SamuellH12 - ICPC Library

8 de setembro de 2025

Conteúdo

1 Data Structures

1.1 BIT2D

```
Complexity: O(Log^2 N)
```

```
3CE const int MAXN = 1e3 + 5;

4BA struct BIT2D {
3C6   int bit[MAXN][MAXN];

710   void update(int X, int Y, int val){
A87     for(int x = X; x < MAXN; x += x&(-x))
9F6       for(int y = Y; y < MAXN; y += y&(-y))
7D9         bit[x][y] += val;
678   }

698   int query(int X, int Y){
A93     int sum = 0;
766     for(int x = X; x > 0; x -= x&(-x))
9A5       for(int y = Y; y > 0; y -= y&(-y))
6F2         sum += bit[x][y];
E66     return sum;
D3C   }

785   void updateArea(int xi, int yi, int xf, int yf, int val); //Same of BIT2DSparse
CD0   int queryArea(int xi, int yi, int xf, int yf); //Same of BIT2DSparse
063 };
```

1.2 BIT2DSparse

```
Sparse Binary Indexed Tree 2D

Recebe o conjunto de pontos que serao usados para fazer os
     updates e as queries e cria uma BIT 2D esparsa que
     independe do "tamanho do grid".

Build: O(N Log N) (N -> Quantidade de Pontos)
Query/Update: O(Log N)
IMPORTANTE! Offline!

BIT2D(pts);   // pts -> vecotor<pii> com todos os pontos em
     que serao feitas queries ou updates
```

```
E40 #define pii pair<ll, ll>
AA8 #define upper(v, x) (upper_bound(begin(v), end(v), x) - begin(v))

4BA struct BIT2D {
D54   vector<ll> ord;
302   vector<vector<ll>> bit, coord;

8A4   BIT2D(vector<pii> pts){
B03     sort(begin(pts), end(pts));

7D3     for(auto [x, y] : pts)
76B       if(ord.empty() || x != ord.back())
580         ord.push_back(x);

261     bit.resize(ord.size() + 1);
3EB     coord.resize(ord.size() + 1);

CC7     sort(begin(pts), end(pts), [&](pii &a, pii &b){ return a.second < b.second; });

7D3     for(auto [x, y] : pts)
837       for(int i=upper(ord, x); i < bit.size(); i += i&-i)
3E1         if(coord[i].empty() || coord[i].back() != y)
739           coord[i].push_back(y);

A22     for(int i=0; i<bit.size(); i++) bit[i].assign(coord[i].size()+1, 0);
461   }

14A   void update(ll X, ll Y, ll v){
784     for(int i = upper(ord, X); i<bit.size(); i += i&-i)
609       for(int j = upper(coord[i], Y); j < bit[i].size(); j += j&-j)
9ED         bit[i][j] += v;
5E0   }

258   ll query(ll X, ll Y){
5FF     ll sum = 0;
2C2     for(int i = upper(ord, X); i > 0; i -= i&-i)
40B       for(int j = upper(coord[i], Y); j > 0; j -= j&-j)
B03         sum += bit[i][j];
E66     return sum;
414   }

867   ll queryArea(ll xi, ll yi, ll xf, ll yf){
ABD     return query(xf, yf) - query(xf, yi-1) - query(xi-1, yf) + query(xi-1, yi-1);
7D1   }

6DB   void updateArea(ll xi, ll yi, ll xf, ll yf, ll val){ // OPTIONAL
C02     update(xi,   yi,    val);  // DOESN'T UPDATE AN AREA !!!
061     update(xf+1, yi,   -val);  // It is like: bit1d.update(l-1, -v), bit1d.update(r, +v)
2ED     update(xi,   yf+1, -val);  // so you can do like bit1d.query(i) to see the value "at" i
2BC     update(xf+1, yf+1,  val);  // in this case, call bit2d.query(X, Y)
A75   }
4F2 };
```

1.3 PrefixSum2D

```
3CE const int MAXN = 1e3 + 5;
B77 int ps [MAXN][MAXN];

131 void calcPS2d(){
```

```
998   for (int i = 1; i < MAXN; i++) ps[0][i] += ps[0][i - 1]; //inicializo a la linha
003   for (int i = 1; i < MAXN; i++) ps[i][0] += ps[i - 1][0]; //inicializo a la coluna

7E4   for (int i = 1; i < MAXN; i++)
582     for (int j = 1; j < MAXN; j++)
0B7       ps[i][j] += ps[i - 1][j] + ps[i][j - 1] - ps[i - 1][j - 1];
577 }
E68 int queryPS2d(int xi, int yi, int xf, int yf){ return ps[xf][yf] - ps[xf][yi-1] - ps[xi-1][yf] + ps[xi-1][yi-1]; }
```

1.4 SegTree

```
CD5 template<typename T> struct SegTree {
130   vector<T> seg;
060   int N;
070   T NEUTRO = 0;
F15   SegTree(int n) : N(n) { seg.assign(4*n, NEUTRO); }
136   SegTree(vector<T> &lista) : N(lista.size()) { seg.assign(4*N); build(1, 0, N-1, lista); }
493   T join(T lv, T rv){ return lv + rv; }

07D   T query(int no, int l, int r, int a, int b){
83C     if(b <  l || r <  a) return NEUTRO;
83F     if(a <= l && r <= b) return seg[no];
A48     int m=(l+r)/2, e=no*2, d=e+1;

703     return join(query(e, l, m, a, b), query(d, m+1, r, a, b));
2F0   }
692   void update(int no, int l, int r, int pos, T v){
085     if(pos < l || r < pos) return;
727     if(l == r){ seg[no] = v; return; }  // set value -> change to += if sum
A48     int m=(l+r)/2, e=no*2, d=e+1;

618     update(e, l,   m, pos, v);
B39     update(d, m+1, r, pos, v);

F93     seg[no] = join(seg[e], seg[d]);
186   }
230   void build(int no, int l, int r, vector<T> &lista){
5FB     if(l == r){ seg[no] = lista[l]; return; }
A48     int m=(l+r)/2, e=no*2, d=e+1;

91F     build(e, l,   m, lista);
415     build(d, m+1, r, lista);

F93     seg[no] = join(seg[e], seg[d]);
F00   }

367   T query(int ls, int rs){ return query (1, 0, N-1, ls, rs); }
345   void update(int pos, T v){      update(1, 0, N-1, pos, v); }
C82 };
```

1.5 SegTree Lazy

```
CD5 template<typename T> struct SegTree {
130   vector<T> seg;
22C   vector<T> lazy;
060   int N;
070   T NEUTRO = 0;
DF1   SegTree(int n) : N(n){ seg.assign(4*N, NEUTRO), lazy.assign(4*N, NEUTRO); }
A94   SegTree(vector<T> &lista) : N(lista.size()){
647     seg.assign(4*N), lazy.assign(4*N, NEUTRO);
575     build(1, 0, N-1, lista);
713   }
493   T join(T lv, T rv){ return lv + rv; }
6B5   void unlazy(int no, int l, int r){
1BB     if(lazy[no] == NEUTRO) return;
A48     int m=(l+r)/2, e=no*2, d=e+1;

5A7     seg[no] += (r-l+1) * lazy[no]; /// Range Sum

1EF     if(l != r) lazy[e] += lazy[no], lazy[d] += lazy[no];
47C     lazy[no] = NEUTRO;
9F0   }

07D   T query(int no, int l, int r, int a, int b){
5C5     unlazy(no, l, r);
83C     if(b <  l || r <  a) return NEUTRO;
83F     if(a <= l && r <= b) return seg[no];
A48     int m=(l+r)/2, e=no*2, d=e+1;

703     return join(query(e, l, m, a, b), query(d, m+1, r, a, b));
E4D   }

DC1   void update(int no, int l, int r, int a, int b, T v){
5C5     unlazy(no, l, r);
2E6     if(b <  l || r <  a) return;
02B     if(a <= l && r <= b){
7CC       lazy[no] = join(lazy[no], v); // cumulative?
8DC       return unlazy(no, l, r);
13F     }
A48     int m=(l+r)/2, e=no*2, d=e+1;

142     update(e, l,   m, a, b, v);
9D3     update(d, m+1, r, a, b, v);

F93     seg[no] = join(seg[e], seg[d]);
B3A   }

230   void build(int no, int l, int r, vector<T> &lista){
5FB     if(l == r){ seg[no] = lista[l]; return; }
A48     int m=(l+r)/2, e=no*2, d=e+1;

91F     build(e, l,   m, lista);
415     build(d, m+1, r, lista);

F93     seg[no] = join(seg[e], seg[d]);
F00   }

367   T query(int ls, int rs){ return query (1, 0, N-1, ls, rs); }
62C   void update(int l, int r, T v){ update(1, 0, N-1, l, r, v); }
2DE };


5C1 -> Segment Tree - Lazy Propagation com:
407   - Query em Range
279   - Update em Range
94E   - Closed interval & 0-indexed: [L, R] & [0, N-1]
B61 Build:   O(N)
E7C Query:   O(log N) | seg.query(l, r);
F5C Update: O(log N) | seg.update(l, r, v);
240 Unlazy: O(1)
84C **Update Join, NEUTRO, Update and Unlazy if needed**
```

1.6 SegTree Persistente

```
-> Segment Tree Persistente: (2x mais rapido que com ponteiro)
Build(1, N) -> Cria uma Seg Tree completa de tamanho N;
      RETORNA o NodeId da Raiz
Update(Root, pos, v) -> Soma +V em POS; RETORNA o NodeId da
      nova Raiz;
Query(Root, a, b) -> RETORNA o valor do range [a, b];
Kth(RootL, RootR, K) -> Faz uma Busca Binaria na Seg de
      diferenca entre as duas versoes.
[ Root -> No Raiz da Versao da Seg na qual se quer realizar a
      operacao ]

Build:  O(N)  !!! Sempre chame o Build
Query:  O(log N)
Update: O(log N)
Kth:  O(Log N)

Comportamento do K-th(SegL, SegR, 1, N, K):
  -> Retorna indice da primeira posicao i cuja soma de
      prefixos [1, i] e >= k na Seg resultante da subtracao dos
      valores da (Seg R) - (Seg L).
  -> Pode ser utilizada para consultar o K-esimo menor valor
      no intervalo [L, R] de um array.
  A Seg deve ser utilizada como um array de frequencias.
      Comece com a Seg zerada (Build).
  Para cada valor V do Array chame um update(roots.back(), 1,
      N, V, 1) e guarde o ponteiro da seg.
  Consultar o K-esimo menor valor de [L, R]: chame
      kth(roots[L-1], roots[R]);
```

```
80E const int MAXN = 1e5 + 5;
2D8 const int MAXLOG = 31 - __builtin_clz(MAXN) + 1;
4B4 typedef int NodeId;
6E2 typedef int STp;

EA9 const STp NEUTRO = 0;
B50 int IDN, LSEG, RSEG;
519 extern struct Node NODES[];

BF2 struct Node {
AEE   STp val;
1BC   NodeId L, R;
9DA   Node(STp v = NEUTRO) : val(v), L(-1), R(-1) {}
2F4   Node& l(){ return NODES[L]; }
F2E   Node& r(){ return NODES[R]; }
5A4 };

318 Node NODES[4*MAXN + MAXLOG*MAXN]; //!!!CUIDADO COM O TAMANHO (aumente se necessario)
1E7 pair<Node&, NodeId> newNode(STp v = NEUTRO){ return {NODES[IDN] = Node(v), IDN++}; }

C3F STp join(STp lv, STp rv){ return lv + rv; }

8B5 NodeId build(int l, int r, bool root=true){
85B   if(root) LSEG = l, RSEG = r;
844   if(l == r) return newNode().second;

EE4   int m = (l+r)/2;
DC6   auto [node, id] = newNode();

C12   node.L = build(l,   m, false);
373   node.R = build(m+1, r, false);
45D   node.val = join(node.l().val, node.r().val);

648   return id;
9D5 }
```

```
2F1 NodeId update(NodeId node, int l, int r, int pos, int v){
703   if( pos < l || r < pos ) return node;
D99   if(l == r) return newNode(NODES[node].val + v).second;

EE4   int m = (l+r)/2;
BE4   auto [nw, id] =newNode();

E2C   nw.L = update(NODES[node].L, l,   m, pos, v);
D4A   nw.R = update(NODES[node].R, m+1, r, pos, v);

6EC   nw.val = join(nw.l().val, nw.r().val);

648   return id;
938 }
8C0 NodeId update(NodeId node, int pos, STp v){ return update(
    node, LSEG, RSEG, pos, v); }

BFA int query(Node& node, int l, int r, int a, int b){
83C   if(b <  l || r <  a) return NEUTRO;
65A   if(a <= l && r <= b) return node.val;

EE4   int m = (l+r)/2;

083   return join(query(node.l(), l, m, a, b), query(node.r(),
    m+1, r, a, b));
7B5 }
8B3 int query(NodeId node, int a, int b){ return query(NODES[
    node], LSEG, RSEG, a, b); }

D0A int kth(Node& Left, Node& Right, int l, int r, int k){
3CE   if(l == r) return l;

A3B   int sum =Right.l().val - Left.l().val;
EE4   int m = (l+r)/2;

BBB   if(sum >= k) return kth(Left.l(), Right.l(), l, m, k);
5D8   return kth(Left.r(), Right.r(), m+1, r, k - sum);
9D7 }
A8D int kth(NodeId Left, NodeId Right, int k){ return kth(
    NODES[Left], NODES[Right], LSEG, RSEG, k); }
```

