

1 string

1.1	kmp
1.2	aho
1.3	trie
1.4	zfunc

2.1	cht2
2.2	cht
2.3	lis

3.1	mint
3.2	extendedEuclidean
3.3	crt
3.4	pollardrho
3.5	frac
3.6	fastexp
3.7	divtrick
3.8	phi
3.9	gaussianElim

8.1	minkowski
4.2	mincircle
4.3	closestpair
4.4	point
4.5	convexhull

5.1	maxqueue
5.2	segtree-lazy
5.3	kd-tree
5.4	colorupdate
5.5	sparsetable
5.6	fenwick-tree
5.7	dynamic-median
5.8	segtree-topdown

6.1	mo
6.2	bitset
6.3	ternary-search
6.4	formulas
6.5	composite

7.1	kosaraju
7.2	dinic
7.3	kruskal
7.4	smallToLarge
7.5	policyBased
7.6	centroid
7.7	articulation
7.8	mcmf
7.9	dsu
7.10	dijkstra
7.11	twosat
7.12	lca
7.13	floydWarshall
7.14	bridges

8.1 Hash Function

1.1 kmp

```

1 642 vector<int> getBorder(string str) {
1 555     int n = str.size();
1 7AD vector<int> border(n, -1);
1 199     for(int i = 1, j = -1; i < n; i++) {
2 904         while(j >= 0 && str[i] != str[j + 1]) {
2 330             j = border[j];
2 A09         }
2 AE4         if(str[i] == str[j + 1]) {
2 04B             j++;
2 B31         }
2 805         border[i] = j;
2 4D9     }
2 887     return border;
3 EA7 }
3 971 int matchPattern(const string &txt, const string &pat,
3         const vector<int> &border) {
3 13D     int freq = 0;
3 035     for(int i = 0, j = -1; i < txt.size(); i++) {
4 8DA         while(j >= 0 && txt[i] != pat[j + 1]) {
4 330             j = border[j];
4 00A         }
4 53D         if(pat[j + 1] == txt[i]) {
4 04B             j++;
4 8CC         }
4 99B         if(j + 1 == (int) pat.size()) {
4 D41             //found occurence
5 CB5             freq++;
5 330             j = border[j];
5 F5F         }
5 D55     }
5 E33     return freq;
5 065 }

```

1.2 aho

```

7 123 struct AhoType {
8 847     static const int ALPHA = 26;
7 1E3     static int f(char c) { return c - 'A'; } // ver se ta
7     maiusculo ou minusculo aqui
8 D13 };
8 29B template<typename AhoType>
8 51F struct AhoCorasick {
8 BF2     struct Node {
64C         int nxt[AhoType::ALPHA] {};;
8 OD9         int p = 0, ch = 0, len = 0;
8 E7A         int link = 0;
8 79F         int occ_link = 0;
9 F4F         Node(int p = 0, int ch = 0, int len = 0) : p(p), ch(ch
9 ), len(len) {}
9     };
9 8ED     vector<Node> tr;
9 69B     AhoCorasick() : tr(1) {}
10 1F7     template<typename Iterator>
10 2CA     void add_word(Iterator first, Iterator last) {
10 AC3         int cur = 0, len = 1;
10 68C         for(; first != last; ++first) {
11 ED9             auto ch = AhoType::f(*first);
11 4F3             if(tr[cur].nxt[ch] == 0) {
11 9BF                 tr[cur].nxt[ch] = int(tr.size());
10 6CC                 tr.emplace_back(cur, ch, len);
11 B7C             }
11 BEE             cur = tr[cur].nxt[ch];

```

```

250         ++len;
159     }
D91     tr[cur].occ_link = cur;
FA3 }
0A8 void build() {
A36     vector<int> bfs(int(tr.size()));
2AA     int s = 0, t = 1;
D33     while(s < t) {
B21         int v = bfs[s++], u = tr[v].link;
F9E         if(tr[v].occ_link == 0) {
99B             tr[v].occ_link = tr[u].occ_link;
E75         }
609         for(int ch = 0; ch < AhoType::ALPHA; ++ch) {
31D             auto& nxt = tr[v].nxt[ch];
9FA             if(nxt == 0) {
2CA                 nxt = tr[u].nxt[ch];
95C             } else {
FE1                 tr[nxt].link = v > 0 ? tr[u].nxt[ch] : 0;
47D                 bfs[t++] = nxt;
353             }
D85         }
FBE     }
7FF }
1F7 template<typename Iterator>

7F0     vector<pair<int,int>> get_all_matches(Iterator first,
Iterator last) const {
FD9     vector<pair<int,int>> occs;
EO9     for(int cur = 0, i = 0; first != last; ++i, ++first) {
ED9         auto ch = AhoType::f(*first);
BEE         cur = tr[cur].nxt[ch];
F2C         for(int v = tr[cur].occ_link; v > 0; v = tr[tr[v].
link].occ_link) {
D41             // i = pos text, v = state
D1A             occs.push_back({i+tr[v].len, i});
34C         }
08F     }
23F     return occs;
DD5 }
67A template<typename T>
578     int get_next(int cur, T ch) const { return tr[cur].nxt[
AhoType::f(ch)]; }
3F9 };

```

1.3 trie

```
CFC  int trie[ms][sigma], terminal[ms], z = 1;

33B  void insert(string &p) {
B3D    int cur = 0;
E2E    for(int i = 0; i < p.size(); i++) {
1BF      int id = p[i] - 'a';
919      if(!trie[cur][id]) {
869        trie[cur][id] = z++;
45C      }
3AD      cur = trie[cur][id];
D9E    }
B07    terminal[cur]++;
5EC  }

684  int count(string &p) {
B3D    int cur = 0;
E2E    for(int i = 0; i < p.size(); i++) {
1BF      int id = p[i] - 'a';
919      if(!trie[cur][id]) {
D1F        return false;
F06      }
3AD      cur = trie[cur][id];
```

```

532     }
89E     return terminal[cur];
B27 }

```

1.4 zfunc

```

403 vector<int> Zfunction(string &s){
163     int n = s.size();
2B1     vector<int> z (n, 0);
A5C     for(int i=1, l=0, r=0; i<n; i++) {
76D         if(i <= r) z[i] = min(z[i-1], r-i+1);
F61         while(z[i] + i < n && s[z[i]] == s[i+z[i]]) z[i]++;
EAF         if(r < i+z[i]-1) l = i, r = i+z[i]-1;
0CD     }
070     return z;
D58 }

```

2 dp

2.1 cht2

```

72C struct Line{
12D     ll a, b;
028     double x_inter;
01D     Line(ll a, ll b, double x_inter = inf) : a(a), b(b),
        x_inter(x_inter){}

D82     bool operator < (double x){
80C         return x_inter < x;
1CB     }
30B     ll eval(ll x){
F27         return a*x + b;
480     }
7DA };

88F double intersect(Line x, Line y){
BBA     assert(x.a != y.a);
CBE     return (1.0d * x.b - y.b) / (1.0d * y.a - x.a);
65D }

4B5 struct CHT{
398     deque<Line> lines;
DC7     void insert_right(ll a, ll b){
D85         while(lines.size() >= 2){
595             Line x = lines[lines.size() - 2], y = lines[
lines.size() - 1];
CBB             if(intersect(y, {a,b,0}) > intersect(x,y))
break;
501             lines.pop_back();
04A         }
4CB         if(!lines.empty()) lines[lines.size() - 1].x_inter
= intersect(lines.back(), {a,b,0});
748         lines.push_back(Line(a, b));
E42     }
7F4     void insert_left(ll a, ll b){
D85         while(lines.size() >= 2){
24D             Line x = lines[1], y = lines[0];
461             if(intersect(y, {a,b,0}) < intersect(x,y))
break;
688             lines.pop_front();
07A         }
29A         lines.push_front(Line(a, b));
359         if(!lines.empty()) lines.back().x_inter = inf;

```

```

BBF         if(lines.size() > 1) lines[0].x_inter = intersect(
lines[0], lines[1]);
1E9     }
BF9     ll qry(ll x){ // todo: fazer two pointers pra ficar
        linear
349         auto lb = lower_bound(begin(lines), end(lines), x)
;
7F9         return (*lb).eval(x);
327     }
AAD     void dbg(){
DDE         for(Line l : lines)
F09             cout << l.a << "x + " << l.b << " " << l.
x_inter << br;
C97     }
F51 };

