
26      }
27      free(t); fix();
28  }
29  bigint& fix(int m = 0) {
30      n = max(m, n);
31      int sign = 0;
32      for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
33          v[i] += e; e = v[i] / BASE; v[i] %= BASE;
34          if (v[i]) sign = (v[i] > 0) ? 1 : -1;
35      }
36
37      for (int i = n - 1; i > 0; i--)
38          if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
39
40      while (n && !v[n]) n--;
41      return *this;
42  }
43  int cmp(const bigint& x = 0) const {
44      int i = max(n, x.n), t = 0;
45      while(1){
46          cout<<mp(v[i], x.v[i])<<"\n";
47          if((t = ::cmp(v[i], x.v[i])) || i-- == 0){
48              cout<<t<<"\n";
49              return t;
50          }
51      }
52  }
53  bool operator <(const bigint& x) const { return cmp(x) < 0; }
54  bool operator ==(const bigint& x) const { return cmp(x) == 0; }
55  bool operator !=(const bigint& x) const { return cmp(x) != 0; }
56  bool operator <=(const bigint& x) const { return cmp(x) <= 0; }
57  bool operator >(const bigint& x) const { return cmp(x) > 0; }
58  bool operator >=(const bigint& x) const { return cmp(x) >= 0; }
59
60  operator string() const {
61      ostringstream s; s << v[n];
62      for (int i = n - 1; i > 0; i--) {
63          s.width(DIG); s.fill('0'); s << abs(v[i]);
64      }
65      return s.str();
66  }
67  friend ostream& operator <<(ostream& o, const bigint& x) {
68      return o << (string) x;
69  }
70
71  bigint& operator +=(const bigint& x) {
72      for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
73      return fix(x.n);
74  }
75  bigint operator +(const bigint& x) { return bigint(*this) += x; }
76  bigint& operator -=(const bigint& x) {

```

```

77     for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
78     return fix(x.n);
79 }
80 bigint operator -(const bigint& x) { return bigint(*this) -= x; }
81 bigint operator -() { bigint r = 0; return r -= *this; }
82 void ams(const bigint& x, int m, int b) { // *this += (x * m) << b;
83     for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
84         v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
85     }
86 }
87 bigint operator *(const bigint& x) const {
88     bigint r;
89     for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
90     return r;
91 }
92 bigint& operator *=(const bigint& x) { return *this = *this * x; }
93 // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
94 bigint div(const bigint& x) {
95     if (x == 0) return 0;
96     bigint q; q.n = max(n - x.n + 1, 0);
97     int d = x.v[x.n] * BASE + x.v[x.n-1];
98     for (int i = q.n; i > 0; i--) {
99         int j = x.n + i - 1;
100         q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
101         ams(x, -q.v[i], i-1);
102         if (i == 1 || j == 1) break;
103         v[j-1] += BASE * v[j]; v[j] = 0;
104     }
105     fix(x.n); return q.fix();
106 }
107 bigint& operator /=(const bigint& x) { return *this = div(x); }
108 bigint& operator %=(const bigint& x) { div(x); return *this; }
109 bigint operator /(const bigint& x) { return bigint(*this).div(x); }
110 bigint operator %(const bigint& x) { return bigint(*this) %= x; }
111 bigint pow(int x) {
112     if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
113     bigint r = 1;
114     for (int i = 0; i < x; i++) r *= *this;
115     return r;
116 }
117 bigint root(int x) {
118     if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
119     if (*this == 1 || x == 1) return *this;
120     if (cmp() < 0) return -(*this).root(x);
121     bigint a = 1, d = *this;
122
123     while (d != 1) {
124         bigint b = a + (d /= 2);
125         if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
126     }
127     return a;
128 }
129 };

```

10.3.1 D&C for Multiplyng two big numbers

```

1 typedef vector<int> poly;
2
3 poly mult(const poly& p, const poly& q) {
4     int sz = p.size(), half = sz/2;
5     assert(sz == q.size() && !(sz&(sz-1)));
6
7     if (sz <= 64) {
8         poly ret(2*sz);
9         for (int i = 0; i < sz; i++)

```

```
10         for(int j = 0; j < sz; j++)
11             ret[i+j] += p[i] * q[j];
12     return ret;
13 }
14
15 poly p1(p.begin(), p.begin() + half), p2(p.begin() + half, p.end());
16 poly q1(q.begin(), q.begin() + half), q2(q.begin() + half, q.end());
17 poly p1p2(half), q1q2(half);
18 for(int i = 0; i < half; i++)
19     p1p2[i] = p1[i] + p2[i], q1q2[i] = q1[i] + q2[i];
20
21 poly low = mult(p1, q1), high = mult(p2, q2), mid = mult(p1p2, q1q2)
22 ;
23 for(int i = 0; i < sz; i++)
24     mid[i] -= high[i] + low[i];
25
26 low.resize(2*sz);
27 for(int i = 0; i < sz; i++)
28     low[i+half] += mid[i], low[i+sz] += high[i];
29
30 return low;
31 }
```

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:

```
1 bitset<size> variable_name;
2 bitset<size> variable_name(DECIMAL_NUMBER);
3 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):

<code>__builtin_popcount(x)</code>	Counts the quantity of one's(set bits) in an integer.
<code>__builtin_parity(x)</code>	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).
<code>__builtin_clz(x)</code>	Counts the leading quantity of zeros of the integer.
<code>__builtin_ctz(x)</code>	Counts the trailing quantity of zeros of the integer.
<code>__builtin_popcountll(x)</code>	Counts the quantity of one's(set bits) in an long long.
<code>__builtin_parityll(x)</code>	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).
<code>__builtin_clzll(x)</code>	Counts the leading quantity of zeros of the long long.
<code>__builtin_ctzll(x)</code>	Counts the trailing quantity of zeros of the long long.

clz can be used to find the first set bit, since u can use:

```
1 int x;  
2 ll x1;  
3 int fsi = 31 - __builtin_clz(x);  
4 int fsll = 63 - __builtin_clzll(x1);
```

10.6 Priority Queue and Set Comparators

Set Comparators Operator Overloading:

```
1 #include <bits/stdc++.h>  
2 using namespace std;  
3  
4 struct Edge {  
5     int a, b, w;  
6     bool operator< (const Edge &y) const { return w < y.w; }  
7 };  
8  
9 int main() {  
10     int M = 4;  
11     set<Edge> v;  
12     for (int i = 0; i < M; ++i) {  
13         int a, b, w;  
14         cin >> a >> b >> w;  
15         v.insert({a, b, w});  
16     }  
17     for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";  
18 }
```

Functors:

```
1 #include <bits/stdc++.h>  
2 using namespace std;  
3  
4 struct Edge {  
5     int a, b, w;  
6 };  
7  
8 struct cmp {  
9     bool operator() (const Edge &x, const Edge &y) const { return x.w < y.w  
10         ; }  
11 };  
12  
13 int main() {  
14     int M = 4;  
15     set<Edge, cmp> v;  
16     for (int i = 0; i < M; ++i) {  
17         int a, b, w;  
18         cin >> a >> b >> w;  
19         v.insert({a, b, w});  
20     }  
21     for (Edge e : v) cout << e.a << " " << e.b << " " << e.w << "\n";  
22 }
```

Functors can also be used in priority queues, but needs vector as a container:

```
1 priority_queue<int, vector<int>, cmp> c;
```

Built in Functors:

```
1 set<int, greater<int>> a; //max set
2 map<int, string, greater<int>> b; // max map
3 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:

```
1 [](){cout << "Hello, world!" << endl;}
2 auto func1 = [](int i) {cout << ":" << i << ":";};
3 func1(x);
```

Using & in capture list it will have access to the scope.

Recursive lambda using function:

```
1 int main() {
2     function<int(int, int)> gcd = [&](int a, int b) {
3         return b == 0 ? a : gcd(b, a % b);
4     };
5     cout << gcd(20, 30) << '\n'; // outputs 10
6 }
```

10.8 Fast input output

```
1 inline int next_int() {
2     int n = 0;
3     char c = getchar_unlocked();
4     bool neg = false;
5     while (!(c >= '0' && c <= '9')){
6         if(c == '-'){
7             c = getchar_unlocked();
8             if(c >= '0' && c <= '9'){
9                 neg = true;
10            }
11        }else c = getchar_unlocked();
12    }
13
14    while ('0' <= c && c <= '9') {
15        n = n * 10 + c - '0';
16        c = getchar_unlocked();
17    }
18
19    if(neg) return -n;
20    else return n;
21 }
22
23 inline char next_char(){
24     char c = getchar_unlocked();
25     while(c == ' ' || c == '\n') c = getchar_unlocked();
26     return c;
27 }
28
29 inline string next_string(){
30     string out;
31     char c = getchar_unlocked();
32     while(c == ' ' || c == '\n') c = getchar_unlocked();
33     while(!(c == ' ' || c == '\n')){
```