1.7 SegTree Iterativa

```
CD5 template<typename T> struct SegTree {
1A8   int n;
130   vector<T> seg;
F93   T join(T&l, T&r){ return l + r; }

5A8   SegTree(int n) : n(n), seg(2*n) {}
BD8   SegTree(){}
D5D   void init(vector<T>&base){
FC7     n = base.size();
A61     seg.resize(2*n);
8DB     for(int i=0; i<n; i++) seg[i+n] = base[i];
2E1     for(int i=n-1; i>0; i--) seg[i] = join(seg[i*2], seg[i
    *2+1]);
D60   }

B7A   T query(int l, int r){ //[L, R] // [0, n-1]
7DE     T lp = 0, rp = 0; //NEUTRO
706     for(l+=n, r+=n+1; l<r; l/=2, r/=2){
8C0       if(l&1) lp = join(lp, seg[l++]);
A01       if(r&1) rp = join(seg[--r], rp);
FE5     }
757     return join(lp, rp);
7E8   }

FB2   void update(int i, T v){ // Set Value seg[i+=n] = v //
    change to += v to sum
CBC     for(seg[i+=n] = v; i/=2;) seg[i] = join(seg[i*2], seg[
    i*2+1]);
5E8   }
406 };
```

1.8 SegTree Lazy Iterativa

```
CD5 template<typename T> struct SegTree {
D16   int n, h;
070   T NEUTRO = 0;
97F   vector<T> seg, lzy;
1DF   vector<int> sz;
F93   T join(T&l, T&r){ return l + r; }

5C7   void init(int _n){
8FD     n = _n;
704     h = 32 - __builtin_clz(n);
A61     seg.resize(2*n);
A88     lzy.assign(n, NEUTRO);
528     sz.resize(2*n, 1);
E3F     for(int i=n-1; i; i--) sz[i] = sz[i*2] + sz[i*2+1];
D41     // for(int i=0; i<n; i++) seg[i+n] = base[i];
D41     // for(int i=n-1; i; i--) seg[i] = join(seg[i*2], seg[
    i*2+1]);
95C   }

45B   void apply(int p, T v){
13A     seg[p] += v * sz[p]; // cumulative?
9F8     if(p < n) lzy[p] += v;
853   }
3B4   void push(int p){
835     for(int s=h, i=p>>s; s; s--, i=p>>s)
E15       if(lzy[i] != 0) {
561         apply(i*2,   lzy[i]);
1AD         apply(i*2+1, lzy[i]);
16B         lzy[i] = NEUTRO; //NEUTRO
227       }
3C7   }
F6E   void build(int p) {
5B2     for(p/=2; p; p/= 2){
F12       seg[p] = join(seg[p*2], seg[p*2+1]);
C3C       if(lzy[p] != 0) seg[p] += lzy[p] * sz[p];
D65     }
972   }

B7A   T query(int l, int r){ //[L, R] & [0, n-1]
0ED     l+=n, r+=n+1;
F4B     push(l); push(r-1);

821     T lp = NEUTRO, rp = NEUTRO; //NEUTRO
DC6     for(; l<r; l/=2, r/=2){
8C0       if(l&1) lp = join(lp, seg[l++]);
A01       if(r&1) rp = join(seg[--r], rp);
833     }
BA7     return ans;
F57   }

FAB   void update(int l, int r, T v){
0ED     l+=n, r+=n+1;
F4B     push(l); push(r-1);

98D     int l0 = l, r0 = r;
DC6     for(; l<r; l/=2, r/=2){
5D1       if(l&1) apply(l++, v);
E94       if(r&1) apply(--r, v);
55B     }
FE7     build(l0); build(r0-1);
E29   }
AEB };
```

1.9 SparseTable

```
875 template<typename T> struct Sparse {
F9A   vector<vector<T>> table;

985   void build(vector<T> &v){
128     int N = v.size(), MLOG = 32 - __builtin_clz(N);
554     table.assign(MLOG, v);

DAD     for(int p=1; p < MLOG; p++)
13B       for(int i=0; i + (1 << p) <= N; i++)
67C         table[p][i] = min(table[p-1][i], table[p-1][i
    +(1<<(p-1))]);
215   }

B7A   T query(int l, int r){
796     int p = 31 - __builtin_clz(r - l + 1);  //floor log
E56     return min(table[p][l], table[p][ r - (1<<p)+1 ]);
3C2   }
B78 };

5EA Sparse Table for Range Minimum Query [L, R] [0, N-1]
DA9 build:  O(N log N)
0EB Query:  O(1)
EEC build(v) -> v = Original Array
331 if you want a static array, do this: for(int i=0; i<N; i
    ++) table[0][i] = v[i];
```

1.10 orderedSet

```
30F #include <ext/pb_ds/tree_policy.hpp>
774 #include <ext/pb_ds/assoc_container.hpp>
0D7 using namespace __gnu_pbds;
7AF template <class T> using ordered_set = tree<T, null_type,
     less<T>, rb_tree_tag, tree_order_statistics_node_update>;
816 template <class K, class V> using ordered_map = tree<K, V,
     less<K>, rb_tree_tag, tree_order_statistics_node_update>;

339 ordered_set<int> os;
1C6 int okey = os.order_of_key(k);  // Number of items
     strictly smaller than K
398 auto kth = os.find_by_order(k); // pointer to K-th element
     in set (0-index)
```

2 dp

2.1 Digit DP

```
Digit DP - Sum of Digits

Solve(K)  ->  Retorna a soma dos digitos de todo numero X tal
     que: 0 <= X <= K
dp[D][S][f]  ->  D: Quantidade de digitos; S: Soma dos
     digitos; f: Flag que indica o limite.
int limite[D] -> Guarda os digitos de K.

Complexity: O(D^2 * B^2)  (B = Base = 10)
```

```
ll dp[2][19][170];

int limite[19];
ll digitDP(int idx, int sum, bool flag){
    if(idx < 0) return sum;
    if(~dp[flag][idx][sum]) return dp[flag][idx][sum];

    dp[flag][idx][sum] = 0;
  int lm = flag ? limite[idx] : 9;

    for(int i=0; i<=lm; i++)
        dp[flag][idx][sum] += digitDP(idx-1, sum+i, (flag && i
            == lm));

    return dp[flag][idx][sum];
}

ll solve(ll k){
    memset(dp, -1, sizeof dp);

  int sz=0;
  while(k){
    limite[sz++] = k % 10LL;
    k /= 10LL;
  }

  return digitDP(sz-1, 0, true);
}
```

2.2 LCS

```
LCS - Longest Common Subsequence

Complexity: O(N^2)

* Recursive: memset(memo, -1, sizeof memo); LCS(0, 0);
* Iterative: LCS_It();

* RecoverLCS O(N)
  Recover just one of all the possible LCS
```

```
const int MAXN = 5*1e3 + 5;
int memo[MAXN][MAXN];

string s, t;

inline int LCS(int i, int j){
  if(i == s.size() || j == t.size()) return 0;
  if(memo[i][j] != -1) return memo[i][j];

  if(s[i] == t[j]) return memo[i][j] = 1 + LCS(i+1, j+1);

  return memo[i][j] = max(LCS(i+1, j), LCS(i, j+1));
}

int LCS_It(){
  for(int i=s.size()-1; i>=0; i--)
    for(int j=t.size()-1; j>=0; j--)
      if(s[i] == t[j])
        memo[i][j] = 1 + memo[i+1][j+1];
      else
        memo[i][j] = max( memo[i+1][j], memo[i][j+1] );

  return memo[0][0];
}

string RecoverLCS(int i, int j){
  if(i == s.size() || j == t.size()) return "";

  if(s[i] == t[j]) return s[i] + RecoverLCS(i+1, j+1);

  if(memo[i+1][j] > memo[i][j+1]) return RecoverLCS(i+1, j);

  return RecoverLCS(i, j+1);
}
```

2.3 LIS

```
LIS - Longest Increasing Subsequence

Complexity: O(N Log N)
* For ICREASING sequence, use lower_bound()
* For NON DECREASING sequence, use upper_bound()
```

```
int LIS(vector<int>& nums){
  vector<int> lis;

  for(auto x : nums)
  {
    auto it = lower_bound(lis.begin(), lis.end(), x);

    if(it == lis.end()) lis.push_back(x);
    else *it = x;
  }

  return (int) lis.size();
}
```

2.4 SOS DP

```
SOS DP - Sum over Subsets

Dado que cada mask possui um valor inicial (iVal), computa
para cada mask a soma dos valores de todas as suas submasks.

N -> Numero Maximo de Bits
iVal[mask] -> initial Value / Valor Inicial da Mask
dp[mask]   -> Soma de todos os SubSets

Iterar por todas as submasks: for(int sub=mask; sub>0;
     sub=(sub-1)&mask)
```

```
F17 const int N = 20;
0A7 ll dp[1<<N], iVal[1<<N];

B70 void sosDP(){ // O(N * 2^N)
8CC     for(int i=0; i<(1<<N); i++)
0B3         dp[i] = iVal[i];

972   for(int i=0; i<N; i++)
D57     for(int mask=0; mask<(1<<N); mask++)
281       if(mask&(1<<i))
E0E         dp[mask] += dp[mask^(1<<i)];
E5B }

7E1 void sosDPsub(){ // O(3^N) //suboptimal
EA1   for (int mask = 0, i; mask < (1<<N); mask++)
CC7     for(i = mask, dp[mask] = iVal[0]; i>0; i=(i-1) & mask)
    //iterate over all submasks
85B       dp[mask] += iVal[i];
986 }
```

3 Grafos

3.1 2-SAT

```
2 SAT - Two Satisfiability Problem
Retorna uma valoracao verdadeira se possivel ou um vetor
     vazio se impossivel;
inverso de u = ~u
```

A	B	OR	AND	NOR	NAND	XOR	XNOR	IMPLY
0	0	0	0	1	1	0	1	1
0	1	1	0	0	1	1	0	1
1	0	1	0	0	1	1	0	0
1	1	1	1	0	0	0	1	1

```
D9D struct TwoSat {
060   int N;
67E   vector<vector<int>> E;

662   TwoSat(int N) : N(N), E(2 * N) {}
3E1   inline int eval(int u) const{ return u < 0 ? ((~u)+N)
     %(2*N) : u; }

B0E   void add_or(int u, int v){
245     E[eval(~u)].push_back(eval(v));
F37     E[eval(~v)].push_back(eval(u));
30A     }
4B9   void add_nand(int u, int v) {
9FA     E[eval(u)].push_back(eval(~v));
CED     E[eval(v)].push_back(eval(~u));
D1C   }
CEB   void set_true (int u){ E[eval(~u)].push_back(eval(u)); }
5A5   void set_false(int u){ set_true(~u); }
286   void add_imply(int u, int v){ E[eval(u)].push_back(eval(
     v)); }
E81   void add_and  (int u, int v){ set_true(u);  set_true(v);
       }
347   void add_nor  (int u, int v){ add_and(~u, ~v); }
A32   void add_xor  (int u, int v){ add_or(u, v); add_nand(u,
     v); }
F65   void add_xnor (int u, int v){ add_xor(u, ~v); }

28E   vector<bool> solve() {
F18     vector<bool> ans(N);
F40     auto scc = tarjan();

51F     for (int u = 0; u < N; u++)
FC2       if(scc[u] == scc[u+N]) return {}; //false
951       else ans[u] = scc[u+N] > scc[u];

BA7     return ans; //true
166   }
BF2 private:
401   vector<int> tarjan() {
C23     vector<int> low(2*N), pre(2*N, -1), scc(2*N, -1);
7B4     stack<int> st;
226     int clk = 0, ncomps = 0;
```

```
3C1     auto dfs = [&](auto&& dfs, int u) -> void {
FD2       pre[u] = low[u] = clk++;
4A6       st.push(u);

7F2       for(auto v : E[u])
325         if(pre[v] == -1) dfs(dfs, v), low[u] = min(low[u], low[v]);
295         else
16E         if(scc[v] == -1) low[u] = min(low[u], pre[v]);

8AD       if(low[u] == pre[u]){
78B         int v = -1;
696         while(v != u) scc[v = st.top()] = ncomps, st.pop();
9DF         ncomps++;
CBB       }
860     };

438     for(int u=0; u < 2*N; u++)
DC6       if(pre[u] == -1)
22C         dfs(dfs, u);

9AB     return scc; //tarjan SCCs order is the reverse of topoSort, so (u->v if scc[v] <= scc[u])
98F   }
DC3 };
```

3.2 BlockCutTree

```
Block Cut Tree - BiConnected Component
BlockCutTree bcc(n);
bcc.addEdge(u, v);
bcc.build();

bcc.tree    -> graph of BlockCutTree (tree.size() <= 2n)
bcc.id[u]   -> componet of u in the tree
bcc.cut[u]  -> 1 if u is a cut vertex; 0 otherwise
bcc.comp[i] -> vertex of comp i (cut are part of multiple
     comp)
```

```
142 struct BlockCutTree {
0AD     vector<vector<int>> g, tree, comp;
657     vector<int> id, cut;
40B     BlockCutTree(int n) : n(n), g(n), cut(n) {}

FAE     void addEdge(int u, int v){
7EA         g[u].emplace_back(v);
4A3         g[v].emplace_back(u);
1DB     }

0A8     void build(){
9AB         pre = low = id = vector<int>(n, -1);
D0A         for(int u=0; u<n; u++, chd=0) if(pre[u] == -1) // if graph is disconected
86E             tarjan(u, -1), makeComp(-1);            // find cut vertex and make components

35C         for(int u=0; u<n; u++) if(cut[u]) comp.emplace_back(1, u); //create cut components
584         for(int i=0; i<comp.size(); i++)
                                //mark id of each node
679             for(auto u : comp[i]) id[u] = i;

6A6         tree.resize(comp.size());
584         for(int i=0; i<comp.size(); i++)
5AE             for(auto u : comp[i]) if(id[u] != i)
30E                 tree[i].push_back(id[u]),
D8D                 tree[id[u]].push_back(i);
1D5     }
BF2 private:
5D4     vector<int> pre, low;
EA9     vector<pair<int, int>> st;
226     int n, clk = 0, chd=0, ct, a, b;

20D     void makeComp(int u){
DAB         comp.emplace_back();
016         do {
986             tie(a, b) = st.back();
D73             st.pop_back();
71A             comp.back().push_back(b);
203         } while(a != u);
7C1         if(~u) comp.back().push_back(u);
5CF     }

701     void tarjan(int u, int p){
FD2         pre[u] = low[u] = clk++;
5C6         st.emplace_back(p, u);

DD3         for(auto v : g[u]) if(v != p){
EE1             if(pre[v] == -1){
3D2                 tarjan(v, u);
AB6                 low[u]  =  min(low[u], low[v]);
30C                 cut[u] |= ct = (~p && low[v] >= pre[u]) || (p==-1 && ++chd >= 2);
10E                 if(ct) makeComp(u);
995             }
553             else low[u] = min(low[u], pre[v]);
AC4         }
0D9     }
D8F };
```