```

2.2 cht

```

72C struct Line {
A3B     ll m, c;
D2C     Line(ll m, ll c) : m(m), c(c) {}
30B     ll eval(ll x) {
255         return m * x + c;
3CD     }
E9C };
4B5 struct CHT {
B57     vector<Line> lines;
1D1     bool bad(Line a, Line b, Line c) {
D41         // trocar pra < se for max
62B         return 1.d * (c.c - a.c)*(a.m - b.m) > 1.d * (b.c - a.
c)*(a.m - c.m);
0B3     }
7CE     void insert(Line line) { // sortar antes de inserir
544         int sz = (int)lines.size();
7D8         for(; sz > 1; --sz) {
DBD             if(bad(lines[sz - 2], lines[sz - 1], line)) {
501                 lines.pop_back();
5E2                 continue;
578             }
C2B             break;
FF1         }
770         lines.emplace_back(line);
4FD     }
4AD     ll query(ll x) {
82D         int l = 0, r = (int)lines.size() - 1;
40C         while(l < r) {
EE4             int m = (l+r)/2;
D41             // trocar pra < se for max
A32             if(lines[m].eval(x) > lines[m+1].eval(x)) {
16D                 l = m + 1;
568             } else {
3E2                 r = m;
476             }
E56         }
348         return lines[l].eval(x);
571     }
7AC };

```

2.3 lis

```

D41 // Longest Increasing Sequence
514 int lis(vector<ll>& nums){
F64     int n = nums.size();
CF7     vector<ll> s;
603     for(int i = 0; i < n; i++){
EEB         auto it = lower_bound(s.begin(), s.end(), nums[i]);

```

```

BA0         if(it == s.end()){
719             s.pb(nums[i]);
C60         }
4E6         else{
570             *it = nums[i];
AD4         }
358     }
8B9     return (int)s.size();
0B2 }

```

3 math

3.1 mint

```

67A template<typename T>
56C T bin_exp(T a, long long e) {
DAC     T r(1);
D0E     for(; e > 0; e >= 1) {
EEE         if(e & 1) {
1C8             r *= a;
D4B         }
70C         a *= a;
EF5     }
4C1     return r;
D51 }
016 template<const uint32_t MOD>
BB6 struct Mod {
622     uint32_t x;
77D     Mod() : x(0) {}
67A     template<typename T>
EA0     Mod(T x) : x(uint32_t(((int64_t(x) % MOD) + MOD) % MOD))
{}
ECC     Mod& operator+=(Mod rhs) {
393         x += rhs.x;
290         if(x >= MOD) x -= MOD;
357         return *this;
7F3     }
1BD     Mod& operator-=(Mod rhs) {
C2B         x += MOD - rhs.x;
290         if(x >= MOD) x -= MOD;
357         return *this;
51D     }
EAD     Mod& operator*=(Mod rhs) {
4E6         auto y = 1ull * x * rhs.x;
2AA         if(y >= MOD) y %= MOD;
A6E         x = uint32_t(y);
357         return *this;
89A     }
4B8     Mod& operator/=(Mod rhs) { return *this *= bin_exp(rhs,
MOD - 2); }
CE9     friend Mod operator+(Mod lhs, Mod rhs) { return lhs +=
rhs; }
16B     friend Mod operator-(Mod lhs, Mod rhs) { return lhs -=
rhs; }
D5C     friend Mod operator*(Mod lhs, Mod rhs) { return lhs *=
rhs; }
5B7     friend Mod operator/(Mod lhs, Mod rhs) { return lhs /=
rhs; }
2B2     bool operator==(Mod rhs) const { return x == rhs.x; }
D50     bool operator!=(Mod rhs) const { return x != rhs.x; }
17E     friend ostream& operator<<(ostream& os, const Mod& o) {
return os << o.x; }
52F     friend istream& operator>>(istream& is, Mod& o) {
C23         int64_t x;
AF7         is >> x;
84C         o = Mod(x);

```

```
FED     return is;
F1B    }
A9E    };
```

3.2 extendedEuclidean

```
89C int gcd(int a, int b, int& x, int& y) {
A30     if (b == 0) {
483         x = 1;
01D         y = 0;
3F5         return a;
433     }
608     int x1, y1;
E8B     int d = gcd(b, a % b, x1, y1);
711     x = y1;
A2A     y = x1 - y1 * (a / b);
BE2     return d;
AF0 }
```

```
D41 // inverso modular de a
43C int inv, y;
7A8 int g = gcd(a,mod,inv,y);
37A inv = (inv % m + m) % m;
```

3.3 crt

```
C22 ll euclid(ll a, ll b, ll&x, ll&y){
1EE     if(!b) return x = 1, y = 0, a;
E3D     ll d = euclid(b, a % b, y, x);
0A4     return y -= a/b * x, d;
33B }
A4B ll crt(vector<ll>& rem, vector<ll>& mod){
1BB     int n = rem.size();
233     if(n == 0) return 0;
2F3     ll ans = rem[0], m = mod[0];
6F5     for(int i = 1; i < n; i++){
0BE         ll x,y;
168         ll g = euclid(mod[i],m,x,y);
D41         // if((ans - rem[i]) % g != 0) return -5;
865         assert((ans - rem[i]) % g == 0);
263         ans = ans + 1LL*(rem[i]-ans)*(m/g)*y;
B68         m = (mod[i]/g)*(m/g)*g;
6AD     }
BA7     return ans;
A58 }
```

3.4 pollardrho

```
D41 // from: https://github.com/kth-competitive-programming/
      kactl
F4C typedef unsigned long long ull;
F85 ull modmul(ull a, ull b, ull M) {
2DD     ll ret = a * b - M * ull(1.L / M * a * b);
964     return ret + M * (ret < 0) - M * (ret >= (1l)M);
E93 }
4F6 ull modpow(ull b, ull e, ull mod) {
C1A     ull ans = 1;
A18     for (; e; b = modmul(b, b, mod), e /= 2)
9E8         if (e & 1) ans = modmul(ans, b, mod);
BA7     return ans;
100 }
DA4 bool isPrime(ull n) {
C16     if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
```

```
43A     ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
1795265022},
C17     s = __builtin_ctzll(n-1), d = n >> s;
E80     for (ull a : A) {
6B4         ull p = modpow(a%n, d, n), i = s;
274         while (p != 1 && p != n - 1 && a % n && i--)
C77             p = modmul(p, p, n);
E28         if (p != n-1 && i != s) return 0;
EDF     }
6A5     return 1;
60D }
7EB ull pollard(ull n) {
222     ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
5F5     auto f = [&](ull x) { return modmul(x, x, n) + i; };
2AC     while (t++ % 40 || __gcd(prd, n) == 1) {
BE9         if (x == y) x = ++i, y = f(x);
70F         if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q
;
B78         x = f(x), y = f(f(y));
989     }
756     return __gcd(prd, n);
CD2 }
591 vector<ull> factor(ull n) {
1B9     if (n == 1) return {};
6B5     if (isPrime(n)) return {n};
BC6     ull x = pollard(n);
52A     auto l = factor(x), r = factor(n / x);
E86     l.insert(l.end(), begin(r),end(r));
792     return l;
C99 }
D41 // hash: 3782
      d14edf6f6c81aa19f1dbbf0b31d3fa0b82704f5aeb2f2554fd3bc8404702
```

3.5 frac

```
D41 // de tfg
4FC template<class T>
28A T gcd(T a, T b) { return b == 0 ? a : gcd(b, a % b); }
4FC template<class T>
67A struct Frac {
4A6     T p, q;
124     Frac() {
FF3         p = 0, q = 1;
DDF     }
735     Frac(T x) {
E94         p = x;
B4A         q = 1;
11B     }
2B4     Frac(T a, T b) {
A30         if(b == 0) {
954             a = 0;
102             b = 1;
597         }
C53         p = a;
008         q = b;
39B         fix();
E90     }
50D     Frac<T> operator + (Frac<T> o) const { return Frac(p * o
.q + o.p * q, q * o.q); }
DF1     Frac<T> operator - (Frac<T> o) const {return Frac(p * o.
q - o.p * q, q * o.q); }
DE5     Frac<T> operator * (Frac<T> o) const { return Frac(p * o
.p, q * o.q); }
1EC     Frac<T> operator / (Frac<T> o) const { return Frac(p * o
.q, q * o.p); }
```

```
01A     void fix() {
4C3         if(q < 0) {
A71             q = -q;
1A2             p = -p;
729         }
BF8         auto g = gcd(max(p, -p), q);
5C4         p /= g;
EC3         q /= g;
698     }
```

```
6EA     bool operator < (Frac<T> o) const { return ((*this) - o)
.p < 0; }
CB2     bool operator > (Frac<T> o) const { return ((*this) - o)
.p > 0; }
EA8     friend ostream& operator << (ostream &os, const Frac<T>
&f) {
603         return os << f.p << '/' << f.q;
E4F     }
5A1     friend istream& operator >> (istream &is, Frac<T> &f) {
B5D         char trash;
F9E         return is >> f.p >> trash >> f.q;
654     }
3F1 };
```