```

34     out.pb(c);
35     c = getchar_unlocked();
36 }
37
38     return out;
39 }
40
41
42 bool read( int &n ) {
43     n = 0;
44     register bool neg = false;
45     register char c = getchar_unlocked();
46     if( c == EOF ) { n = -1; return false; }
47     while ( !('0' <= c && c <= '9') ) {
48         if( c == '-' ) neg = true;
49         c = getchar_unlocked();
50     }
51     while ( '0' <= c && c <= '9' ) {
52         n = n * 10 + c - '0';
53         c = getchar_unlocked();
54     }
55     n = (neg ? (-n) : (n));
56     return true;
57 }
58
59 inline void writeInt(int n){
60     register int idx = 20;
61     if( n < 0 ) putchar_unlocked('-');
62     n = abs(n);
63     char out[21];
64     out[20] = ' ';
65     do{
66         idx--;
67         out[idx] = n % 10 + '0';
68         n/= 10;
69     }while(n);
70     do{ putchar_unlocked(out[idx++]); } while (out[idx] != ' ');
71 }

```

10.9 Trick for faster Unordered Map

```

1 struct pair_hash {
2     size_t operator()(const pair<int,int>&x) const{
3         return hash<ll>()(((ll)x.first)^(((ll)x.second)<<32));
4     }
5 };
6 unordered_map<pair<int,int>,int,pair_hash> mp;
7
8 int main()
9 {
10     //usar o tamanho esperado do umap
11     mp.reserve(N*4);
12     mp.max_load_factor(0.25);
13 }

```

10.10 Faster Hash Table

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 gp_hash_table<int, int> table;
4
5 //hash definida

```

```

6  const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch
    ().count();
7  struct chash {
8      int operator()(int x) const { return x ^ RANDOM; }
9  };
10 gp_hash_table<key, int, chash> table;
11
12 //hash de pair
13 struct chash {
14     int operator()(pii x) const { return x.first* 31 + x.second; }
15 };
16
17 gp_hash_table<pii, int, chash> table;
18 // hash de vector
19 struct VectorHasher {
20     int operator()(const vector<int> &V) const {
21         int hash = V.size();
22         for(auto &i : V) {
23             hash ^= i + 0x9e3779b9 + (hash << 6) + (hash >> 2);
24         }
25         return hash;
26     }
27 };

```

10.11 StringStream

A way to parse strings in c++.

```

1  //istringstream
2  //ler linhas do tipo: +0 +2 -0 +1 -1 -2
3  string s;
4  istringstream iss;
5  rep(i, 0, n){
6      getline(cin, s);
7      iss.clear();
8      iss.str(s);
9
10     while(iss >> ch){
11         iss >> in;
12     }
13 }
14
15 //ostringstream
16 ostringstream oss;
17
18 int idade1 = 30;
19 string nome1 = "Alice";
20 double salario1 = 2500.75;
21 oss << "óRelatrio 1\n";
22 oss << "Nome: " << nome1 << ", Idade: " << idade1 << ", áSalrio: R$ " <<
    salario1 << "\n";
23 cout << oss.str() << endl;
24
25 oss.str("");
26 oss.clear();
27
28 int idade2 = 45;
29 string nome2 = "Bob";
30 double salario2 = 3200.50;
31 oss << "óRelatrio 2\n";
32 oss << "Nome: " << nome2 << ", Idade: " << idade2 << ", áSalrio: R$ " <<
    salario2 << "\n";
33 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 heuristic algorithm is $\mathcal{O}(n^{-lg(n)}) = \mathcal{O}(\frac{1}{n^{lg(n)}})$

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int karmakarp(multiset<int>& items, multiset<int>& P1, multiset<int>& P2
5 ) {
6     if (items.size() == 1) {
7         P1.clear();
8         P1.insert(*items.begin());
9         P2.clear();
10        return *items.begin();
11    }
12
13    int bigger1 = *items.rbegin();
14    items.erase(items.find(*items.rbegin()));
15    int bigger2 = *items.rbegin();
16    items.erase(items.find(*items.rbegin()));
17
18    int diff = bigger1 - bigger2;
19    items.insert(diff);
20
21    int bestDiff = karmakarp(items, P1, P2);
22
23    if (P1.find(diff) == P1.end()) {
24        swap(P1, P2);
25    }
26
27    P1.erase(P1.find(diff));
28    P1.insert(bigger1);
29
30    P2.insert(bigger2);
31    return bestDiff;
32 }
33
34 int main() {
35
36     vector<int> vet = {3, 8, 12, 16, 38, 41, 73};
37     multiset<int> P1, P2;
38
39     multiset<int> items;
40     for (int v : vet)
41         items.insert(v);
42
43     cout << karmakarp(items, P1, P2) << "\n";
44
45     for (int v : P1)
46         cout << v << " ";
47     cout << "\n";
48
49     for (int v : P2)
50         cout << v << " ";
51     cout << "\n";
52
53     return 0;
54 }

```