3.3 Centroid Decomposition

Centroid Decomposition

Complexity: O(N*LogN)

```
dfsc() -> para criar a centroid tree

rem[u]    -> True se U ja foi removido (pra dfsc)
szt[u]    -> Size da subarvore de U (pra dfsc)
parent[u] -> Pai de U na centroid tree *parent[ROOT] = -1
distToAncestor[u][i] -> Distancia na arvore original de u para
seu i-esimo pai na centroid tree *distToAncestor[u][0] = 0

dfsc(u=node, p=parent(subtree), f=parent(centroid tree),
     sz=size of tree)
```

```
229 const int MAXN = 1e6 + 5;

A34 vector<int> grafo[MAXN];
BE9 deque<int> distToAncestor[MAXN];

C76 bool rem[MAXN];
BBD int szt[MAXN], parent[MAXN];

1B0 void getDist(int u, int p, int d=0){
F3E   for(auto v : grafo[u])
A6B     if(v != p && !rem[v])
334       getDist(v, u, d+1);
F0D   distToAncestor[u].emplace_front(d);
C46 }

3A5 int getSz(int u, int p){
030   szt[u] = 1;
F3E   for(auto v : grafo[u])
A6B     if(v != p && !rem[v])
35F       szt[u] += getSz(v, u);
865   return szt[u];
FD9 }

994 void dfsc(int u=0, int p=-1, int f=-1, int sz=-1){
C0F   if(sz < 0) sz = getSz(u, -1); //starting new tree

F3E   for(auto v : grafo[u])
E5C     if(v != p && !rem[v] && szt[v]*2 >= sz)
6F7       return dfsc(v, u, f, sz);

2EA   rem[u] = true, parent[u] = f;
C5E   getDist(u, -1, 0); //get subtree dists to centroid

F3E   for(auto v : grafo[u])
D8A     if(!rem[v])
D8F       dfsc(v, u, u, -1);
B0F }
```

3.4 Dijkstra

```
const int MAXN = 1e6 + 5;
#define INF 0x3f3f3f3f
#define vi vector<int>

vector<pii> grafo [MAXN];

vi dijkstra(int s){
  vi dist (MAXN, INF); // !!! Change MAXN to N

  priority_queue<pii, vector<pii>, greater<pii>> fila;
  fila.push({0, s});
  dist[s] = 0;

  while(!fila.empty())
  {
    auto [d, u] = fila.top();
    fila.pop();

    if(d > dist[u]) continue;

    for(auto [v, c] : grafo[u])
      if( dist[v] > dist[u] + c )
      {
        dist[v] = dist[u] + c;
        fila.push({dist[v], v});
      }
  }

  return dist;
}
Dijkstra - Shortest Paths from Source

caminho minimo de um vertice u para todos os
outros vertices de um grafo ponderado

Complexity: O(N Log N)
```

```
dijkstra(s)       ->  s : Source, Origem. As distancias serao
    calculadas com base no vertice s
grafo[u] = {v, c};    ->  u : Vertice inicial, v : Vertice
    final, c : Custo da aresta
priority_queue<pii, vector<pii>, greater<pii>> ->  Ordena pelo
     menor custo -> {d, v} -> d : Distancia, v : Vertice
```

3.5 Dinic

```
    Dinic - Max Flow Min Cut
Algoritmo de Dinitz para encontrar o Fluxo Maximo.
Casos de Uso em [Theorems/Flow]
IMPORTANTE! O algoritmo esta 0-indexado

Complexity:
O( V^2 * E )      ->  caso geral
O( sqrt(V) * E )  ->  grafos com cap = 1 para toda Edge //
    matching bipartido

* Informacoes:
  Crie o Dinic: Dinic dinic(n, src, sink);
  Adicione as edges: dinic.addEdge(u, v, capacity);
  Para calcular o Fluxo Maximo: dinic.maxFlow()
  Para saber se um vertice U esta no Corte Minimo:
     dinic.inCut(u)

* Sobre o Codigo:
  vector<Edge> edges;  -> Guarda todas as edges do grafo e do
     grafo residual
  vector<vector<int>> adj; -> Guarda em adj[u] os indices de
     todas as edges saindo de u
  vector<int> ptr;   ->  Pointer para a proxima Edge ainda
     nao visitada de cada vertice
  vector<int> lvl; -> Distancia em vertices a partir do
     Source. Se igual a N o vertice nao foi visitado.
  A BFS retorna se Sink e alcancavel de Source. Se nao e
     porque foi atingido o Fluxo Maximo
  A DFS retorna um possivel aumento do Fluxo
```

```
E9B struct Edge {
37D   int u, v; ll cap;
525   Edge(int u, int v, ll cap) : u(u), v(v), cap(cap) {}
15B };

14D struct Dinic {

B82   int n, src, sink;
903   vector<vector<int>> adj;
321   vector<Edge> edges;
B4A   vector<int> lvl, ptr; //pointer para a proxima Edge nao
    saturada de cada vertice

4A1   Dinic(int n, int src, int sink) : n(n), src(src), sink(
    sink) { adj.resize(n); }

078   void addEdge(int u, int v, ll cap)
F95   {
471     adj[u].push_back(edges.size());
497     edges.emplace_back(u, v, cap);

282     adj[v].push_back(edges.size());
659     edges.emplace_back(v, u, 0);
1F3   }

AD2   ll dfs(int u, ll flow = 1e9){
87D     if(flow == 0) return 0;
B2A     if(u == sink) return flow;

AD2     for(int &i = ptr[u];  i < adj[u].size();  i++)
F95     {
023       int at = adj[u][i];
C99       int v = edges[at].v;

6A0       if(lvl[u] + 1 != lvl[v]) continue;

4A1       if(ll got = dfs(v, min(flow, edges[at].cap)) )
F95       {
6FA         edges[at].cap -= got;
E39         edges[at^1].cap += got;
529         return got;
357       }
656     }

BB3     return 0;
95A   }

838   bool bfs(){
26B     lvl = vector<int> (n, n);
91E     lvl[src] = 0;

26A     queue<int> q;
8A7     q.push(src);

EE6     while(!q.empty())
F95     {
E4A       int u = q.front();
833       q.pop();

E20       for(auto i : adj[u]){
628         int v = edges[i].v;

1B2         if(edges[i].cap == 0 || lvl[v] <= lvl[u] + 1 )
    continue;

97B         lvl[v] = lvl[u] + 1;
2A1         q.push(v);
714       }
6D8     }

710     return lvl[sink] < n;
752   }

D6E   bool inCut(int u){ return lvl[u] < n; }

FE4   ll maxFlow(){
04B     ll ans = 0;

6D4     while( bfs() ){
11B       ptr = vector<int> (n+1, 0);

CF2       while(ll got = dfs(src)) ans += got;
815     }

BA7     return ans;
E9E   }
36C };
```

3.6 DSU Rollback

```
Disjoint Set Union with Rollback - O(Log n)
checkpoint() -> salva o estado atual
rollback() -> restaura no ultimo checkpoint
save another var? +save in join & +line in pop
```

```
4EA struct DSUr {
ECD   vector<int> pai, sz, savept;
D35   stack<pair<int&, int>> st;
EB0   DSUr(int n) : pai(n+1), sz(n+1, 1) {
51E     for(int i=0; i<=n; i++) pai[i] = i;
6CE   }

43F   int find(int u){ return pai[u] == u ? u : find(pai[u]);
    }

AF9   void join(int u, int v){
B80     u = find(u), v = find(v);

360     if(u == v) return;
844     if(sz[v] > sz[u]) swap(u, v);

A60     save(pai[v]); pai[v] = u;
5DA     save(sz[u]);  sz[u] += sz[v];
047   }

2D0   void save(int &x){ st.emplace(x, x); }
42D   void pop(){
6A1     st.top().first = st.top().second; st.pop();
6A1     st.top().first = st.top().second; st.pop();
4DD   }

6E6   void checkpoint(){ savept.push_back(st.size()); }
5CF   void rollback(){
8EB     while(st.size() > savept.back()) pop();
520     savept.pop_back();
BB2   }
9E2 };
```

3.7 DSU Persistente

```
SemiPersistent Disjoint Set Union - O(Log n)
find(u, q) -> Retorna o pai de U no tempo q
* tim -> tempo em que o pai de U foi alterado
```

```
2CE struct DSUp {
AE4   vector<int> pai, sz, tim;
258   int t=1;
910   DSUp(int n) : pai(n+1), sz(n+1, 1), tim(n+1) {
51E     for(int i=0; i<=n; i++) pai[i] = i;
50F   }

7F9   int find(int u, int q = INT_MAX){
568     if( pai[u] == u || q < tim[u] ) return u;
8B3     return find(pai[u], q);
0A1   }

AF9   void join(int u, int v){
B80     u = find(u), v = find(v);

360     if(u == v) return;
844     if(sz[v] > sz[u]) swap(u, v);

555     pai[v] = u;
36E     tim[v] = t++;
CC3     sz[u] += sz[v];
8D8   }
96D };
```

3.8 Euler Path

Euler Path - Algoritmo de Hierholzer para caminho Euleriano

Complexity: O(V + E)

IMPORTANTE! O algoritmo esta 0-indexado

* Informacoes
  addEdge(u, v) -> Adiciona uma aresta de U para V
  EulerPath(n) -> Retorna o Euler Path, ou um vetor vazio se impossivel
  vi path -> vertices do Euler Path na ordem
  vi pathId -> id das Arestas do Euler Path na ordem

Euler em Undirected graph:
  - Cada vertice tem um numero par de arestas (circuito); OU
  - Exatamente dois vertices tem um numero impar de arestas (caminho);

Euler em Directed graph:
  - Cada vertice tem quantidade de arestas |entrada| == |saida| (circuito); OU
  - Exatamente 1 tem |entrada|+1 == |saida| && exatamente 1 tem |entrada| == |saida|+1 (caminho);

* Circuito -> U e o primeiro e ultimo
* Caminho -> U e o primeiro e V o ultimo

```
0C1 #define vi vector<int>

229 const int MAXN = 1e6 + 5;
210 const bool BIDIRECIONAL = true;

7F4 vector<pii> grafo[MAXN];
CBD vector<bool> used;

FAE void addEdge(int u, int v){
FD8   grafo[u].emplace_back(v, used.size()); if(BIDIRECIONAL && u != v)
7F0   grafo[v].emplace_back(u, used.size());
EDA   used.emplace_back(false);
3C1 }

EFB pair<vi, vi> EulerPath(int n, int src=0){
79C   int s=-1, t=-1;
E4D   vector<int> selfLoop(n*BIDIRECIONAL, 0);

C30   if(BIDIRECIONAL)
F95   {
C0F     for(int u=0; u<n; u++) for(auto&[v, id] : grafo[u]) if(u==v) selfLoop[u]++;
19E     for(int u=0; u<n; u++)
D2B       if((grafo[u].size() - selfLoop[u])%2)
A4F         if(t != -1) return {vi(), vi()};   // mais que 2 com grau impar
F8A         else t = s, s = u;

C0E     if(t == -1 && t != s) return {vi(), vi()}; // so 1 com grau impar
E78     if(s == -1 || t == src) s = src;            // se possivel, seta start como src
E07   }
295   else
F95   {
8E2     vector<int> in(n, 0), out(n, 0);

19E     for(int u=0; u<n; u++)
0DB       for(auto [v, edg] : grafo[u])
8C0         in[v]++, out[u]++;

19E     for(int u=0; u<n; u++)
074       if(in[u] - out[u] == -1 && s == -1) s = u;  else
3C0       if(in[u] - out[u] ==  1 && t == -1) t = u;  else
825       if(in[u] !=out[u]) return {vi(), vi()};

755     if(s == -1 && t == -1) s = t = src;          // se possivel, seta s como src
A6E     if(s == -1 && t != -1) return {vi(), vi()}; // Existe S mas nao T
1E2     if(s != -1 && t == -1) return {vi(), vi()}; // Existe T mas nao S
667   }

84C   for(int i=0; grafo[s].empty() && i<n; i++) s =(s+1)%n; //evita s ser vertice isolado

D41   /////// DFS ///////
66A   vector<int> path, pathId, idx(n, 0);
982   stack<pii> st; // {Vertex, EdgeId}
D1E   st.push({s, -1});

2C8   while(!st.empty())
F95   {
723     auto [u, edg] = st.top();
B38     while(idx[u] < grafo[u].size() && used[grafo[u][idx[u]].second]) idx[u]++;

704     if(idx[u] < grafo[u].size())
F95     {
CAD       auto [v, id] = grafo[u][idx[u]];
3C1       used[id] = true;
F26       st.push({v, id});
5E2       continue;
2A1     }

960     path.push_back(u);
E1A     pathId.push_back(edg);
25A     st.pop();
5E9   }

301   pathId.pop_back();
023   reverse(begin(path),   end(path));
6FF   reverse(begin(pathId), end(pathId));

D41   ///  Grafo conexo ? ///
ADC   int edgesTotal = 0;
4B4   for(int u=0; u<n; u++) edgesTotal += grafo[u].size() + (BIDIRECIONAL ? selfLoop[u] : 0);
0A8   if(BIDIRECIONAL) edgesTotal /= 2;
934   if(pathId.size() != edgesTotal) return {vi(), vi()};

438   return {path, pathId};
722 }
```

3.9 HLD

Heavy-Light Decomposition

Complexity: O(LogN * (qry || updt))

Change qry(l, r) and updt(l, r) to call a query and update structure of your will