3.6 fastexp

```
D41 // Fast Exp
031 const ll mod = 1e9+7;
8D8 ll fexpll(ll a, ll n){
D54     ll ans = 1;
02A     while(n){
A19         if(n & 1) ans = (ans * a) % mod;
4E2         a = (a * a) % mod;
9D3         n >>= 1;
CAB     }
BA7     return ans;
D19 }
D41 // matriz quadrada
BE9 class Matrix{
673     public:
21E     vector<vector<ll>> mat;
2E6     int m;
1D7     Matrix(int m): m(m){
593         mat.resize(m);
2BC         for(int i = 0; i < m; i++) mat[i].resize(m,0);
809     }
215     Matrix operator * (const Matrix& rhs){
8EB         Matrix ans = Matrix(m);
94F         for(int i = 0; i < m; i++)
A75             for(int j = 0; j < m; j++)
800                 for(int k = 0; k < m; k++)
1F7                     ans.mat[i][j] = (ans.mat[i][j] + (mat[i][k] *
rhs.mat[k][j]) % mod) % mod;
BA7         return ans;
2E6     }
A70 };
```

```
E2E Matrix fexp(Matrix a, ll n){
71E     int m = a.m;
8EB     Matrix ans = Matrix(m);
642     for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
02A     while(n){
A50         if(n & 1) ans = ans * a;
476         a = a * a;
```

```
9D3     n >>= 1;
CDF     }
BA7     return ans;
966 }
```

3.7 divtrick

```
79C for(int l = 1, r; l <= n; l = r + 1) {
746     r = n / (n / l);
D41     // n / i has the same value for l <= i <= r
D41     // O(sqrt(n)) different floor(n/i) values
5BF }
```

3.8 phi

```
A8C const int LIM = 1e6+5;
C75 int phi[LIM];
8E0 void sieve() {
9A6     iota(phi, phi + LIM, 0);
C35     for(int i = 2; i < LIM; i++) {
729         if(phi == i) {
EBC             for(int j = i; j < LIM; j += i) {
A9B                 phi[j] -= phi[j] / i;
4BC             }
37B         }
02A     }
953 }
```

```
67A template<typename T>
E6F T phi(T n) {
FC4     T ans = n;
D24     for(T p = 2; p * p <= n; p++) {
80A         if(n % p == 0) {
B7F             ans -= ans / p;
03E             while(n % p == 0) {
F4A                 n /= p;
91F             }
D76         }
4BB     }
B26     if(n > 1) {
675         ans -= ans / n;
C1B     }
BA7     return ans;
427 }
```

3.9 gaussianElim

```
67A template<typename T>
029 struct GaussElim {
757     vector<vector<T>> rows;
AF2     vector<int> where;
BDF     vector<bool> hasRow;
B5C     int m, n;
75D     GaussElim(int vars) : m(vars) {
E53         where.assign(m, -1);
84E         n = 0;
9CB     }
FC0     void add_eq(vector<T> row) { // colocar o b aqui tb
6D8         rows.push_back(row);
15A         hasRow.push_back(false);
015         n++;
CA0     }
71A     int status() {
D41         // 0: no solution, 1: unique, 2: infinite
```

```
830     for(int i = 0; i < n; i++)
673         if(!hasRow[i] && rows[i].back() != T(0))
BB3             return 0;
A75     for(int j = 0; j < m; j++)
EA5         if(where[j] == -1)
18B             return 2;
```

```
6A5     return 1;
2A2 }
C44 int go() {
CA3     int n = rows.size();
891     for(int j = 0; j < m; j++) {
603         for(int i = 0; i < n; i++) {
A03             if(rows[i][j] != T(0) && !hasRow[i]) {
33C                 where[j] = i;
258                 hasRow[i] = 1;
C2B                 break;
4D1             }
E80         }
DC1         if(where[j] == -1) continue;
D41         // fix linha where[j]
F71         T div = rows[where[j]][j];
38C         for(int k = 0; k <= m; k++) {
24C             rows[where[j]][k] /= div;
603         }
853         for(int i = 0; i < n; i++) {
F58             if(i == where[j]) continue;
38C             T mul = -1*rows[i][j];
B6E             for(int k = 0; k <= m; k++)
D94                 rows[i][k] += mul*rows[where[j]][k];
8AD         }
D22         return status();
12F     }
05B     vector<T> get() {
D41         // assert status = 1
DF8         vector<T> ret(m);
94F         for(int i = 0; i < m; i++)
F59             ret[i] = rows[where[i]].back();
EDF         return ret;
5BF     }
D7C };
```

4 geometry

4.1 minkowski

```
F05 using P = PT<double>;
B3C vector<P> minkowskiSum(vector<P> p, vector<P> q) {
D27     if(p.empty() || q.empty()) return {};
B23     auto fix = [](vector<P>& x) {
48D         rotate(x.begin(), min_element(x.begin(), x.end()), x.
end());
72A         x.push_back(x[0]), x.push_back(x[1]);
D06     };
00A     fix(p); fix(q);
2E3     vector<P> ret;
692     int i = 0, j = 0;
2EE     while (i < p.size()-2 or j < q.size()-2) {
898         ret.push_back(p[i] + q[j]);
132         auto c = ((p[i+1] - p[i]).cross(q[j+1] - q[j]));
EBC         if (c >= 0) i = min<int>(i+1, p.size()-2);
81E         if (c <= 0) j = min<int>(j+1, q.size()-2);
40F     }
EDF     return ret;
D08 }
```

```
312 double segDist(P s, P e, P p) {
BD2     if (s==e) return (p-s).len();
564     auto d = (e-s).dist2(), t = min(d,max(.01,(p-s).dot(e-s)
));
9E6     return ((p-s)*d-(e-s)*t).len()/d;
824 }
```

```
638 double dist_convex(vector<P> p, vector<P> q) {
116     for (P& i : p) i = i * -1;
029     auto s = minkowskiSum(p, q);
B2F     if (isInside(s, P(0,0))) return 0;
49D     double ans = 1e18; // INF
1DC     int ssz = s.size();
F26     for(int i = 0; i < ssz; i++) {
06B         int j = (i+1)%ssz;
6AA         ans = min(ans, segDist(s[i], s[j], P(0,0)));
809     }
BA7     return ans;
EE2 }
```

4.2 mincircle

```
16E typedef PT<double> P;
406 double ccRadius(P& A, P& B, P& C) {
D82     return (B-A).len()* (C-B).len()* (A-C).len() /
83F         abs((B-A).cross(C-A))/2.0;
6BC }
```

```
07B P ccCenter(P& A, P& B, P& C) {
28A     P b = C-A, c = B-A;
680     return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)
/2;
24F }
D41 // mt19937 rng(chrono::steady_clock::now().
time_since_epoch().count());
006 pair<P, double> mec(vector<P>& pts) {
03B     shuffle(begin(pts), end(pts), rng);
185     P o = pts[0];
FCB     const double EPSS = 1+1e-8;
996     double r = 0;
743     for(int i = 0; i < pts.size(); i++) if((o-pts[i]).len()
> r * EPSS) {
759         o = pts[i], r = 0;
6E9         for(int j = 0; j < i; j++) if((o-pts[j]).len() > r *
EPSS) {
FD8             o = (pts[i]+pts[j])/2.0;
DAE             r = (o - pts[i]).len();
FE0             for(int k = 0; k < j; k++) if((o-pts[k]).len() > r *
EPSS) {
ECD                 o = ccCenter(pts[i],pts[j],pts[k]);
DAE                 r = (o - pts[i]).len();
5AB             }
102         }
509     }
645     return {o, r};
8E0 }
```

4.3 closestpair

```
531 pii ClosestPair(vector<PT<ll>>& pts) {
62D     ll dist = (pts[0]-pts[1]).dist2();
1A2     pii ans(0, 1);
CA0     int n = pts.size();
47B     vector<int> p(n);
469     iota(begin(p), end(p), 0);
```

```

F85  sort(p.begin(), p.end(), [&](int a, int b) { return pts[
a].x < pts[b].x; });
3E7  set<pii> points;
526  auto sqr = [] (long long x) -> long long { return x * x;
};
637  for(int l = 0, r = 0; r < n; r++) {
39E      while(sqr(pts[p[r]].x - pts[p[l]].x) > dist) {
9F7          points.erase(pii(pts[p[l]].y, p[l]));
63B          l++;
FFB      }
7CF      ll delta = sqrt(dist) + 1;
92B      auto itl = points.lower_bound(pii(pts[p[r]].y - delta,
-1));
6C3      auto itr = points.upper_bound(pii(pts[p[r]].y + delta,
n + 1));
901      for(auto it = itl; it != itr; it++) {
C66          ll curDist = (pts[p[r]] - pts[it->second]).dist2();
OCF          if(curDist < dist) {
1E3              dist = curDist;
8E7              ans = pii(p[r], it->second);
AE0          }
C07      }
EEA      points.insert(pii(pts[p[r]].y, p[r]));
9B6  }
EBB  if(ans.first > ans.second)
6FE      swap(ans.first, ans.second);
BA7  return ans;
D1D }