HLD hld(n); //call init
hld.add_edges(u, v); //add all edges
hld.build(); //Build everthing for HLD

tin[u] -> Pos in the structure (Seg, Bit, ...)
nxt[u] -> Head/Endpoint
IMPORTANTE! o grafo deve estar 0-indexado!

```
EAA const bool EDGE = false;
403 struct HLD {
673 public:
789   vector<vector<int>> g; //grafo
575   vector<int> sz, parent, tin, nxt;
1B1   HLD(){}
90C   HLD(int n){ init(n); }
940   void init(int n){
A34     t = 0;
8F5     g.resize(n); tin.resize(n);
7BA     sz.resize(n);nxt.resize(n);
62B     parent.resize(n);
D94   }
FAE   void addEdge(int u, int v){
7EA     g[u].emplace_back(v);
4A3     g[v].emplace_back(u);
1DB   }
1F8   void build(int root=0){
E4A     nxt[root]=root;
043     dfs(root, root);
7D9     hld(root, root);
F40   }

3D1   ll query_path(int u, int v){
0E8     if(tin[u] < tin[v]) swap(u, v);
D63     if(nxt[u] == nxt[v]) return qry(tin[v]+EDGE, tin[u]);
7C8     return qry(tin[nxt[u]], tin[u]) + query_path(parent[nxt[u]], v);
C6B   }

2F3   void update_path(int u, int v, ll x){
0E8     if(tin[u] < tin[v]) swap(u, v);
D55     if(nxt[u] == nxt[v]) return updt(tin[v]+EDGE, tin[u], x);
0A7     updt(tin[nxt[u]], tin[u], x); update_path(parent[nxt[u]], v, x);
177   }

BF2 private:
EBB   ll qry(int l, int r){ if(EDGE && l>r) return 0;/*NEUTRO*/ } //call Seg, BIT, etc
6D9   void updt(int l, int r, ll x){ if(EDGE && l>r) return; } //call Seg, BIT, etc

FB6   void dfs(int u, int p){
573     sz[u] = 1, parent[u] = p;
E69     for(auto &v : g[u]) if(v != p) {
1FB       dfs(v, u); sz[u] += sz[v];

14A       if(sz[v] > sz[g[u][0]] || g[u][0] == p)
06F         swap(v, g[u][0]);
7E2     }
53F   }

6BB   int t=0;
11E   void hld(int u, int p){
2C6     tin[u] = t++;
BF0     for(auto &v : g[u]) if(v != p)
B18       nxt[v] = (v == g[u][0] ? nxt[u] : v),
42C       hld(v, u);
36C   }

D41   /// OPTIONAL ///
310   int lca(int u, int v){
582     while(!inSubtree(nxt[u], v)) u = parent[nxt[u]];
```

```
E1D     while(!inSubtree(nxt[v], u)) v = parent[nxt[v]];
40A     return tin[u] < tin[v] ? u : v;
AEB   }
65E   bool inSubtree(int u, int v){ return tin[u] <= tin[v] &&
      tin[v] < tin[u] + sz[u]; }
D41   //query/update_subtree[tin[u]+EDGE, tin[u]+sz[u]-1];
095   vector<pair<int, int>> pathToAncestor(int u, int a){
F77         vector<pair<int, int>> ans;
7F3         while(nxt[u] != nxt[a])
FCA             ans.emplace_back(tin[nxt[u]], tin[u]),
5C3             u = parent[nxt[u]];
B35         ans.emplace_back(tin[a], tin[u]);
BA7         return ans;
52A     }
BF7 };
```

3.10 LCA

```
  LCA - Lowest Common Ancestor - Binary Lifting
Algoritmo para encontrar o menor ancestral comum
entre dois vertices em uma arvore enraizada

IMPORTANTE! O algoritmo esta 0-indexado

Complexity:
buildBL() ->  O(N Log N)
lca()     ->  O(Log N)

* Informacoes
  -> chame dfs(root, root) para calcular o pai e a altura de
     cada vertice
  -> chame buildBL() para criar a matriz do Binary Lifting
  -> chame lca(u, v) para encontrar o menor ancestral comum
  bl[i][u] -> Binary Lifting com o (2^i)-esimo pai de u
  lvl[u]   -> Altura ou level de U na arvore
```

```
81D const int MAXN = 1e4 + 5;
633 const int MAXLG = 16;

A34 vector<int> grafo[MAXN];

A87 int bl[MAXLG][MAXN], lvl[MAXN];

80E void dfs(int u, int p, int l=0){
34C   lvl[u] = l;
4FB   bl[0][u] = p;

F3E   for(auto v : grafo[u])
F6B     if(v != p)
0C5       dfs(v, u, l+1);
9A8 }

555 void buildBL(int N){
977   for(int i=1; i<MAXLG; i++)
51F     for(int u=0; u<N; u++)
69C       bl[i][u] = bl[i-1][bl[i-1][u]];
59A }

310 int lca(int u, int v){
DC4   if(lvl[u] < lvl[v]) swap(u, v);

D07   for(int i=MAXLG-1; i>=0; i--)
179     if(lvl[u] - (1<<i) >= lvl[v])
319       u = bl[i][u];

60E   if(u == v) return u;

D07   for(int i=MAXLG-1; i>=0; i--)
BFA     if(bl[i][u] != bl[i][v])
E01       u = bl[i][u],
4BC       v = bl[i][v];

68E   return bl[0][u];
381 }
```

3.11 MinCostMaxFlow

```
E9B struct Edge {
F0B   int u, v; ll cap, cost;
DC4   Edge(int u, int v, ll cap, ll cost) : u(u), v(v), cap(
    cap), cost(cost) {}
49B };

6F3 struct MCMF {
878   const ll INF = numeric_limits<ll>::max();
DA6   int n, src, snk;
903   vector<vector<int>> adj;
321   vector<Edge> edges;
39D   vector<ll> dist, pot;
E3B   vector<int> from;

00D   MCMF(int n, int src, int snk) : n(n), src(src), snk(snk)
      { adj.resize(n); pot.resize(n); }

461   void addEdge(int u, int v, ll cap, ll cost){
471     adj[u].push_back(edges.size());
986     edges.emplace_back(u, v, cap, cost);

282     adj[v].push_back(edges.size());
29F     edges.emplace_back(v, u, 0, -cost);
CA1   }

26A   queue<int> q;
B57   vector<bool> vis;
791   bool SPFA(){
EF2     dist.assign(n, INF);
0B5     from.assign(n, -1);
543     vis.assign(n, false);

8A7     q.push(src);
E13     dist[src] = 0;

14D     while(!q.empty()){
E4A       int u = q.front();
833       q.pop();

776       vis[u] = false;

E20       for(auto i : adj[u]){
F42         if(edges[i].cap == 0) continue;
628         int v = edges[i].v;
99A         ll cost = edges[i].cost;

148         if(dist[v] > dist[u] + cost + pot[u] - pot[v]){
DEC           dist[v] = dist[u] + cost + pot[u] - pot[v];
203           from[v] = i;
A1A           if(!vis[v]) q.push(v), vis[v] = true;
888         }
652       }
344     }

19E     for(int u=0; u<n; u++)  //fix pot
067       if(dist[u] < INF)
AB7         pot[u] += dist[u];

071     return dist[snk] < INF;
532   }

B84   pair<ll, ll> augment(){
20B     ll flow = edges[from[snk]].cap, cost = 0; //fixed flow
    : flow = min(flow, remainder)

473     for(int v=snk; v != src; v = edges[from[v]].u)
73D       flow = min(flow, edges[from[v]].cap),
871       cost += edges[from[v]].cost;

473     for(int v=snk; v != src; v = edges[from[v]].u)
86A       edges[from[v]].cap   -= flow,
674       edges[from[v]^1].cap += flow;

884     return {flow, cost};
890   }

164   bool inCut(int u){ return dist[u] < INF; }

6DC   pair<ll, ll> maxFlow(){
D7D     ll flow = 0, cost = 0;

4EB     while( SPFA() ){
274       auto [f, c] = augment();
C87       flow += f;
BFC       cost += f*c;
35C     }
884     return {flow, cost};
D37   }
586 };
```

3.12 SCC - Kosaraju

```
Kosaraju - Strongly Connected Component
Algoritmo de Kosaraju para encontrar Componentes Fortemente
     Conexas

Complexity: O(V + E)
IMPORTANTE! O algoritmo esta 0-indexado

* Variaveis e explicacoes *
int C   -> C e a quantidade de Componetes Conexas. As
     componetes estao numeradas de 0 a C-1
dag     -> Apos rodar o Kosaraju, o grafo das componentes
     conexas sera criado aqui
comp[u] -> Diz a qual componente conexa U faz parte
order   -> Ordem de saida dos vertices. Necessario para o
     Kosaraju
grafo   -> grafo direcionado
greve -> grafo reverso (que deve ser construido junto ao
     grafo normal) !!!

NOTA: A ordem que o Kosaraju descobre as componentes e uma
     Ordenacao Topologica do SCC
em que o dag[0] nao possui grau de entrada e o dag[C-1] nao
     possui grau de saida
```

```
0C1 #define vi vector<int>

229 const int MAXN = 1e6 + 5;

C92 vi grafo[MAXN];
```

```
4ED vi greve[MAXN];
404 vi dag[MAXN];
104 vi comp, order;
B57 vector<bool> vis;
868 int C;

315 void dfs(int u){
B9C   vis[u] = true;
F3E   for(auto v : grafo[u])
C2D     if(!vis[v])
6B4       dfs(v);
C75   order.push_back(u);
8C4 }

163 void dfs2(int u){
361   comp[u] = C;
6A8   for(auto v : greve[u])
750     if(comp[v] == -1)
D5A       dfs2(v);
1F8 }

955 void kosaraju(int n){
070   order.clear();
E28   comp.assign(n, -1);
543   vis.assign(n, false);

84D   for(int v=0; v<n; v++)
C2D     if(!vis[v])
6B4       dfs(v);

796   C = 0;
3B9   reverse(begin(order), end(order));

961   for(auto v : order)
750     if(comp[v] == -1)
400       dfs2(v), C++;

D41   //// Montar DAG  ////
78F   vector<bool> marc(C, false);

687   for(int u=0; u<n; u++){
F3E     for(auto v : grafo[u])
F95     {
264       if(comp[v] == comp[u] || marc[comp[v]]) continue;

812       marc[comp[v]] = true;
F26       dag[comp[u]].emplace_back(comp[v]);
0DC     }

09D     for(auto v : grafo[u]) marc[comp[v]] = false;
A85   }
80A }
```

3.13 Tarjan

```
Tarjan - Pontes e Pontos de Articulacao
Algoritmo para encontrar pontes e pontos de articulacao.

Complexity: O(V + E)
IMPORTANTE! Lembre do memset(pre, -1, sizeof pre);

* Variaveis e explicacoes *
pre[u] = "Altura", ou, x-esimo elemento visitado na DFS.
    Usado para saber a posicao de um vertice na arvore de DFS
low[u] = Low Link de U, ou a menor aresta de retorno (mais
    proxima da raiz) que U alcanca entre seus filhos
chd = Children. Quantidade de componentes filhos de U. Usado
    para saber se a Raiz e Ponto de Articulacao.
any = Marca se alguma aresta de retorno em qualquer dos
    componentes filhos de U nao ultrapassa U. Se isso for
    verdade, U e Ponto de Articulacao.

if(low[v] >  pre[u]) pontes.emplace_back(u, v); -> se a mais
    alta aresta de retorno de V (ou o menor low) estiver
    abaixo de U, entao U-V e ponte
if(low[v] >= pre[u]) any = true;        -> se a mais alta
    aresta de retorno de V (ou o menor low) estiver abaixo de
    U ou igual a U, entao U e Ponto de Articulacao
```

```
229 const int MAXN = 1e6 + 5;
F4C int pre[MAXN], low[MAXN], clk=0;
A34 vector<int> grafo [MAXN];

A2B vector<pair<int, int>> pontes;
252 vector<int> cut;

A76 #warning "lembrar do memset(pre, -1, sizeof pre);"
CF2 void tarjan(int u, int p = -1){
FD2   pre[u] = low[u] = clk++;

B10   bool any = false;
378   int chd = 0;

2BF   for(auto v : grafo[u]){
730     if(v == p) continue;

9BE     if(pre[v] == -1)
F95     {
3D2       tarjan(v, u);

E7F       low[u] = min(low[v], low[u]);

334       if(low[v] >  pre[u]) pontes.emplace_back(u, v);
23A       if(low[v] >= pre[u]) any = true;

87D       chd++;
302     }
295     else
201       low[u] = min(low[u], pre[v]);
6D3   }

B82   if(p == -1 && chd >= 2) cut.push_back(u);
5F3   if(p != -1 && any)      cut.push_back(u);
E3D }
```

4 Strings

4.1 hash2

```
String Hash - Double Hash
precalc()     -> O(N)
StringHash()  -> O(|S|)
gethash()     -> O(1)

StringHash hash(s); -> Cria o Hash da string s
hash.gethash(l, r); -> Hash [L,R] (0-Indexado)
```

```
229 const int MAXN = 1e6 + 5;

E8E const ll MOD1 = 131'807'699;
D5D const ll MOD2 = 1e9 + 9;
145 const ll base = 157;

DB4 ll expb1[MAXN], expb2[MAXN];

921 #warning "Call precalc() before use StringHash"
FE8 void precalc(){
6D8     expb1[0] = expb2[0] = 1;

7E4   for(int i=1;i<MAXN;i++)
E0E         expb1[i] = expb1[i-1]*base % MOD1,
C4B         expb2[i] = expb2[i-1]*base % MOD2;
A02 }

3CE struct StringHash{
0DD     vector<pair<ll,ll>> hsh;
AC0     string s; // comment S if you dont need it

6F2     StringHash(string& s) : s(s){
63F         hsh.assign(s.size()+1, {0,0});

724         for (int i=0;i<s.size();i++)
B7A             hsh[i+1].first  = ( hsh[i].first *base % MOD1
    + s[i] ) % MOD1,
08F             hsh[i+1].second = ( hsh[i].second*base % MOD2
    + s[i] ) % MOD2;
5A6     }

2F0     ll gethash(int a,int b){
F96         ll h1 = (MOD1+ hsh[b+1].first  - hsh[a].first *
    expb1[b-a+1] % MOD1) % MOD1;
F4A         ll h2 = (MOD2+ hsh[b+1].second - hsh[a].second*
    expb2[b-a+1] % MOD2) % MOD2;
D23         return (h1<<32) | h2;
C77     }
1D3 };

0FB int firstDiff(StringHash& a, int la, int ra, StringHash& b
    , int lb, int rb){
7E5   int l=0, r=min(ra-la, rb-lb), diff=r+1;
3D5   while(l <= r){
EE4     int m = (l+r)/2;
065     if(a.gethash(la, la+m) == b.gethash(lb, lb+m)) l = m
    +1;
72D     else r = m-1, diff = m;
BAD   }
2B1   return diff;
C88 }

03D int hshComp(StringHash& a, int la, int ra, StringHash& b,
    int lb, int rb){
E85   int diff = firstDiff(a, la, ra, b, lb, rb);
23E   if(diff > ra-la && ra-la == rb-lb) return 0; //equal
D15   if(diff > ra-la || diff  >  rb-lb) return ra-la < rb-lb
    ? -2 : +2; //prefix of the other
626   return a.s[la+diff] < b.s[lb+diff] ? -1 : +1;
8C4 }
```

4.2 KMP

```
692 vector<int> Pi(string &t){
82B   vector<int> p(t.size(), 0);
```

```
6F4   for(int i=1, j=0; i<t.size(); i++){
90B     while(j > 0  && t[j] != t[i]) j = p[j-1];
3C7     if(t[j] == t[i]) j++;
F8C     p[i] = j;
9E8   }
74E   return p;
85D }

2AD vector<int> kmp(string &s, string &t){
D9E   vector<int> p = Pi(t), occ;

1EF   for(int i=0, j=0; i<s.size(); i++){
705     while( j > 0 && s[i] != t[j]) j = p[j-1];
566     if(s[i]==t[j]) j++;
2F0     if(j == t.size()) occ.push_back(i-j+1), j = p[j-1];
6C4   }
FB0   return occ;
087 }
```

Optional: KMP Automato. j = state atual [root=j=0]

```
3E3 struct Automato {
632   vector<int> p;
78F     string t;
119   Automato(string &t) : t(t), p(Pi(t)){}
6DD     int next(int j, char c){ //return nxt state
E60         if(final(j)) j = p[j-1];
28D         while(j && c != t[j]) j = p[j-1];
5B4         return j + (c == t[j]);
26F     }
DFA     bool final(int j){ return j == t.size(); }
8C2 };

0F8 KMP - Knuth-Morris-Pratt Pattern Searching
05C Complexity: O(|S|+|T|)
DB8 kmp(s, t) -> returns all occurences of t in s
020 p = Pi(t) -> p[i] = biggest prefix that is a sufix of t[0,
      i]
```

4.3 Aho-Corasick

```
Aho-Corasick: Trie automaton to search multiple patterns in a
     text
Complexity: O(SUM|P| + |S|) * ALPHA

for(auto p: patterns) aho.add(p);
aho.buildSufixLink();
auto ans = aho.findPattern(s);

parent(p), sufixLink(sl), outputLink(ol), patternID(idw)
outputLink -> edge to other pattern end (when p is a sufix of
     it)
ALPHA -> Size of the alphabet. If big, consider changing nxt
     to map

To find ALL occurrences of all patterns, don't delete ol in
     findPattern. But it can be slow (at number of occ), so
     consider using DP on the automaton.
If you need a nextState function, create it using the while
     in findPattern.
if you need to store node indexes add int i to Node, and in
     Aho add this and change the new Node() to it:
vector<trie> nodes;
trie new_Node(trie p, char c){
    nodes.push_back(new Node(p, c));
    nodes.back()->i = nodes.size()-1;
    return nodes.back();
}
```

```
322 const int ALPHA = 26, off = 'a';
BF2 struct Node {
E05     Node* p  = NULL;
A26     Node* sl = NULL;
C3A     Node* ol = NULL;
CB8     array<Node*, ALPHA> nxt;

7DE     char c;
BBC     int idw = -1;

212     Node(){ nxt.fill(NULL); }
B04     Node(Node* p, char c) : p(p), c(c) { nxt.fill(NULL); }
92D };
2CA typedef Node* trie;
C99 struct Aho {
ACD     trie root;
EAA     int nwords = 0;
63B     Aho(){ root = new Node(); }

22D     void add(string &s){
346         trie t = root;
242         for(auto c : s){ c -= off;
508             if(!t->nxt[c])
02F                 t->nxt[c] = new Node(t, c);
4F8             t = t->nxt[c];
E9A         }
71E         t->idw = nwords++; //cuidado com strings iguais!
    use vector
625     }

34A     void buildSufixLink(){
A2F         deque<trie> q(1, root);

14D         while(!q.empty()){
81D             trie t = q.front();
CED             q.pop_front();

630             if(trie w = t->p){
29D                 do w = w->sl; while(w && !w->nxt[t->c]);
619                 t->sl = w ? w->nxt[t->c] : root;
D7B                 t->ol = t->sl->idw == -1 ? t->sl->ol : t->
    sl;
8DB             }

806             for(int c=0; c<ALPHA; c++)
F72                 if(t->nxt[c])
78D                     q.push_back(t->nxt[c]);
693         }
09C     }

66F     vector<bool> findPattern(string &s){
BFD         vector<bool> ans(nwords, 0);
82D         trie w = root;
242         for(auto c : s){ c -= off;
A7A             while(w && !w->nxt[c]) w = w->sl;  // trie
    next(w, c)
AEA             w = w ? w->nxt[c] : root;

5BE             for(trie z=w, nl; z; nl=z->ol, z->ol=NULL, z=
    nl)
972                 if(z->idw != -1) //get ALL occ: dont
    delete ol (may slow)
31E                     ans[z->idw] = true;
B04         }
BA7         return ans;
C8E     }
FE8 };
```

4.4 Suffix Array

```
sf = suffixArray(s) -> O(N log N)
LCP(s, sf) -> O(N)

SuffixArray -> index of suffix in lexicographic order
LCP[i] -> LargestCommonPrefix of sufix at sf[i] and sf[i-1]
LCP(i,j) = min(lcp[i+1...j])

To better understand, print: lcp[i] sf[i] s.substr(sf[i])
```

```
B6C vector<int> suffixArray(string s){
274     int n = (s += "$").size(); //if s is vector, push_back
    (-INF);
6B4     vector<int> sf(n), ord(n), aux(n), cnt(n);
CE4     iota(begin(sf), end(sf), 0);
30A     sort(begin(sf), end(sf), [&](int i, int j){ return s[i
    ] < s[j]; });

104     int cur = ord[sf[0]] = 0;
AA4     for(int i=1; i<n; i++)
0BB         ord[sf[i]] = s[sf[i]] == s[sf[i-1]] ? cur : ++cur;

C1E     for(int k=1; cur+1 < n && k < n; k<<=1){
727         cnt.assign(n, 0);
8FF         for(auto &i : sf)         i = (i-k+n)%n, cnt[ord[i
    ]]++;
DC5         for(int i=1; i<n; i++)    cnt[i] += cnt[i-1];
0A4         for(int i=n-1; i>=0; i--) aux[--cnt[ord[sf[i]]]] =
     sf[i];
71C         sf.swap(aux);

662         aux[sf[0]] = cur = 0;
AA4         for(int i=1; i<n; i++)
AEB             aux[sf[i]] = ord[sf[i]] == ord[sf[i-1]] &&
E19             ord[(sf[i]+k)%n] == ord[(sf[i-1]+k)%n] ? cur :
     ++cur;
43A         ord.swap(aux);
52E     }
61D     return vector<int>(begin(sf)+1, end(sf));
1FC }

B1D vector<int> LCP(string &s, vector<int> &sf){
163     int n = s.size();
BF1     vector<int> lcp(n), pof(n);
E51     for(int i=0; i<n; i++) pof[sf[i]] = i;

9A7     for(int i=0, j, k=0; i<n; k?--k:k, i++){
76D         if(!pof[i]) continue;
D5B         j = sf[pof[i]-1];
329         while(i+k<n && j+k<n && s[i+k]==s[j+k]) k++;
F12         lcp[pof[i]] = k;
1D0     }
5ED     return lcp;
EC1 }
```

4.5 trie

```
Trie - Arvore de Prefixos
insert(P) - O(|P|)
count(P)  - O(|P|)
MAXS  - Soma do tamanho de todas as Strings
sigma - Tamanho do alfabeto
```

```
AAF const int MAXS = 1e5 + 10;
70C const int sigma = 26;

F6C int trie[MAXS][sigma], terminal[MAXS], 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';

BCF     if(trie[cur][id] == -1 ){
616       memset(trie[z], -1, sizeof trie[z]);
869       trie[cur][id] = z++;
CAE     }

3AD     cur = trie[cur][id];
A9E   }

B07   terminal[cur]++;
C89 }

684 int count(string &p){
B3D   int cur = 0;

E2E   for(int i=0; i<p.size(); i++){
94B     int id = (p[i] - 'a');

C39     if(trie[cur][id] == -1) return 0;

3AD     cur = trie[cur][id];
ADB   }
89E   return terminal[cur];
D3C }

CA2 void init(){
E6F   memset(trie[0], -1, sizeof trie[0]);
34E   z = 1;
A11 }
```

4.6 Manacher

```
DC6 vector<int> manacher(string &st){
E13   string s = "$_";
821   for(char c : st){ s += c; s += "_"; }
095   s += "#";

995   int n = s.size()-2;

BD7   vector<int> p(n+2, 0);
7CD   int l=1, r=1;

557   for(int i=1, j; i<=n; i++)
F95   {
DAF     p[i] = max(0, min(r-i, p[l+r-i]) ); //atualizo o valor
      atual para o valor do palindromo espelho na string ou
      para o total que esta contido

A5F     while( s[i-p[i]] == s[i+p[i]] ) p[i]++;

39C     if( i+p[i] > r ) l = i-p[i], r = i+p[i];
A83   }

6AE   for(auto &x : p) x--; //o valor de p[i] e igual ao
      tamanho do palindromo + 1
74E   return p;
907 }

BEF Manacher Algorithm
64E Find every palindrome in string
80E Complexidade: O(N)
```

4.7 Z-Function

```
403 vector<int> Zfunction(string &s){ // O(N)
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-l], 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 }
```

5 others

5.1 MO

Algoritmo de MO para query em range

Complexity: O((N + Q) * SQRT(N) * F) | F e a complexidade do Add e Remove

IMPORTANTE! Queries devem ter seus indices (Idx) 0-indexados!

Modifique as operacoes de Add, Remove e GetAnswer de acordo com o problema.
BLOCK_SZ pode ser alterado para aproximadamente SQRT(MAX_N)

```
861 const int BLOCK_SZ = 700;

670 struct Query{
738   int l, r, idx;

991   Query(int l, int r, int idx) : l(l), r(r), idx(idx) {}

406   bool operator < (Query q) const {
6EB     if(l / BLOCK_SZ != q.l / BLOCK_SZ) return l < q.l;
387     return (l / BLOCK_SZ &1) ? ( r < q.r ) : (r > q.r );
667   }
F51 };

543 void add(int idx);
F8A void remove(int idx);
AD7 int getAnswer();

73F vector<int> MO(vector<Query> &queries){
51F   vector<int> ans(queries.size());

BFA   sort(queries.begin(), queries.end());

32D   int L = 0, R = 0;
49E   add(0);
FE9   for(auto [l, r, idx] : queries){
128     while(l < L) add(--L);
C4A     while(r > R) add(++R);
684     while(l > L) remove(L++);
B50     while(r < R) remove(R--);

830     ans[idx] = getAnswer();
08D   }

BA7   return ans;
ACF }

D41 /* IF you want to use hilbert curves on MO
0BD vector<ll> h(ans.size());
CEC for (int i = 0; i < ans.size(); i++) h[i] = hilbert(
     queries[i].l, queries[i].r);
063 sort(queries.begin(), queries.end(), [&](Query&a, Query&b)
     { return h[a.idx] < h[b.idx]; });   */
E51 inline ll hilbert(int x, int y){
C85   static int N = 1 << (__builtin_clz(0) - __builtin_clz(
     MAXN));
B69   int rx, ry, s; ll d = 0;
43B   for(s = N/2; s > 0; s /= 2){
C95     rx = (x & s) > 0, ry = (y & s) > 0;
F15     d += s * (ll)(s) * ((3 * rx) ^ ry);
E2D     if(ry == 0) { if(rx == 1) x = N-1 - x, y = N-1 - y;
     swap(x, y); }
200   }
BE2   return d;
038 }
```

5.2 MOTree

Algoritmo de MO para query de caminho em arvore
Complexity: O((N + Q) * SQRT(N) * F) | F e a complexidade do Add e Remove
IMPORTANTE! 0-indexado!

```
80E const int MAXN = 1e5+5;
F5A const int BLOCK_SZ = 500;
304 struct Query{int l, r, idx;}; //same of MO. Copy operator
     <

282 vector<int> g[MAXN];
212 int tin[MAXN], tout[MAXN];
03B int pai[MAXN], order[MAXN];

179 void remove(int u);
C8B void add(int u);
AD7 int getAnswer();

C0A void go_to(int ti, int tp, int otp){
B21   int u = order[ti], v, to;
61E   to = tout[u];
AA5   while(!(ti <= tp && tp <= to)){ //subo com U (ti) ate
     ser ancestral de W
E7C     v = pai[u];

BAF     if(ti <= otp && otp <= to) add(v);
96E     else remove(u);

A68     u = v;
363     ti = tin[u];
61E     to = tout[u];
```

```
462   }

915   int w = order[tp];
D88   to = tout[w];
082   while(ti < tp){ //subo com W (tp) ate U
80E     v = pai[w];

F19     if(tp <= otp && otp <= to) remove(v);
7AC     else add(w);

9A1     w = v;
FCA     tp = tin[w];
D88     to = tout[w];
34D   }
B15 }

1D4 int TIME = 0;
FB6 void dfs(int u, int p){
49E   pai[u] = p;
6FD   tin[u] = TIME++;
A2B   order[tin[u]] = u;

70D   for(auto v : g[u])
F6B     if(v != p)
95E       dfs(v, u);
916   tout[u] = TIME-1;
686 }

73F vector<int> MO(vector<Query> &queries){
51F   vector<int> ans(queries.size());
564   dfs(0, 0);

C89   for(auto &[u, v, i] : queries)
563     tie(u, v) = minmax(tin[u], tin[v]);
BFA   sort(queries.begin(), queries.end());

49E   add(0);
7AC   int Lm = 0, Rm = 0;
FE9   for(auto [l, r, idx] : queries){
9D4     if(l < Lm) go_to(Lm, l, Rm), Lm = l;
0E8     if(r > Rm) go_to(Rm, r, Lm), Rm = r;
A5C     if(l > Lm) go_to(Lm, l, Rm), Lm = l;
035     if(r < Rm) go_to(Rm, r, Lm), Rm = r;
830     ans[idx] = getAnswer();
30A   }

BA7   return ans;
64A }
```

5.3 Hungarian

Hungarian Algorithm - Assignment Problem
Algoritmo para o problema de atribuicao minima.