```

4.4 point

```

D41 // hypot, atan2, gcd
1D5 const double PI = acos(-1);
48B template <class T> int sgn(T x) { return (x > 0) - (x < 0)
; }
67A template<typename T>
C19 struct PT{
645     T x, y;
7AB     PT(T x=0, T y=0) : x(x), y(y){}
425     bool operator < (PT o) const { return tie(x,y) < tie(o.x
,o.y); }
987     bool operator == (PT o) const { return tie(x,y) == tie(o
.x,o.y); }
EB1     PT operator + (PT o) const { return PT(x+o.x,y+o.y); }
D02     PT operator - (PT o) const { return PT(x-o.x,y-o.y); }
EAB     PT operator * (T k) const { return PT(x*k,y*k); }
593     PT operator / (T k) const { return PT(x/k,y/k); }
6F1     T cross(PT o) const { return x*o.y - y*o.x; }
F53     T cross(PT a, PT b) const { return (a-*this).cross(b-*
this); }
A65     T dot(PT o) const { return x*o.x + y*o.y; }
F68     T dist2() const { return x*x + y*y; }
0A7     double len() const { return hypot(x,y); }
559     PT perp() const { return PT(-y,x); }
2B2     PT rotate(double a) const { return PT(x*cos(a)-y*sin(a),
x*sin(a)+y*cos(a)); }
950     int quad() { return (x<0)^3*(y<0); }
CF5     bool ccw(PT<T> q, PT<T> r){ return (q-*this).cross(r-q)
> 0; }
E63 };
67A template<typename T>
306 bool cmp_ang(PT<T> p, PT<T> q){
6E2     if (p.quad() != q.quad()) return p.quad() < q.quad();
B84     return q.ccw(PT<T>(0,0),p);
FD9 }

BA2 ostream &operator<<(ostream &os, const PT<int> &p) {
D80     return os << "(" << p.x << ", " << p.y << ")";

```

4.5 convexhull

```

D41 // retorna poligono no sentido anti horario, trocar pra <
se quiser horario
67A template<typename T>
F1C vector<PT<T>> convexHull(vector<PT<T>>& pts, bool sorted =
false){
EC1     if(!sorted) sort(begin(pts),end(pts));
661     vector<PT<T>> h;
89A     h.reserve(pts.size() + 1);
7A4     for(int it = 0; it < 2; it++){
5C9         int start = h.size();
BB9         for(PT<T>& c : pts){
C31             while((int)h.size() >= start + 2){
184                 PT<T> a = h[h.size()-2], b = h.back();
D41                 // '>' pra nao descartar pontos colineares
630                 if((b-a).cross(c-a) > 0) break;
BFB                 h.pop_back();
748             }
476             h.push_back(c);
A0A         }
A8B         reverse(begin(pts),end(pts));
BFB         h.pop_back();
AAD     }
E30     if(h.size() == 2 && h[0] == h[1]) h.pop_back();
81C     return h;
AD6 }

D41 // nao funciona se tem pontos colineares!!!!
D41 // considera ponto na aresta como dentro
67A template<typename T>
319 bool isInside(vector<PT<T>>& hull, PT<T> p) {
ACE     int n = hull.size();
8DD     PT<T> v0 = p - hull[0], v1 = hull[1] - hull[0], v2 =
hull[n-1] - hull[0];
6A6     if(v0.cross(v1) > 0 || v0.cross(v2) < 0){
D1F         return false;
769     }
9D9     int l = 1, r = n - 1;
E55     while(l != r){
264         int mid = (l + r + 1) / 2;
549         PT<T> v0 = p - hull[0], v1 = hull[mid] - hull[0];
2C9         if(v0.cross(v1) < 0)
229             l = mid;
295         else
982             r = mid - 1;
406     }
87D     v0 = hull[(l+1)%n] - hull[l], v1 = p - hull[l];
57C     return v0.cross(v1) >= 0;
76A }

D41 // poligonos
36E ll polygon_area_db(const vector<Point>& poly){
604     ll area = 0;
F6A     for(int i = 0, n = (int)poly.size(); i < n; ++i) {
A91         int j = i + 1 == n ? 0 : i + 1;
140         area += cross(poly[i], poly[j]);
A87     }
199     return abs(area);
315 }
D41 // Teorema de Pick para lattice points
D41 // Area = insidePts + boundPts/2 - 1
D41 // 2A - b + 2 = 2i
D41 // usar gcd dos lados pra contar bound pts
E82 ll cntInsidePts(ll area_db, ll bound){
97C     return (area_db + 2LL - bound)/2;

```

CDC }

5 data-structures

5.1 maxqueue

```

2D5 template <class T, class C = less<T>>
CF4 struct MaxQueue {
8E5     MaxQueue() { clear(); }
75A     void clear() {
B08         id = 0;
4B6         q.clear();
5DD     }
3FC     void push(T x) {
9FB         pair<int, T> nxt(1, x);
285         while(q.size() > id && cmp(q.back().second, x)) {
7A6             nxt.first += q.back().first;
AB1             q.pop_back();
F02         }
986         q.push_back(nxt);
FAF     }
577     T gry() { return q[id].second; }
42D     void pop() {
62D         q[id].first--;
097         if(q[id].first == 0) { id++; }
3CB     }
BF2 private:
4A7     vector<pair<int, T>> q;
53E     int id;
1B8     C cmp;
961 };

```

5.2 segtree-lazy

```

D41 // Lazy SegTree ta meio desatualizado mas sei modificar
35A const int mx = 2e5+5;
131 vector<ll> seg(4*mx);
3F0 vector<ll> lazy(4*mx,0);
C6B vector<ll> nums(mx);
3B9 int n,q;

353 void build(int l = 0, int r = n-1, int idx = 0){
893     if(l == r){
389         seg[idx] = nums[l];
824         lazy[idx] = 0;
505         return;
821     }
EE4     int m = (l+r)/2;
EED     int left = 2*idx+1;
680     int right = 2*idx+2;
84F     build(l,m,left);
C3B     build(m+1,r,right);
5B2     seg[idx] = seg[left] + seg[right];
67B }

175 void prop(int l = 0, int r = n-1, int idx = 0){
A39     seg[idx] += (ll)(r-l+1)*lazy[idx];
579     if(l != r){ // nao for folha
EED         int left = 2*idx+1;
680         int right = 2*idx+2;
774         lazy[left] += lazy[idx];
AAE         lazy[right] += lazy[idx];
4D4     }
824     lazy[idx] = 0;

```

```

5EF }
BFE void update(int L, int R, ll val, int l = 0, int r = n-1,
    int idx = 0){
483     if(R < l || L > r) return;
E1B     prop(l,r,idx);
761     if(L <= l && r <= R){
F6E         lazy[idx] = val;
E1B         prop(l,r,idx);
64B     }
4E6     else{
EE4         int m = (l+r)/2;
EED         int left = 2*idx+1;
680         int right = 2*idx+2;
AC4         update(L,R,val,l,m,left);
CDB         update(L,R,val,m+1,r,right);
5B2         seg[idx] = seg[left] + seg[right];
01D     }
375 }

6F7 ll query(int L, int R, int l = 0, int r = n-1, int idx =
    0){
E1B     prop(l,r,idx);
1BA     if(R < l || L > r) return 0;
761     if(L <= l && r <= R){
A9D         return seg[idx];
2BD     }
EE4     int m = (l+r)/2;
EED     int left = 2*idx+1;
680     int right = 2*idx+2;
1FE     return query(L,R,l,m,left) + query(L,R,m+1,r,right);
F3A }

```

5.3 kd-tree

```

605 template<class T, const int D = 2>
21E class KD_Tree {
673 public:
E72     using PT = array<T, D>;

BF2     struct Node {
BB6         PT point;
795         Node *left, *right;
41D     };

D02     void init(vector<PT> pts) {
BA3         if(pts.size() == 0) {
505             return;
6AD         }
443         int n = 0;
9FC         tree.resize(2 * pts.size());
C8F         build<0>(pts.begin(), pts.end(), n);
D41         //assert(n <= (int) tree.size());
7BD     }

4C7     pair<T, PT> nearestNeighbor(PT point) {
D41         // assert(tree.size() > 0);
916         pair<T, PT> ans;
465         ans.first = 8.1e18; // BE CAREFUL HERE
010         removedEqual = false;
2B2         nearestNeighbor<0>(&tree[0], point, ans);
BA7         return ans;
8E7     }
BF2 private:
8A5     vector<Node> tree;

C5A     template<const int d>
256     Node* build(auto l, auto r, int &n) {

```

```

0B1         if(l >= r) {
EA9             return NULL;
248         }
AB9         int id = n++;
652         if(r - l == 1) {
270             tree[id].left = tree[id].right = NULL;
547             tree[id].point = *l;
C82         } else {
263             auto mid = l + ((r - l) / 2);
FF2             nth_element(l, mid - 1, r, [](const PT &u,
const PT &v) { return u[d] < v[d]; });
6FF             tree[id].point = *(mid - 1);
D41             // BE CAREFUL!
D41             // DO EVERYTHING BEFORE BUILDING THE LOWER
PART!
2B4             tree[id].left = build<(d+1)%D>(l, mid-1, n);
57F             tree[id].right = build<(d+1)%D>(mid, r, n);
4B5         }
FAB         return &tree[id];
BEC     }
491     bool removedEqual;
C5A     template<const int d>
F9C     void nearestNeighbor(Node* node, PT point, pair<T, PT>
&ans) {
260         if(!node) {
505             return;
08D         }
7DC         const T dist = sqrtDist(point, node->point);
9A9         if((point != node->point || removedEqual) && dist
< ans.first) {
D41             // THIS WAS FOR A PROBLEM
D41             // THAT YOU DON'T CONSIDER THE DISTANCE TO
ITSELF!
5BE             ans = {dist, node->point};
BB0         } else if(point == node->point) {
60E             removedEqual = true;
7B6         }
B67         T delta = point[d] - node->point[d];
5DE         if(delta <= 0) {
871             nearestNeighbor<(d+1)%D>(node->left, point,
ans);
09A             if(ans.first > delta * delta) {
008                 nearestNeighbor<(d+1)%D>(node->right,
point, ans);
74C             }
A87         } else {
008             nearestNeighbor<(d+1)%D>(node->right, point,
ans);
09A             if(ans.first > delta * delta) {
871                 nearestNeighbor<(d+1)%D>(node->left, point
, ans);
9F1             }
212         }
B7A     }

363     T sqrtDist(const PT &u, const PT &v) {
966         T ans = 0;
772         for(int i = 0; i < D; i++) {
ED8             T delta = u[i] - v[i];
928             ans += delta * delta;
B96         }
BA7         return ans;
1B7     }
03D };

```

5.4 colorupdate

```

DEE template <class InfoType = int32_t, class RangeType =
    int32_t>
F1D struct ColorUpdate {
673 public:
3D4     struct Range {
032         Range(RangeType _l = 0) : l(_l) {}
9A5         Range(RangeType _l, RangeType _r, InfoType _v) : l
(_l), r(_r), v(_v) {}
A41         RangeType l, r;
0C1         InfoType v;

DBA         bool operator < (const Range &b) const { return l
< b.l; }
F3C     };

2A4     vector<Range> erase(RangeType l, RangeType r) {
D02         vector<Range> ans;
AF6         if(l >= r) return ans;
EFB         auto it = ranges.lower_bound(l);
A43         if(it != ranges.begin()) {
049             it--;
86E             if(it->r > l) {
BF0                 auto cur = *it;
649                 ranges.erase(it);
206                 ranges.insert(Range(cur.l, l, cur.v));
FD9                 ranges.insert(Range(l, cur.r, cur.v));
C39             }
FC5             }
7D1             it = ranges.lower_bound(r);
A43             if(it != ranges.begin()) {
049                 it--;
E91                 if(it->r > r) {
BF0                     auto cur = *it;
649                     ranges.erase(it);
D16                     ranges.insert(Range(cur.l, r, cur.v));
3A3                     ranges.insert(Range(r, cur.r, cur.v));
2E9                 }
7DB             }
444             for(it = ranges.lower_bound(l); it != ranges.end()
&& it->l < r; it++) {
3BD                 ans.push_back(*it);
992             }
E1D             ranges.erase(ranges.lower_bound(l), ranges.
lower_bound(r));
BA7             return ans;
39A         }

B86     vector<Range> upd(RangeType l, RangeType r, InfoType v
) {
F3D         auto ans = erase(l, r);
ED3         ranges.insert(Range(l, r, v));
BA7         return ans;
F56     }

8DB     bool exists(RangeType x) {
3D4         auto it = ranges.upper_bound(x);
ED2         if(it == ranges.begin()) return false;
049         it--;
03C         return it->l <= x && x < it->r;
E44     }
BF2 private:
4A2         set<Range> ranges;
7C5 };

```

5.5 sparsetable

```

D41 // use const, principalmente no log
D41 // log = maior msb a[i]

```



```

B55 const int LOG = 20;
D41 // query [L,R]
67A template<typename T>
7E9 struct SparseTable{
1A8     int n;
F9A     vector<vector<T>> table;
8EE     SparseTable() {}
407     SparseTable(vector<T>& v){
D6C         n = v.size();
310         table = vector<vector<T>>(n);
760         table[0] = v;
F89         for(int lg = 0; lg < LOG; lg++){
603             for(int i = 0; i < n; i++){
5FE                 if (i + (1 << lg) >= n) break;
F88                 table[lg + 1][i] = min(table[lg][i], table[lg][i +
(1 << lg)]);
8A6             }
F11         }
A47     }
C90     T qry(int l, int r){
884         int k = 31 - __builtin_clz(r-l+1);
DC9         return min(table[k][l], table[k][r - (1 << k) + 1]);
6EF     }
EAB };

```

5.6 fenwick-tree

```

35A const int mx = 2e5+5;
C14 ll bit[mx];
3B9 int n, q;

2AF ll qry(int i){ // [1,i] 1 indexado
B73     ll ret = 0;
FBD     for(; i > 0; i -= i & -i)
7CA         ret += bit[i];
EDF     return ret;
1D1 }

9A6 void increment(ll i, ll v){ // 1 indexado (+v)
A6D     for(; i <= n; i += i & -i)
963         bit[i] += v;
28C }

```

5.7 dynamic-median

```

4B4 const ll inf = 1e18 + 5;
283 struct DynamicMedian{
E01     multiset<ll> left, right;
A2C     ll leftsum = 0, rightsum = 0;
6DE     ll get(){
D41         // if(left.empty()) return -1; // cuidar aqui
668         return *left.rbegin();
833     }
F4C     ll qry(){ // somatorio de distancia absoluta pra mediana
239         ll m = get();
D41         // if(m == -1) return -1;
099         return left.size()*m - leftsum + rightsum - right.size
()*m;
AF5     }
01A     void fix(){
D41         // (L,R) ou (L+1,R)
770         while(right.size() + 1 < left.size()){
D41             // tirar do l e colocar no r
BF5             auto lst = --left.end();
FC3             rightsum += *lst;
FCD             leftsum -= *lst;

```

```

64F         right.insert(*lst);
47B         left.erase(lst);
761     }
49B     while(right.size() > left.size()){
D41         // tirar do r e colocar no l
D50         leftsum += *right.begin();
9C1         rightsum -= *right.begin();
449         left.insert(*right.begin());
5E3         right.erase(right.begin());
C41     }
78A }

6CD void insert(ll x){
6A4     ll m = (left.empty() ? inf : get());
D3B     if(x <= m){
BE2         left.insert(x);
00B         leftsum += x;
23E     }else{
AD2         right.insert(x);
D20         rightsum += x;
941     }
39B     fix();
3E3 }
C95 void erase(ll x){
F83     auto l = left.find(x);
FE1     if(l != left.end()){
CDF         leftsum -= *l;
898         left.erase(l);
138     }
4E6     else{
6DA         auto r = right.find(x);
519         rightsum -= *r;
74F         right.erase(r);
DD8     }
39B     fix();
326 }
781 };

5.8 segtree-topdown

D41 // SegTree
35A const int mx = 2e5 + 5;
ADA ll seg[4*mx];
56A ll a[mx];
3B9 int n,q;
4B5 ll join(ll a, ll b){
534     return a+b;
2D6 }

353 void build(int l = 0, int r = n - 1, int idx = 0){
893     if(l == r){
B28         seg[idx] = a[l];
505         return;
3A6     }
AE0     int mid = (l + r)/2;
8E4     build(l, mid, 2*idx + 1);
2F1     build(mid + 1, r, 2*idx + 2);
AED     seg[idx] = join(seg[2*idx + 1], seg[2*idx + 2]);
830 }

6F7 ll query(int L, int R, int l = 0, int r = n - 1, int idx =
0){
1BA     if(R < l || L > r) return 0; // elemento neutro
FA9     if(L <= l && r <= R) return seg[idx];
AE0     int mid = (l + r)/2;
9D8     return join(query(L, R, l, mid, 2*idx + 1), query(L, R,
mid + 1, r, 2*idx + 2));
579 }

```

```

61D void update(int i, ll val, int l = 0, int r = n - 1, int
idx = 0){
893     if(l == r){
873         seg[idx] = val;
505         return;
741     }
AE0     int mid = (l + r)/2;
8B0     if(i <= mid) update(i, val, l, mid, 2*idx + 1);
AAB     else update(i, val, mid + 1, r, 2*idx + 2);
AED     seg[idx] = join(seg[2*idx + 1], seg[2*idx + 2]);
EA9 }