Complexity: O(N^2 * M)

hungarian(int n, int m); -> Retorna o valor do custo minimo
getAssignment(int m) -> Retorna a lista de pares <linha, Coluna> do Minimum Assignment

n -> Numero de Linhas // m -> Numero de Colunas

IMPORTANTE! O algoritmo e 1-indexado
IMPORTANTE! O tipo padrao esta como int, para mudar para outro tipo altere | typedef <TIPO> TP; |

Extra: Para o problema da atribuicao maxima, apenas multiplique os elementos da matriz por -1

```
941 typedef int TP;

3CE const int MAXN = 1e3 + 5;
657 const TP INF = 0x3f3f3f3f;

F31 TP matrix[MAXN][MAXN];
F10 TP row[MAXN], col[MAXN];
E1F int match[MAXN], way[MAXN];

E5E TP hungarian(int n, int m){
715   memset(row, 0, sizeof row);
CD2   memset(col, 0, sizeof col);
187   memset(match, 0, sizeof match);

535   for(int i=1; i<=n; i++)
F95   {
96C     match[0] = i;
23B     int j0 = 0, j1, i0;
76E     TP delta;

693     vector<TP> minv (m+1, INF);
C04     vector<bool> used (m+1, false);

016     do {
472       used[j0] = true;
F81       i0 = match[j0];
B27       j1 = -1;
7DA       delta = INF;

2E2       for(int j=1; j<=m; j++)
F92         if(!used[j]){
76D           TP cur = matrix[i0][j] - row[i0] - col[j];

9F2           if( cur < minv[j] ) minv[j] = cur, way[j] = j0;
821           if(minv[j] < delta) delta = minv[j], j1 = j;
6FD         }

FC9       for(int j=0; j<=m; j++)
E48         if(used[j]){
7AC           row[match[j]] += delta,
429           col[j] -= delta;
72C         }else
299           minv[j] -= delta;

6D4       j0 = j1;
A95     } while(match[j0]);


016     do {
B8C       j1 = way[j0];
77A       match[j0] = match[j1];
6D4       j0 = j1;
196     } while(j0);
7B1   }

A33   return -col[0];
7FF }

3B4 vector<pair<int, int>> getAssignment(int m){
F77   vector<pair<int, int>> ans;

8EA   for(int i=1; i<=m; i++)
843     ans.push_back(make_pair(match[i], i));

BA7   return ans;
01D }
```

5.4 Date

converts Gregorian date to integer (Julian day number)

```
B37 int dateToInt (int m, int d, int y){ return
B8C     + 1461 * (y + 4800 + (m - 14) / 12) / 4
CAD     + 367  * (m - 2    - (m - 14) / 12  * 12)  / 12
47F     - 3    *((y + 4900 + (m - 14) / 12) / 100) / 4
6BC     + d - 32075;
C1B }
```

converts integer (Julian day number) to Gregorian date: day/month/year

```
32D tuple<int, int, int> intToDate(int jd){
402     int x, n, i, j, d, m, y;
33A     x = jd + 68569;
403     n = 4 * x / 146097;
33E     x -= (146097 * n + 3) / 4;
6FC     i = (4000 * (x + 1)) / 1461001;
B1D     x -= 1461 * i / 4 - 31;
FC9     j = 80 * x / 2447;
C8D     d = x - 2447 * j / 80;
179     x = j / 11;
335     m = j + 2 - 12 * x;
23D     y = 100 * (n - 49) + i + x;
B86     return {d, m, y};
4AC }
```

converts integer (Julian day number) to day of week

```
58B string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "
      Sat", "Sun"};
264 string intToWeek (int jd){ return dayOfWeek[jd % 7]; }
```

6 Math

6.1 fexp

```
ll mod = 1e9 + 7;

ll fexp(ll b, ll p){
  ll ans = 1;
  while(p){
    if(p&1) ans = ans * b % mod;
    b = b * b % mod;
    p >>= 1;
  }
  return ans;
}
// O(Log P) // b - Base // p - Potencia
```

6.2 CRT

```
D40 #define ld long double

593 ll modinv(ll a, ll b, ll s0=1, ll s1=0){ return b == 0 ?
      s0 : modinv(b, a%b, s1, s0 - s1 * (a/b)); }
D8B ll mul(ll a, ll b, ll m){
C95     ll q = (ld)a*(ld)b / (ld)m;
1A8     ll r = a*b - q*m;
B8B     return (r + m) % m;
```

```
154 }

28D struct Equation {
4C5     ll mod, ans;
08F     bool valid;
0FC     Equation() { valid = false; }
5E2     Equation(ll a, ll m) { mod = m, ans = (a % m + m) % m, valid = true; }
4D3     Equation(Equation a, Equation b){
355         if(!a.valid || !b.valid){ valid = false; return; }
85C         ll g = gcd(a.mod, b.mod);
DBE         if((a.ans - b.ans) % g != 0){ valid = false; return; }

AF0         valid = true;
B98         mod = a.mod * (b.mod / g);
2F6         ans = a.ans;
5E0         ans += mul( mul(a.mod,  modinv(a.mod, b.mod), mod) ,   (b.ans - a.ans) / g, mod);

C4C         ans = (ans % mod + mod) % mod;
2DB     }
634     Equation operator+(const Equation& b) const { return Equation(*this, b); }
E15 };
D41 // Equation eq1(2, 3); // x = 2 mod 3
D41 // Equation eq2(3, 5); // x = 3 mod 5
D41 // Equation ans = eq1 + eq2;
```

6.3 mint

```
031 const ll mod = 1e9+7;

E54 struct mint {
60E   ll v = 0;
279   mint(ll x=0) : v((x%mod+mod)%mod){}
2D0   mint operator+ (const mint &b) const { ll a = v+b.v; return a < mod ? a : a-mod; }
348   mint operator- (const mint &b) const { ll a = v-b.v; return a <  0  ? a+mod : a; }
AE3   mint operator* (const mint &b) const { return v * b.v % mod; }
834   mint operator/ (const mint &b) const { return v * fexp(b.v, mod-2) % mod;   }
39E     bool operator< (const mint &b) const { return v < b.v; }
A49 };
```

6.4 FFT

Fast Fourier Transform for polynomials multiplication

conv(a, b) = c, where $c[x] = \sum a[i]b[x-i]$.

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.

Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs).

O(N log N) // N=|A|+|B| (1s N <= 2^22)

```
8E9 #define ld double //(10% slower if long double)
A18 typedef complex<ld> CD;
B4C void fft(vector<CD>& a){
A5B   int n = a.size(), L = 31 - __builtin_clz(n);

F82   static vector<complex<long double>> R(2, 1);
6B4   static vector<CD> rt(2, 1);

AD8     for(static int k = 2; k < n; k *= 2){
411         auto x = polar(1.0L, acos(-1.0L)/k);
E92         R.resize(n); rt.resize(n);

1D3     for(int i=k; i<2*k; i++)
CD4             rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
040   }

808   vector<int> rev(n);
5EB   for(int i=0; i<n; i++) rev[i] = (rev[i/2] | (i&1)<<L)/2;
EE4   for(int i=0; i<n; i++) if(i<rev[i]) swap(a[i], a[rev[i]]);

657     for(int k=1; k<n; k*=2)
1E5     for(int i=0; i<n; i+=2*k)
0C2             for(int j=0; j<k; j++){
CD2                 auto x=(ld*)&rt[j+k], y=(ld*)&a[i+j+k];
219                 CD z (x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
D41             //  CD z = rt[j+k] * a[i+j+k]; //(~25% slower, but less code. Delete 2lines above)
20A                 a[i+j+k] = a[i+j] - z;
1B0                 a[i+j] += z;
707             }
F60 }

17B vector<ld> conv(const vector<ld>& a, const vector<ld>& b){
F88   if(a.empty() || b.empty()) return {};
BBB     vector<ld> res(a.size() + b.size() - 1);
E9A     int n = 1<<(32 - __builtin_clz(res.size()));

576     vector<CD> in(n), out(n);
F83     copy(begin(a), end(a), begin(in));
234     for(int i=0; i<b.size(); i++) in[i].imag(b[i]);

21A     fft(in);
11C     for(auto& x : in) x *= x;
2FC     for(int i=0; i<n; i++) out[i] = in[-i&(n-1)] - conj(in[i]);
3D7     fft(out);

E35     for(int i=0; i<res.size(); i++) res[i] = imag(out[i]) / (4*n);
B50     return res;
733 }
```

6.5 FFT MOD

Fast Fourier Transform for polynomials multiplication with MOD
Can be used for convolutions modulo arbitrary integers.
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, mod). !!!
Get the fft function from fft section.
O(N log N) // (2x slower than NTT or FFT)

```
7A4 #include "FFT.cpp"

6D7 template<const int mod> vector<ll> convMod(const vector<ll> &a, const vector<ll> &b){
F88   if (a.empty() || b.empty()) return {};
290   vector<ll> res(a.size() + b.size() - 1);
A04   int B=32-__builtin_clz(res.size()), n=1<<B, cut=int(sqrt(mod));
584   vector<CD> L(n), R(n), outs(n), outl(n);

FCF     for(int i=0; i<a.size(); i++) L[i] = CD((int)a[i] / cut, (int)a[i] % cut);
71C   for(int i=0; i<b.size(); i++) R[i] = CD((int)b[i] / cut, (int)b[i] % cut);
5D5     fft(L), fft(R);

603   for(int i=0; i<n; i++){
39D     int j = -i&(n-1);
65E     outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
91A     outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
20D   }
D08     fft(outl), fft(outs);

2C0     for(int i=0; i<res.size(); i++){
54F     ll av = (ll)(real(outl[i])+.5) % mod;
FA2     ll bv = (ll)(imag(outl[i])+.5) + (ll)(real(outs[i])+.5);
A36         ll cv = (ll)(imag(outs[i])+.5);
557     res[i] = ((av * cut + bv) % mod * cut + cv) % mod;
6B2   }
B50   return res;
F58 }
```

6.6 NTT

Number Theoretic Transform for polynomials multiplication MOD

conv(a, b) = c, where $c[x] = \sum a[i]b[x-i]$.

!!! Inputs must be in [0, mod). !!!

For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back.
Consider using template<const ll mod, const ll root> in conv and ntt if you need more than one mod.
Mod primes must be of the form $2^a b + 1$,
Consider using CRT (Chinese Remainder Theorem) or FFTmod if you need a different MOD.
ntt(a) computes $\hat{f}(k) = \sum_x a[x] g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$.

O(N log N)

```
A6B const ll mod = 998244353, root = 62; //// 9e8 < mod1 < 1e9

15A void ntt(vector<ll> &a){
A5B   int n = a.size(), L = 31 - __builtin_clz(n);

D51     static vector<ll> rt(2, 1);
8EE   for(static int k=2, s=2; k<n; k*=2, s++){
335     rt.resize(n);
8AA     ll z[] = {1, fexp(root, mod >> s)};
631     for(int i=k; i<2*k; i++) rt[i] = rt[i/2] * z[i&1] % mod;
E44   }

808   vector<int> rev(n);
5EB   for(int i=0; i<n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
```

```
EE4   for(int i=0; i<n; i++) if (i < rev[i]) swap(a[i], a[rev[
      i]]);

657     for(int k=1; k<n; k*=2)
1E5     for(int i=0; i<n; i+=2*k)
0C2           for(int j=0; j<k; j++){
86E               ll z = rt[j+k] * a[i+j+k] % mod, &ai =a[i+
      j];
598               a[i+j+k] = ai - z + (z>ai? mod:0);
4B8               ai += z - (ai+z>=mod? mod:0);
D6A           }
FB7 }

CCC vector<ll> conv(const vector<ll> &a, const vector<ll> &b)
     {
F88   if (a.empty() || b.empty()) return {};
919   int s = a.size()+b.size()-1, B = 32 - __builtin_clz(s),
      n = 1<<B;

F94     vector<ll> L(a), R(b), out(n);
6B4   L.resize(n), R.resize(n);
D9E     ntt(L), ntt(R);

649     int inv = fexp(n, mod - 2);
9CD     for(int i=0; i<n; i++) out[-i&(n-1)] = L[i]*R[i] % mod
      * inv % mod;

EC9     ntt(out);
C20     return {out.begin(), out.begin() + s};
4BF }

A01 const ll mod2 = 918552577, root2 = 63; // 9e8 < mod2 < 1e9
      //also valid mods
551 const ll mod3 = 7340033,   root3 = 25; // 7e6 < mod3 < 1e7
```

6.7 random

```
C8A mt19937 rng(chrono::steady_clock::now().time_since_epoch()
      .count());
D41 //int x = rng();

463 int uniform(int l, int r){
A7F   uniform_int_distribution<int> uid(l, r);
F54   return uid(rng);
D9E }
```

7 Geometry

7.1 Point

Dot product p*q $= p \cdot q$ | inner product | norm | lenght^2

$$u \cdot v = x_1 x_2 + y_1 y_2 = \|u\| \, \|v\| \cos\theta.$$

$u \cdot v > 0 \Rightarrow$ angle $\theta < 90^\circ$ (acute);
$u \cdot v = 0 \Rightarrow$ angle $\theta = 90^\circ$ (perpendicular);
$u \cdot v < 0 \Rightarrow$ angle $\theta > 90^\circ$ (obtuse);

Cross product p % q $= p \times q$: | Vector product | Determinant

$$u \times v = x_1 y_2 - y_1 x_2 = \|u\| \, \|v\| \sin\theta.$$

$u \times v > 0 \Rightarrow v$ is to the *left* of u
$u \times v = 0 \Rightarrow u$ and v are collinear.
$u \times v < 0 \Rightarrow v$ is to the *right* of u
It equals the signed area of the parallelogram spanned by u and v.