```

6 etc

6.1 mo

Mo em arvore: queries em caminhos. Olhar carinhas que aparecem quantidade impar de vezes
Seja u menor tin
se u for lca de v: range = [tin[u], tin[v]]
c.c.: range = [tout[u], tin[v]] U [tin[lca],tin[lca]] -- tratar lca separado quando tiver respondendo

```

D41 // Mo apelao
D41 // Ordering based on the Hilbert curve
905 inline int64_t hilbertOrder(int x, int y, int pow, int
rotate){
51A     if(pow == 0) return 0;
A6E     int hpow = 1 << (pow - 1);
01F     int seg = (x < hpow) ? ( (y < hpow) ? 0 : 3) : ( (y <
hpow) ? 1 : 2);
6D9     seg = (seg + rotate) & 3;
F96     const int rotateDelta[4] = {3, 0, 0, 1};
D0B     int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
385     int nrot = (rotate + rotateDelta[seg]) & 3;
7AC     int64_t subSquareSize = int64_t(1) << (2*pow - 2);
98B     int64_t ans = seg * subSquareSize;
B22     int64_t add = hilbertOrder(nx, ny, pow - 1, nrot);
7C5     ans += (seg == 1 || seg == 2) ? add : (subSquareSize -
add - 1);
BA7     return ans;
E55 }
670 struct Query{
738     int l, r, idx;
CE8     int64_t ord;
36F     Query(int l, int r, int idx) : l(l), r(r), idx(idx) {
11F         ord = hilbertOrder(l, r, 21, 0);
B25     }
E07     bool operator < (Query &other){
328         return ord < other.ord;
007     }
912 };
D41 // Mo normal
A47 const int MXN = 2e5;
D24 const int B = sqrt(MXN) + 1;
670 struct Query {
738     int l, r, idx;
665     bool operator<(Query o) const{
736         return make_pair(l / B, ((l/B) & 1) ? -r : r) <
make_pair(o.l / B, ((o.l/B) & 1) ? -o.r : o.r);
A4B     }
868 };

CD3 ll a[MXN];

```

```
C3D ll resp = 0;
EAC void add(int x);
292 void remove(int x);

E8D int main(){
C5F     int n, q; cin >> n >> q;
830     for(int i = 0; i < n; i++){
788         cin >> a[i];
240     vector<Query> queries;
ABF     for(int i = 0; i < q; i++){
5DA         int l, r; cin >> l >> r;
29D         queries.push_back(Query(l-1,r-1,i));
08A     }
671     sort(begin(queries),end(queries));
153     vector<ll> answers(q);
70A     int L = 0, R = -1;
ECD     for(Query qr : queries){
981         while (L > qr.l) add(--L);
FF1         while (R < qr.r) add(++R);
167         while (L < qr.l) remove(L++);
A20         while (R > qr.r) remove(R--);
620         answers[qr.id] = resp;
1BD     }
EDF     for(int i = 0; i < q; i++){
04D         cout << answers[i] << "\n";
459 }
```

6.2 bitset

```
D41 // Comando hash de codigo :w !sha256sum

D41 // Bitset operations
99C __builtin_popcount(int x);
65C __builtin_popcountll(ll x);
302 const int SZ = 1e6;
596 bitset<SZ> b;
155 b.reset(); // 00 ... 00
29C b.set(); // 11 ... 11
98B b.flip();
C72 b._Find_first(); // retorna SZ se nao tiver
235 b._Find_next(i);
B9B b.to_ulong();
ED7 b.to_string();
E9E b.count();

D41 // rng
C8A mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
8A3 shuffle(begin(x),end(x),rng);
598 uniform_int_distribution<int>(0,x)(rng);
```

6.3 ternary-search

```
D5D double f(double t){
D41     // alguma funcao unimodal -> maximo ou minimo
D41     //      /\
D41     //   /\  \
D41     //  __/  \__
392 }

CD1 double tern_search(double l, double r){
D1A     for(int it = 0; it < 300; it++){
A50         double m1 = l + (r-l)/3;
7E7         double m2 = r - (r-l)/3;
```

```
87F     double f1 = f(m1), f2 = f(m2);
19D     if(f1 < f2) l = m1; //change to > to find maximum
064     else r = m2;
ABB     }
792     return l;
058 }
D41 // golden section search
036 double gss(double a, double b) {
2B1     const double r = (sqrt(5)-1)/2, eps = 1e-7;
B87     double x1 = b - r*(b-a), x2 = a + r*(b-a);
47D     double f1 = f(x1), f2 = f(x2);
F2C     for(int it = 0; it < 250 && b-a > eps; it++){
F4D         if (f1 < f2) { //change to > to find maximum
DA5             b = x2; x2 = x1; f2 = f1;
DFB             x1 = b - r*(b-a); f1 = f(x1);
451         } else {
D6E             a = x1; x1 = x2; f1 = f2;
815             x2 = a + r*(b-a); f2 = f(x2);
505         }
3F5         return a;
894 }
D41 // retorna mais a esquerda no empate
E15 int int_tern_search(int l, int r){
7AD     int lo = l - 1, hi = r;
FF8     while(hi - lo > 1){
591         int m = (lo+hi)/2;
42D         if(f(m) < f(m+1)){ //
04A             lo = m;
FA4         }else{
89E             hi = m;
DAB         }
30C     }
480     return lo + 1;
88D }
```

6.4 formulas

Soma de pg: = $a1 \cdot (q^n - 1) / (q - 1)$
Soma dos impares = n^2
Soma de i^2 : = $n(n+1)(2n+1)/6$

Number theory:
gcd(a+k*b,b) = gcd(a,b)
phi(n) = #coprimos com n <=n
phi(n) >= log2(n)
phi(phi(n)) <= n/2
 $a^{\phi(n)} \equiv 1 \pmod n$
 $a^{-1} \equiv a^{(m-2) \bmod m}$
Conjectura de Goldbach's: todo numero par $n > 2$ pode ser representado com $n = a + b$ onde a e b sao primos
Twin prime: existem infinitos pares p, p + 2 onde ambos sao primos
Legendre's: sempre tem um primo entre n^2 e $(n+1)^2$
Lagrange's: todo numero inteiro pode ser inscrito como a soma de 4 quadrados
Wilson's: n eh primo quando $(n-1)! \bmod n = n - 1$
McNugget: Para dois coprimos x, y a quantidade de inteiros que nao pode ser escrito como $ax + by$ eh $(x-1)(y-1)/2$, o maior inteiro que nao consegue eh $x*y-x-y$

Geometria:
 $V+F=A+2$
Formula de heron: $\sqrt{s(s-a)(s-b)(s-c)}$, s = semiperimetro
Volume de esfera: $4/3\pi r^3$
Area da esfera: $4\pi r^2$
Volume tetraedro: $1^3 \cdot \sqrt{2}/12$
Projecao u em v = $(u \cdot v) / (v \cdot v) \cdot v$

6.5 composite

number	divisors	factorization
120	16	$2^3 \cdot 3 \cdot 5$
1.260	36	$2^2 \cdot 3^2 \cdot 5 \cdot 7$
10.080	72	$2^5 \cdot 3^2 \cdot 5 \cdot 7$
110.880	144	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$
1.081.080	256	$2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
10.810.800	480	
$2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$		
110.270.160	800	
$2^4 \cdot 3^4 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$		
1.102.701.600	1440	
$2^5 \cdot 3^4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$		

7 graph

7.1 kosaraju

```
D41 // Kosaraju
F9F const int ms = 1e5 + 5;
9CB vector<int> G[ms], Gt[ms];
9F5 vector<int> id, order, root;
B57 vector<bool> vis;
1A8 int n;
CB7 void dfs1(int u) { // ordem de saida
B9C     vis[u] = true;
E44     for(int v : G[u])
C2D         if(!vis[v])
3BA             dfs1(v);
C75     order.push_back(u);
7A7 }
6A1 void dfs2(int u, int idx) {
846     id[u] = idx;
51D     for(int v : Gt[u])
BBC         if(id[v] == -1)
039             dfs2(v,idx);
594 }
D41 // retorna quantidade de componentes
973 int kosaraju() {
543     vis.assign(n,false);
C2F     id.assign(n,-1);
830     for(int i = 0; i < n; i++){
F69         if(!vis[i])
6D1             dfs1(i);
3B9         reverse(begin(order),end(order));
503         int idx = 0;
87A         for(int u : order)
98B             if(id[u] == -1)
B4B                 dfs2(u, idx++), root.push_back(u);
449         return idx;
359 }
```

7.2 dinic

```
D41 //O(V^2 E), O(E sqrtV) in unit networks
67A template<typename T>
E9B struct Edge {
791     int to;
D90     T cap, flow;
112     Edge(int to, T cap) : to(to), cap(cap), flow(0) {}
338     T res() const { return cap - flow; }
```



```

E92 };
67A template<typename T>
14D struct Dinic {
4D8     using E = Edge<T>;
05B     int m = 0, n;
976     vector<E> ed;
789     vector<vector<int>>> g;
3B3     vector<int> dist, ptr;
085     Dinic(int n) : n(n), g(n), dist(n), ptr(n) {}
555     void add_edge(int u, int v, T cap) {
DF5         if(u != v) {
65F             ed.emplace_back(v, cap);
F0D             ed.emplace_back(u, 0);
329             g[u].emplace_back(m++);
4C9             g[v].emplace_back(m++);
688         }
680     }
123     bool bfs(int s, int t) {
FD5         fill(begin(dist), end(dist), n + 1);
A93         dist[s] = 0;
0B4         queue<int> q({s});
14D         while(!q.empty()) {
E4A             int u = q.front();
833             q.pop();
4B5             if(u == t) break;
10F             for(int id : g[u]) {
22E                 E& e = ed[id];
D9E                 if(e.res() > 0 && dist[e.to] > dist[u] + 1) {
29B                     dist[e.to] = dist[u] + 1;
A78                     q.emplace(e.to);
08B                 }
3D5             }
53A         }
8B6         return dist[t] != n + 1;
B70     }
D6A     T dfs(int u, int t, T flow) {
3B2         if(u == t || flow == 0) {
99D             return flow;
B48         }
C53         for(int& i = ptr[u]; i < (int)g[u].size(); ++i) {
04D             E& e = ed[g[u][i]];
A8D             E& eRev = ed[g[u][i] ^ 1];
AF4             if(dist[e.to] == dist[eRev.to] + 1) {
4CF                 T amt = min(flow, e.res());
F17                 if(T ret = dfs(e.to, t, amt)) {
786                     e.flow += ret;
CEB                     eRev.flow -= ret;
EDF                     return ret;
95A                 }
C73             }
B2E         }
BB3         return 0;
C50     }
9C4     T max_flow(int s, int t) {
C80         T total = 0;
8CE         while(bfs(s, t)) {
197             fill(begin(ptr), end(ptr), 0);
419             while(T flow = dfs(s, t, numeric_limits<T>::max()))
{
810                 total += flow;
136             }
70C         }
994         return total;
EB4     }
159     bool cut(int u) const { return dist[u] == n + 1; }
064 };

```

7.3 kruskal

```

6D0 int n = 1e5;
E81 DSU dsu = DSU(n+5);
C5C using tp = tuple<ll,int,int>
820 vector<tp> edges(e);
0F4 for(auto& [w, u, v] : edges){
52D     cin >> u >> v >> w;
7B8 }
F80 sort(begin(edges),end(edges));
854 ll cost = 0;
AC9 int cnt = 0;
2A3 for(auto [w, u, v] : edges){
AC3     if(dsu.unite(u,v)){
45F         cost += w;
F65         cnt++;
6CE     }
0B3 }
D41 // if(cnt != n-1) cout << "IMPOSSIBLE" << br;

```

7.4 smallToLarge

```

D41 // nlogn small to large (offline)

1C7 int tin[mxn],tout[mxn];
9E5 vector<int> g[mxn];
05B int clr[mxn];
A0E int inv[mxn];
5BC int sz[mxn];
910 int cnt[mxn];
A10 void calcsz(int u, int p){
267     sz[u] = 1;
73E     tin[u] = ++timer;
5E0     inv[timer] = u;
D76     for(int v : g[u]) if(v != p){
086         calcsz(v,u);
CC3         sz[u] += sz[v];
91B     }
4F8     tout[u] = timer;
CAA }

D78 void dfs(int u, int p, bool mata = 0){
599     int big = -1;
A8C     int howBig = 0;
D76     for(int v : g[u]) if(v != p){
987         if(sz[v] > howBig){
737             big = v;
005             howBig = sz[big];
E24         }
F73     }
181     for(int v : g[u]) if(v != p && v != big){
427         dfs(v,u,1);
02D     }
A5F     if(big != -1)
003         dfs(big,u,0);
181     for(int v : g[u]) if(v != p && v != big){
9AD         for(int l = tin[v]; l <= tout[v]; l++){
3BC             int who = inv[l];
029             cnt[clr[who]]++;
7C4         }
D64     }
D34     cnt[clr[u]]++;
D41     // solve queries aqui
D41     //
35E     if(mata){
21E         for(int l = tin[u]; l <= tout[u]; l++){
3BC             int who = inv[l];
733             cnt[clr[who]]--;

```

```

C46         }
4D2     }
51F }

```

7.5 policyBased

```

774 #include <ext/pb_ds/assoc_container.hpp> // Common file
30F #include <ext/pb_ds/tree_policy.hpp> // Including
tree_order_statistics_node_update
0D7 using namespace __gnu_pbds;
DBF typedef tree<int, null_type, less<int>, rb_tree_tag,
39F tree_order_statistics_node_update> ordered_set;
609 ordered_set X;
766 X.insert(1); X.find_by_order(0); // Acha a key na ordem Y
9C3 X.order_of_key(-5); // Acha a ordem da key Y
FEC end(X), begin(X);

```

7.6 centroid

```

A3D bool vis[mxn];
03A int par[mxn]; // parent na centroid tree, par do primeiro
centroid eh -1
5BC int sz[mxn];
3A0 int find_centroid(int u, int p, int s){
F5E     for(int v : g[u]) if(v != p && !vis[v] && sz[v] > s /
2){
062         return find_centroid(v,u,s);
8E4     }
03F     return u;
1C8 }

39F void calc_size(int u, int p){
D3C     if(vis[u]){
94A         sz[u] = 0;
505         return;
039     }
267     sz[u] = 1;
D76     for(int v : g[u]) if(v != p){
51D         calc_size(v,u);
CC3         sz[u] += sz[v];
6A2     }
C4D }

889 void init_centroid_tree(int u, int p = -1){
716     calc_size(u,u);
B3F     int c = find_centroid(u,u,sz[u]);
929     vis[c] = 1;
14D     par[c] = p;
45B     for(int v : g[c]) if(!vis[v]){
F15         init_centroid_tree(v,c);
811     }
E67 }

```

7.7 articulation

```

14E int n, m;
21F const int mxn = 1e5 + 5;
9E5 vector<int> g[mxn];
B1B int tin[mxn], low[mxn];
A34 vector<int> art;
813 int timer = 1;

FB6 void dfs(int u, int p){
406     tin[u] = timer++;

```

```

E99 low[u] = tin[u];
612 int ch = 0;
E67 int fw = 0;
D76 for(int v : g[u]) if(v != p){
85A     if(tin[v]) // lowlink direta
4EC         low[u] = min(tin[v], low[u]);
4E6     else{
95E         dfs(v, u);
AD9         fw++;
E7F         low[u] = min(low[v], low[u]);
918         ch = max(low[v], ch);
534     }
B36 }
9EC if(u == p && fw > 1) art.push_back(u);
7BD else if(u != p && ch && tin[u] <= ch) art.push_back(u);
2E6 }

```

7.8 mcmf

```

39C template<typename Cap, typename Cost>
6F3 struct MCMF{
523     const Cost INF = numeric_limits<Cost>::max();
E9B     struct Edge {
791         int to;
F23         Cap cap, flow;
CB9         Cost cost;
A6D         Edge(int to, Cap cap, Cost cost) : to(to), cap(cap),
flow(0), cost(cost) {}
8FD         Cap res() const { return cap - flow; }
1C8     };
05B     int m = 0, n;
321     vector<Edge> edges;
789     vector<vector<int>> g;
ADE     vector<Cap> neck;
35F     vector<Cost> dist, pot;
E3B     vector<int> from;
8C3     MCMF(int n) : n(n), g(n), neck(n), pot(n) {}
780     void add_edge(int u, int v, Cap cap, Cost cost) {
DF5         if(u != v) {
C1A             edges.emplace_back(v, cap, cost);
FB6             edges.emplace_back(u, 0, -cost);
329             g[u].emplace_back(m++);
4C9             g[v].emplace_back(m++);
A0D         }
451     }
E3A     void spfa(int s) {
E59         vector<bool> inq(n, false);
0B4         queue<int> q({s});
14D         while(!q.empty()) {
352             auto u = q.front();
833             q.pop();
E0A             inq[u] = false;
B35             for(auto e : g[u]) {
5F3                 auto ed = edges[e];
CDD                 if(ed.res() == 0) continue;
B5D                 Cost w = ed.cost + pot[u] - pot[ed.to];
BAC                 if(pot[ed.to] > pot[u] + w) {
CB4                     pot[ed.to] = pot[u] + w;
863                     if(!inq[ed.to]) {
113                         inq[ed.to] = true;
FC4                         q.push(ed.to);
586                     }
5CB                 }
B95             }
3F4         }
58B     }
9DB     bool dijkstra(int s, int t) {
EF2         dist.assign(n, INF);

```

```

0B5         from.assign(n, -1);
2DE         neck[s] = numeric_limits<Cap>::max();
C6F         using ii = pair<Cost, int>;
D9A         priority_queue<ii, vector<ii>, greater<ii>> pq;
6BD         pq.push({dist[s] = 0, s});
502         while(!pq.empty()) {
E18             auto [d_u, u] = pq.top();
716             pq.pop();
624             if(dist[u] != d_u) continue;
829             for(auto i : g[u]) {
D26                 auto ed = edges[i];
B5D                 Cost w = ed.cost + pot[u] - pot[ed.to];
C6C                 if(ed.res() > 0 && dist[ed.to] > dist[u] + w) {
A3F                     from[ed.to] = i;
1A3                     pq.push({dist[ed.to] = dist[u] + w, ed.to});
68F                     neck[ed.to] = min(neck[u], ed.res());
7D8                 }
D59             }
291         }
85D         return dist[t] < INF;
CCB     }
47B     pair<Cap, Cost> mcmf(int s, int t, Cap k =
numeric_limits<Cap>::max()) {
717         Cap flow = 0;
247         Cost cost = 0;
FBB         spfa(s);
C28         while(flow < k && dijkstra(s, t)) {
D00             Cap amt = min(neck[t], k - flow);
0D7             for(int v = t; v != s; v = edges[from[v] ^ 1].to) {
2AE                 cost += edges[from[v]].cost * amt;
3B4                 edges[from[v]].flow += amt;
60F                 edges[from[v] ^ 1].flow -= amt;
48F             }
2E8             flow += amt;
F5D             fix_pot();
7C0         }
884         return {flow, cost};
AA4     }
2C0     void fix_pot() {
19F         for(int u = 0; u < n; ++u) {
35E             if(dist[u] < INF) {
AB7                 pot[u] += dist[u];
BC9             }
AC5         }
011     }
18C };
D41 // hash: 8615758555
a5fbae52f7e33dad88b6571dcf9bbb7841fb78589debed2a13d424