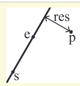
+ p.cross(a, b) $= (a - p) \times (b - p)$
- > 0: CCW (left); ↶
- $= 0$: collinear; ⇒
- < 0: CW (right); ↷

```
8E9 #define ld double

C19 struct PT {
0BE   ll x, y;
0A5   PT(ll x=0, ll y=0) : x(x), y(y) {}

006   PT operator+(const PT&a)const{return PT(x+a.x, y+a.y);}
0DC   PT operator-(const PT&a)const{return PT(x-a.x, y-a.y);}
954   ll operator*(const PT&a)const{return  (x*a.x + y*a.y);}
      //DOT
A68   ll operator%(const PT&a)const{return  (x*a.y - y*a.x);}
      //Cross
B54   PT operator*(ll c) const{ return PT(x*c, y*c); }
B25   PT operator/(ll c) const{ return PT(x/c, y/c); }
5C7   bool operator==(const PT&a) const{ return x == a.x && y
      == a.y; }
B4F   bool operator< (const PT&a) const{ return x != a.x ? x <
      a.x : y < a.y; }

D41   // utils
652   ld len() const { return hypot(x,y); } // sqrt(p*p)
3FC   ll cross(const PT&a, const PT&b) const{ return (a-*this)
      % (b-*this); } // (a-p) % (b-p)
950   int quad() { return (x<0)^3*(y<0); } //cartesian plane
      quadrant |0++|1-+|2--|3+-|
94A   bool ccw(PT q, PT r){ return (q-*this) % (r-q) > 0;}
8EA };

33E ld dist(PT p, PT q){ return sqrtl((p-q)*(p-q)); }
0FB ld proj(PT p, PT q){ return p*q / q.len(); }
D41 //Projection size from A to B

C4F const ld PI = acos(-1.0L);
50C ld angle(PT p, PT q){ return atan2(p%q, p*q); } // Angle
      between vectors p and q [-pi, pi] | acos(a*b/a.len()/b.len
      ()))
E07 ld polarAngle(PT p){  return atan2(p.y, p.x); } // Angle
      to x-axis [-pi, pi]
AF5 bool cmp_ang(PT p, PT q){ return p.quad() != q.quad() ? p.
      quad()<q.quad() : q.ccw(PT(0,0), p); }

874 PT rotateCCW90(PT p){ return PT(-p.y, p.x); } // perp
222 PT rotateCW90(PT p){ return PT(p.y, -p.x); }
96F PT rotateCCW(PT p, ld t){
E8C   ld c = cos(t), s = sin(t);
D80   return PT(p.x*c - p.y*s, p.x*s + p.y*c);
93E }
```

7.2 Line

```
D41 //if p is on line s to e
77D bool onLine(PT s, PT e, PT p){ return p.cross(s, e) == 0;}
```

Returns the **signed dist** from p and the **line** of a and b. Positive value on left side and negative on right as seen from a->b. (a!=b)




```
41B ld lineDist(PT& a, PT& b, PT& p){ return (b-a) % (p-a) / (
      b-a).len(); }
```

Intersection between two lines
Unique -> {+1, pt}
No inter -> { 0, pt}
Infinity -> {-1, pt}May be rounded if inter isn't integer; Watch out for overflow if long long.

```
5E1 pair<int, PT> lineInter(PT a, PT b, PT e, PT f){
8B1   auto d = (b-a) % (f-e);
FC7   if(d == 0) return {-(a.cross(b, e) == 0), PT()}; //
      parallel
F29   auto p = e.cross(b, f), q = e.cross(f, a);
336   return {1, (a * p + b * q) / d};
F59 }
```

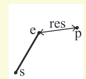
Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead.

```
4E5 PT lineProj(PT a, PT b, PT p, bool refl=false) {
493   PT v = b-a;
7A4   return p - rotateCCW90(v) * (1+refl) * (v%(p-a)) / (v*v)
      ;
7E1 }
```

7.3 Segment

```
D41 //if p is on segment s to e
C39 bool onSegment(PT s, PT e, PT p){
6A6   return p.cross(s, e) == 0 && (s-p) * (e-p) <= 0;
960 }
```

Returns the shortest **distance** between point p and the **segment** s->e.




```
95D ld segmentDist(PT& s, PT& e, PT& p){
BD2   if (s==e) return (p-s).len();
4B2   ld d = (e-s)*(e-s);
385   ld t = min(d, max<ld>(0, (p-s)*(e-s)));
9E6   return ((p-s)*d - (e-s)*t).len() / d;
A45 }
```

Segment intersection
Unique -> {p}
No inter -> { }
Infinity -> {a, b}, the endpoints of the common segment.
May be rounded if inter isn't integer; Watch out for overflow if long long.

```
3DA int sgn(ll x){ return (x>0) - (x<0); }
FFB vector<PT> segInter(PT a, PT b, PT c, PT d){
E62   auto oa = c.cross(d, a), ob = c.cross(d, b);
473   auto oc = a.cross(b, c), od = a.cross(b, d);
914   if(sgn(oa)*sgn(ob) < 0 && sgn(oc)*sgn(od) < 0)
E5B     return {(a*ob - b*oa) / (ob-oa)};
529   set<PT> s;
CCB   if(onSegment(c, d, a)) s.insert(a);
0AD   if(onSegment(c, d, b)) s.insert(b);
3D8   if(onSegment(a, b, c)) s.insert(c);
2FA   if(onSegment(a, b, d)) s.insert(d);
C2C   return {begin(s), end(s)};
276 }
```

7.4 ConvexHull

Given a vector of points, return the convex hull in CCW order.
A convex hull is the smallest convex polygon that contains all the points.

If you want colinear points in border, change the >=0 to >0 in the while's.
WARNING:if collinear and all input PT are collinear, may have duplicated points (the round trip)

```
CD7 vector<PT> ConvexHull(vector<PT> pts, bool sorted=false){
EC1   if(!sorted) sort(begin(pts), end(pts));
6E7   pts.resize(unique(begin(pts), end(pts)) - begin(pts));
64B   if(pts.size() <= 1) return pts;

B4E   int s=0, n=pts.size();
988   vector<PT> h(2*n+1);

AA9   for(int i=0; i<n; h[s++] = pts[i++])
316     while(s > 1 && (pts[i] - h[s-2]) % (h[s-1] - h[s-2]) >= 0 )
351       s--;

61B   for(int i=n-2, t=s; ~i; h[s++] = pts[i--])
644     while(s > t && (pts[i] - h[s-2]) % (h[s-1] - h[s-2]) >= 0 )
351       s--;

CBB   h.resize(s-1);
81C   return h;
CBB } //PT operators needed: {- % == <}
```

Check **if a point is inside convex hull** (CCW, no collinear).If strict == true, then pt on boundary return falseO(log N)

```
3D7 bool isInside(const vector<PT>& h, PT p, bool strict = true){
579   int a = 1, b = h.size() - 1, r = !strict;
795   if(h.size() < 3) return r && onSegment(h[0], h.back(), p);
59E   if(h[0].cross(h[a], h[b]) > 0) swap(a, b);
317   if(h[0].cross(h[a], p) >= r || h[0].cross(h[b], p) <= -r) return false;
48A   while(abs(a-b) > 1){
4F7     int c = (a + b) / 2;
142     if(h[0].cross(h[c], p) > 0) b = c;
1B9     else a = c;
7E3   }
B11   return h[a].cross(h[b], p) < r;
EB9 }
```

Check **if a point is inside convex hull**
O(log N)

```
E13 bool isInside(const vector<PT> &h, PT p){
66D   if(h[0].cross(p, h[1]) > 0 || h[0].cross(p, h.back()) < 0) return false;
B28     int n = h.size(), l=1, r = n-1;
E55     while(l != r){
264         int mid = (l+r+1)/2;
B64         if(h[0].cross(p, h[mid]) < 0) l = mid;
943         else r = mid - 1;
D3D     }
0F2     return h[l].cross(h[(l+1)%n], p) >= 0;
CBC }
```

Given a convex hull h and a point p, returns the indice of h **where the dot product is maximized.**This code assumes that there are NO 3 colinear points!

```
DD1 int maximizeScalarProduct(const vector<PT> &h, PT v) {
A75     int ans = 0, n = h.size();
F37     if(n < 20){
830     for(int i=0; i<n; i++)
070             if(v*h[ans] < v*h[i])
C46                 ans = i;
BA7     return ans;
E80     }

866   for(int rep=0; rep<2; rep++){
D47     int l = 2, r = n-1;
E55     while(l != r){
264       int mid = (l+r+1)/2;
9E8       int f = v*h[mid] >= v*h[mid-1];

FCF       if(rep) f |= v*h[mid-1] < v*h[0];
622       else    f &= v*h[mid]   >= v*h[0];

109       if(f) l = mid;
943       else  r = mid - 1;
9A3     }
48D     if(v*h[ans] < v*h[l]) ans = l;
6A2   }
3D0   if(v*h[ans] < v*h[1] ) ans = 1;
BA7   return ans;
E80 }
```

7.5 Poligons

Returns **twice area of a simple polygon.** area*2 (Shoelace Formula: signed cross product sum)

```
5AB ll Area2x(vector<PT>& p){
604   ll area = 0;
37F   for(int i=2; i < p.size(); i++)
20A     area += (p[i]-p[0]) % (p[i-1]-p[0]);
199   return abs(area);
64B }
```

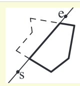
Returns if a point is **inside a triangle** (or in the border).

```
5CA bool ptInsideTriangle(PT p, PT a, PT b, PT c){
58B   if((b-a) % (c-b) < 0) swap(a, b);
805   if(onSegment(a,b,p)) return 1;
1A3   if(onSegment(b,c,p)) return 1;
1DB   if(onSegment(c,a,p)) return 1;
13A   bool x = (b-a) % (p-b) < 0;
B85   bool y = (c-b) % (p-c) < 0;
CE5   bool z = (a-c) % (p-a) < 0;
4B5   return x == y && y == z;
9C6 }
```

Returns the **center of mass for a polygon.** O(n)

```
303 PT polygonCenter(const vector<PT>& v){
313   PT res(0, 0); double A = 0;
E3C   for(int i=0, j=v.size()-1; i<v.size(); j=i++){
FF1     res = res + (v[i]+v[j]) * (v[j]%v[i]);
587     A += v[j] % v[i];
D4F   }
33C   return res / A / 3;
CD0 }
```

PolygonCut: Returns the vertices of the polygon cut away at the left of the line s->e.polygonCut(p, PT(0,0), PT(1,0));



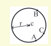
```
767 vector<PT> polygonCut(const vector<PT>& poly, PT s, PT e){
81A   vector<PT> res;
6F1   for(int i=0; i<poly.size(); i++){
431     PT cur = poly[i], prev = i ? poly[i-1] : poly.back();
C5F     auto a = s.cross(e, cur), b = s.cross(e, prev);
498     if((a < 0) != (b < 0)) res.push_back(cur + (prev - cur) * (a / (a - b)));
DDB     if(a < 0) res.push_back(cur);
1E0   }
B50   return res;
D6D }
```

Pick's theorem for **lattice points** in a simple polygon. (lattice points = integer points)Area = insidePts + boundPts/2 - 12A - b + 2 = 2i

```
CDC ll cntInsidePts(ll area_db, ll bound){ return (area_db + 2LL - bound)/2; }
ED9 ll latticePointsInSeg(PT a, PT b){
FA7   ll dx = abs(a.x - b.x);
97A   ll dy = abs(a.y - b.y);
695   return gcd(dx, dy) + 1;
FA7 }
```

7.6 Circles

The **circumcirle of a triangle** is the circle intersecting all three vertices.



```
8BC double ccRadius(PT& A, PT& B, PT& C) {
F6D   return (B-A).len()*(C-B).len()*(A-C).len() / abs(A.cross(B, C))/2;
BEA }
660 PT ccCenter(PT& A, PT& B, PT& C) {
0BF   PT b = C-A, c = B-A;
D0F   return A + rotateCCW90(b*(c*c) - c*(b*b)) / (b%c) / 2;
311 }
```

Return the points at **two circles intersection.** If none or infinity, returns empty

```
240 vector<PT> circleCircleInter(PT a, ld r1, PT b, ld r2){
AC5   if (a == b) return {}; //r1==r2? infinity : none
493   PT v = b-a;

95B   ld d2 = v*v, sum = r1+r2, dif = r1-r2;
102   ld p = (d2 + r1*r1 - r2*r2) / (d2+d2), h2 = r1*r1 - p*p*d2;

975   if(sum*sum < d2 || dif*dif > d2) return {};

56B   PT mid=a+v*p, per=rotateCCW90(v)*sqrt(fmax(0, h2) / d2);

677   set<PT> ans = {mid + per, mid - per};
C85   return {begin(ans), end(ans)};
8C4 }
```

Return the **circle line intersection.** Return a vector of 0,1 or 2 PTs

```
CD6 vector<PT> circleLineInter(PT c, ld r, PT a, PT b){
C12   PT ab = b-a;
288   PT p = a + ab * ((c-a)*ab) / (ab*ab);
A8D   ld s = a.cross(b, c);
90B   ld h2 = r*r - s*s / (ab*ab);
3E4   if(h2 < 0) return {};
071   if(h2 == 0) return {p};
99E   PT h = ab/ab.len() * sqrt(h2);
D65   return {p - h, p + h};
```

```
8BF }
```