```

7.9 dsu

```

D56 struct DSU{
1A8     int n;
2E3     vector<int> p, sz;
2EB     DSU(int n) : n(n) {
1C4         p.resize(n);
529         sz.resize(n, 1);
469         iota(begin(p), end(p), 0);
701     }
686     int size(int a){ return sz[root(a)]; }
E22     int root(int a){ return p[a] = (p[a] == a ? a : root(p[a]
)); }
765     bool unite(int a, int b){
6CE         int ra = root(a), rb = root(b);
5C1         if(ra == rb) return 0;
F13         if(sz[ra] < sz[rb]) swap(ra, rb);
484         p[rb] = ra;
9FA         sz[ra] += sz[rb];

```

```

6A5     return 1;
DC9 }
CA7 };

```

7.10 dijkstra

```

1BC const int mx = 1e5+5;
E55 using pii = pair<ll, int>;
93C vector<pii> g[mx];
57C const ll inf = 8e18;
FF9 ll dist[mx]; // setar tudo inf

38F void dijkstra(ll src){
E13     dist[src] = 0;
F37     priority_queue<pii, vector<pii>, greater<pii>> pq;
E8D     pq.push({0, src});
502     while(!pq.empty()){
2F9         auto [d, u] = pq.top();
716         pq.pop();
3E1         if(d > dist[u]) continue;
78E         for(auto [w, v] : g[u]){
C48             ll cur = dist[u] + w;
F65             if(cur < dist[v]){
D84                 dist[v] = cur;
E9A                 pq.push({cur, v});
AAA             }
E5F         }
E9F     }
67C }

```

7.11 twosat

usar ~ para negacao
regras logica
 $A \rightarrow B = \sim B \rightarrow \sim A$ (contrapositiva)
 $A \rightarrow B = \sim A \vee B$ (lei da implicacao)
 $\sim(A \vee B) = \sim A \wedge \sim B$ (de morgan)
 $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$ (distributiva)

```
8B2 #define PB push_back
```

```

D9D struct TwoSat{
1A8     int n;
99F     vector<vector<int>> G, Gt;
556     vector<int> id, order, ans;
B57     vector<bool> vis;
E53     TwoSat() {}
4B1     TwoSat(int n) : n(n){
FC7         G.resize(2*n);
E60         Gt.resize(2*n);
8B0         id.assign(2*n, -1);
CD7         ans.resize(n);
A89     }
D41     // negativos na esquerda
58B     void add_edge(int u, int v){
8EB         u = (u < 0 ? -1-u : u + n);
E41         v = (v < 0 ? -1-v : v + n);
A23         G[u].PB(v);
03D         Gt[v].PB(u);
276     }
B58     void add_or(int a, int b){
FED         add_edge(~a, b);
65F         add_edge(~b, a);
878     }

```

```

D41 // Apenas algum ser 1
D9B void add_xor(int a, int b){
23B     add_or(a,b);
6A8     add_or(~a,~b);
B18 }
D41 // set(a) = 1, set(~a) = 0
F75 void set(int a) { // (a/a)
679     add_or(a,a);
46C }
D41 // Mesmo valor
AA6 void add_xnor(int a, int b) {
170     add_xor(~a,b);
D41 }
CB7 void dfs1(int u){
B9C     vis[u] = true;
E44     for(int v : G[u])
C2D         if(!vis[v])
3BA             dfs1(v);

26F     order.PB(u);
92F }
6A1 void dfs2(int u, int idx){
846     id[u] = idx;
51D     for(int v : Gt[u])
BBC         if(id[v] == -1)
039         dfs2(v,idx);
594 }
DB8 void kosaraju(){
DF8     vis.assign(2*n,false);
3DF     for(int i = 0; i < 2*n; i++)
F69         if(!vis[i])
6D1             dfs1(i);
3B9     reverse(begin(order),end(order));
503     int idx = 0;
882     for(int u : order){
98B         if(id[u] == -1)
96A             dfs2(u, idx++);
D65     }
656 }
271 bool satisfiable(){
75D     kosaraju();
603     for(int i = 0; i < n; i++){
C8B         if(id[i] == id[i + n]) return false;
3C8         ans[i] = (id[i] < id[i + n]);
1B8     }
8A6     return true;
35D }
3E3 };

```

7.12 lca

```

90A const int mxn = 2e5+5;
853 const int LOG = 22;
3B9 int n, q;
1C7 int tin[mxn], tout[mxn];
5FE vector<vector<int>>> up; // up[v][k] = 2^k-esimo ancestor
    de v
9E5 vector<int> g[mxn];
8E0 int lvl[mxn];
2DC int timer = 0;
FB6 void dfs(int u, int p){
73E     tin[u] = ++timer;
D0F     lvl[u] = lvl[p] + 1;
22B     up[u][0] = p;
C64     for(int i = 1; i <= LOG; i++){
88A         up[u][i] = up[ up[u][i-1] ][i-1];

```

```

378     }
4D5     for(int v : g[u]){
6F3         if(v != u && !tin[v])
95E             dfs(v,u);
D69     }
5EF     tout[u] = ++timer;
557 }

F31 bool is_ancestor(int u, int v){
B6F     return tin[u] <= tin[v] && tout[u] >= tout[v];
88C }

7BE int lca(int a, int b){
727     if(is_ancestor(a,b)) return a;
9ED     if(is_ancestor(b,a)) return b;
E70     for(int i = LOG; i >= 0; i--){
AF6         if(!is_ancestor(up[a][i], b)){
3F4             a = up[a][i];
EEF         }
B9D     }
E6F     return up[a][0];
8A8 }

```

7.13 floydWarshall

```

9F9 const int mxn = 505;
3BF const ll inf = 1e18;
3F3 ll g[mxn][mxn]; // setar tudo infinito menos (i,i) como 0
1A8 int n;
3F0 void addEdge(int u, int v, ll w){
E24     g[u][v] = min(g[u][v],w);
9C7     g[v][u] = min(g[v][u],w); // tirar se for 1 dir
CBD }

EDA void floyd(){
E22     for(int k = 0; k < n; k++) // << k
830         for(int i = 0; i < n; i++)
F90             for(int j = 0; j < n; j++)
6A8                 if(g[i][k] + g[k][j] < g[i][j]) // cuida overflow
                    aqui (inf)
FE5                     g[i][j] = g[i][k] + g[k][j];
99E }

```

7.14 bridges

```

14E int n, m;
21F const int mxn = 1e5 + 5;
9E5 vector<int> g[mxn];
B1B int tin[mxn], low[mxn];
C83 vector<pii> bridges;
813 int timer = 1;

FB6 void dfs(int u, int p){
406     tin[u] = timer++;
E99     low[u] = tin[u];
612     int ch = 0;
D76     for(int v : g[u]) if(v != p){
85A         if(tin[v]) // lowlink direta
4EC             low[u] = min(tin[v],low[u]);
4E6         else{
95E             dfs(v,u);
E7F             low[u] = min(low[v],low[u]);
E80             if(tin[u] < low[v]) bridges.push_back({u,v});
6A0         }
CFF     }
2DD }

```

8 Extra

8.1 Hash Function

Call

```

g++ hash.cpp -o hash
./hash < code.cpp

```

to get the hash of the code.

The hash ignores comments and whitespaces.

The hash of a line whit } is the hash of all the code since the { that opens it. (is the hash of that context)

(Optional) To make letters upperCase: for(auto&c:s)if('a'<=c) c^=32;

```

DE3 string getHash(string s){
909     ofstream ip("temp.cpp"); ip << s; ip.close();
EE9     system("g++ -E -P -dD -fpreprocessed ./temp.cpp | tr -d
    '[:space:]' | md5sum > hsh.temp");
CEF     ifstream fo("hsh.temp"); fo >> s; fo.close();
A15     return s.substr(0, 3);
17A }

```

```

E8D int main(){
973     string l, t;
3DA     vector<string> st(10);
C61     while(getline(cin, l)){
54F         t = l;
242         for(auto c : l)
F11             if(c == '(') st.push_back(""); else
2F0                 if(c == '}') t = st.back() + l, st.pop_back();
C33         cout << getHash(t) + " " + l + "\n";
1ED         st.back() += t + "\n";
D1B     }
B65 }

```