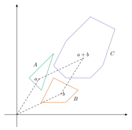
Returns the **minimum enclosing circle** for a set of points. Expected O(n)

```
839 pair<PT, ld> minEnclose(vector<PT> ps) {
504   shuffle(begin(ps), end(ps), mt19937(time(0)));
11E   PT o = ps[0];
F92   ld r=0, EPS = 1 + 1e-8;

860   for(int i=0; i<ps.size(); i++) if(dist(o, ps[i]) > r*EPS){
5CC     o = ps[i], r = 0;

373     for(int j=0; j<i; j++) if(dist(o, ps[j]) > r*EPS){
A30       o = (ps[i] + ps[j]) / 2;
FD2       r = dist(o, ps[i]);

A09       for(int k=0; k<j; k++) if(dist(o, ps[k]) > r*EPS){
FA9         o = ccCenter(ps[i], ps[j], ps[k]);
ED2         r = (o - ps[i]).len();
8BA       }
A2E     }
277   }

645   return {o, r};
AC9 }
```

7.7 Minkowski

Minkowski Sum of convex polygons – O(N)

Returns a convex hull of two polygons minkowski sum.
The minkowski sum of polygons A and B is a polygon such that every vectorinside it is the sum of a vector in A and a vector in B. $A+B = C = \{a+b \mid a \in A,\ b \in B\}$
min(a.size(), b.size()) >= 2



```
D41 // rotate the polygon such that the (bottom, left)-most point is at the first position
C16 void reorder_polygon(vector<PT> &p){
BEC     int pos = 0;
BAA     for(int i = 1; i < p.size(); i++)
8EE         if(pair(p[i].y, p[i].x) < pair(p[pos].y, p[pos].x)) //if(p[i].y < p[pos].y || (p[i].y == p[pos].y && p[i].x < p[pos].x))
E4C             pos = i;
D3C     rotate(p.begin(), p.begin() + pos, p.end());
E7B }

809 vector<PT> minkowski(vector<PT> a, vector<PT> b){
83C     int n = a.size(), m = b.size(), i=0, j=0;
490     reorder_polygon(a); reorder_polygon(b);
5CA     a.push_back(a[0]); a.push_back(a[1]);
258     b.push_back(b[0]); b.push_back(b[1]);

649     vector<PT> c;
59B     while(i < n || j < m){
018         c.push_back(a[i] + b[j]);
47E         auto p = (a[i+1] - a[i]) % (b[j+1] - b[j]);
46D         if(p >= 0) i++;
7D0         if(p <= 0) j++;
266     }
807     return c;
DBA }
```

7.8 LineContainer

```
72C struct Line {
3E2   mutable ll k, m, p;
CA5   bool operator<(const Line& o) const { return k < o.k; }
ABF   bool operator<(ll x) const { return p < x; }
7E3 };

781 struct LineContainer : multiset<Line, less<>> {
FD2   static const ll inf = LLONG_MAX; // Double: inf = 1/.0, div(a,b) = a/b
10F   ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); } //floored division

A1C   bool isect(iterator x, iterator y) {
A95     if(y == end()) return x->p = inf, 0;
9CB     if(x->k == y->k) x->p = x->m > y->m ? inf : -inf;
591     else x->p = div(y->m - x->m, x->k - y->k);
870     return x->p >= y->p;
2FA   }

141   void add_line(ll k, ll m){ // kx + m  //if minimum k*=-1, m*=-1, query*-1
116     auto z = insert({k, m, 0}), y = z++, x = y;
7B1     while(isect(y, z)) z = erase(z);
141     if(x != begin() && isect(--x, y)) isect(x, y = erase(y));
1A4     while((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
17C   }

4AD   ll query(ll x) {
229     assert(!empty());
7D1     auto l = *lower_bound(x);
96A     return l.k * x + l.m;
D21   }
0B9 };
```

8 Theorems

8.1 Propriedades Matemáticas

- **Conjectura de Goldbach:** Todo número par $n > 2$ pode ser representado como $n = a + b$, onde a e b são primos.
- **Primos Gêmeos:** Existem infinitos pares de primos p, $p + 2$.
- **Conjectura de Legendre:** Sempre existe um primo entre n^2 e $(n+1)^2$.
- **Lagrange:** Todo número inteiro pode ser representado como soma de 4 quadrados.
- **Zeckendorf:** Todo número pode ser representado como soma de números de Fibonacci diferentes e não consecutivos.
- **Tripla de Pitágoras (Euclides):** Toda tripla pitagórica primitiva pode ser gerada por $(n^2-m^2, 2nm, n^2+m^2)$ onde n e m são coprimos e um deles é par.
- **Wilson:** n é primo se e somente se $(n-1)! \mod n = n - 1$.
- **Problema do McNugget:** Para dois coprimos x e y, o número de inteiros que não podem ser expressos como $ax + by$ é $(x-1)(y-1)/2$. O maior inteiro não representável é $xy - x - y$.
- **Fermat:** Se p é primo, então $a^{p-1} \equiv 1 \mod p$. Se x e m são coprimos e m primo, então $x^k \equiv x^{k \bmod (m-1)} \mod m$. *Euler:* $x^{\varphi(m)} \equiv 1 \mod m$. $\varphi(m)$ é o totiente de Euler.
- **Teorema Chinês do Resto:** Dado um sistema de congruências:

  $$x \equiv a_1 \mod m_1, \quad \ldots, \quad x \equiv a_n \mod m_n$$

  com m_i coprimos dois a dois. E seja $M_i = \frac{m_1 m_2 \cdots m_n}{m_i}$ e $N_i = M_i^{-1} \mod m_i$. Então a solução é dada por:

  $$x = \sum_{i=1}^{n} a_i M_i N_i$$

  Outras soluções são obtidas somando $m_1 m_2 \cdots m_n$.
- **Números de Catalan:** Exemplo: expressões de parênteses bem formadas. $C_0 = 1$, e:

  $$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1}\binom{2n}{n}$$

- **Bertrand (Ballot):** Com $p > q$ votos, a probabilidade de sempre haver mais votos do tipo A do que B até o fim é: $\frac{p-q}{p+q}$ Permitindo empates: $\frac{p+1-q}{p+1}$. Multiplicando pela combinação total $\binom{p+q}{q}$, obtém-se o número de possibilidades.
- **Linearidade da Esperança:** $E[aX+bY] = aE[X]+bE[Y]$
- **Variância:** $\mathrm{Var}(X) = E[(X-\mu)^2] = E[X^2] - E[X]^2$
- **Progressão Geométrica:** $S_n = a_1 \cdot \frac{q^n-1}{q-1}$
- **Soma dos Cubos:** $\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k\right)^2$
- **Lindström-Gessel-Viennot:** A quantidade de caminhos disjuntos em um grid pode ser computada como o determinante da matriz do número de caminhos.
- **Lema de Burnside:** Número de colares diferentes (sem contar rotações), com m cores e comprimento n:

  $$\frac{1}{n}\left(m^n + \sum_{i=1}^{n-1} m^{\gcd(i,n)}\right)$$

- **Inversão de Möbius:**

  $$\sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1 \\ 0, & \text{caso contrário} \end{cases}$$

- **Propriedades de Coeficientes Binomiais:**

$$\binom{N}{K} = \binom{N}{N-K} = \frac{N}{K}\binom{N-1}{K-1}$$

$$\sum_{k=0}^{m}(-1)^k\binom{n}{k} = (-1)^m\binom{n-1}{m}$$

$$\sum_{m=0}^{n}\binom{m}{k} = \binom{n+1}{k+1}$$

$$\sum_{k=0}^{m}\binom{n+k}{k} = \binom{n+m+1}{m}$$

$$\sum_{k=0}^{n}\binom{n}{k}^2 = \binom{2n}{n}$$

$$\sum_{k=0}^{n}\binom{n}{k} = 2^n$$

$$\sum_{k=0}^{n}k\binom{n}{k} = n \cdot 2^{n-1}$$

$$\sum_{k=0}^{n}\binom{n-k}{k} = F_{n+1}$$

- **Identidades Clássicas:**
  - **Hockey-stick:** $\sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$
  - **Vandermonde:** $\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}$
- **Distribuições de Probabilidade:**
  - **Uniforme:** $X \in \{a, a+1, \ldots, b\}$, $E[X] = \frac{a+b}{2}$
  - **Binomial:** n tentativas com probabilidade p de sucesso:

$$P(X = x) = \binom{n}{x}p^x(1-p)^{n-x}, \quad E[X] = np$$

  - **Geométrica:** Número de tentativas até o primeiro sucesso:

$$P(X = x) = (1-p)^{x-1}p, \quad E[X] = \frac{1}{p}$$

8.2 Geometria

- **Fórmula de Euler:** Em um grafo planar ou poliedro convexo, temos: $V - E + F = 2$ onde V é o número de vértices, E o número de arestas e F o número de faces.
- **Teorema de Pick:** Para polígonos com vértices em coordenadas inteiras:

$$\text{Área} = i + \frac{b}{2} - 1$$

onde i é o número de pontos interiores e b o número de pontos sobre o perímetro.

- **Teorema das Duas Orelhas (Two Ears Theorem):** Todo polígono simples com mais de três vértices possui pelo menos duas "orelhas"— vértices que podem ser removidos sem gerar interseções. A remoção repetida das orelhas resulta em uma triangulação do polígono.
- **Incentro de um Triângulo:** É o ponto de interseção das bissetrizes internas e centro da circunferência inscrita. Se a, b e c são os comprimentos dos lados opostos aos vértices $A(X_a, Y_a)$, $B(X_b, Y_b)$ e $C(X_c, Y_c)$, então o incentro (X, Y) é dado por:

$$X = \frac{aX_a + bX_b + cX_c}{a+b+c}, \quad Y = \frac{aY_a + bY_b + cY_c}{a+b+c}$$

- **Triangulação de Delaunay:** Uma triangulação de um conjunto de pontos no plano tal que nenhum ponto está dentro do círculo circunscrito de qualquer triângulo. Essa triangulação:
  - Maximiza o menor ângulo entre todos os triângulos.
  - Contém a árvore geradora mínima (MST) euclidiana como subconjunto.
- **Fórmula de Brahmagupta:** Para calcular a área de um quadrilátero cíclico (todos os vértices sobre uma circunferência), com lados a, b, c e d:

$$s = \frac{a+b+c+d}{2}, \quad \text{Área} = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

Se $d = 0$ (ou seja, um triângulo), ela se reduz à fórmula de Heron:

$$\text{Área} = \sqrt{(s-a)(s-b)(s-c)s}$$

8.3 Grafos

- **Fórmula de Euler (para grafos planares):**

$$V - E + F = 2$$

onde V é o número de vértices, E o número de arestas e F o número de faces.

- **Handshaking Lemma:** O número de vértices com grau ímpar em um grafo é par.
- **Teorema de Kirchhoff (contagem de árvores geradoras):** Monte a matriz M tal que:

$$M_{i,i} = \deg(i), \quad M_{i,j} = \begin{cases} -1 & \text{se existe aresta } i-j \\ 0 & \text{caso contrário} \end{cases}$$

O número de árvores geradoras (spanning trees) é o determinante de qualquer co-fator de M (remova uma linha e uma coluna).

- **Condições para Caminho Hamiltoniano:**
  - **Teorema de Dirac:** Se todos os vértices têm grau $\geq n/2$, o grafo contém um caminho Hamiltoniano.
  - **Teorema de Ore:** Se para todo par de vértices não adjacentes u e v, temos $\deg(u) + \deg(v) \geq n$, então o grafo possui caminho Hamiltoniano.
- **Algoritmo de Borůvka:** Enquanto o grafo não estiver conexo, para cada componente conexa escolha a aresta de menor custo que sai dela. Essa técnica constrói a árvore geradora mínima (MST).
- **Árvores:**
  - Existem C_n árvores binárias com n vértices (C_n é o n-ésimo número de Catalan).
  - Existem C_{n-1} árvores enraizadas com n vértices.
  - **Fórmula de Cayley:** Existem n^{n-2} árvores com vértices rotulados de 1 a n.
  - **Código de Prüfer:** Remova iterativamente a folha com menor rótulo e adicione o rótulo do vizinho ao código até restarem dois vértices.
- **Fluxo em Redes:**
  - **Corte Mínimo:** Após execução do algoritmo de fluxo máximo, um vértice u está do lado da fonte se $\text{level}[u] \neq -1$.
  - **Máximo de Caminhos Disjuntos:**
    - **Arestas disjuntas:** Use fluxo máximo com capacidades iguais a 1 em todas as arestas.
    - **Vértices disjuntos:** Divida cada vértice v em v_{in} e v_{out}, conectados por aresta de capacidade 1. As arestas que entram vão para v_{in} e as que saem saem de v_{out}.
  - **Teorema de König:** Em um grafo bipartido:

$$\text{Cobertura mínima de vértices} = \text{Matching máximo}$$

O complemento da cobertura mínima de vértices é o conjunto independente máximo.

- **Coberturas:**
  - **Vertex Cover mínimo:** Os vértices da partição X que **não** estão do lado da fonte no corte mínimo, e os vértices da partição Y que **estão** do lado da fonte.
  - **Independent Set máximo:** Complementar da cobertura mínima de vértices.
  - **Edge Cover mínimo:** É $N-$matching, pegando as arestas do matching e mais quaisquer arestas restantes para cobrir os vértices descobertos.
- **Path Cover:**
  - **Node-disjoint path cover mínimo:** Duplicar vértices em tipo A e tipo B e criar grafo bipartido com arestas de $A \to B$. O path cover é $N -$ matching.
  - **General path cover mínimo:** Criar arestas de $A \to B$ sempre que houver caminho de A para B no grafo. O resultado também é $N -$ matching.
- **Teorema de Dilworth:** O path cover mínimo em um grafo dirigido acíclico é igual à **antichain máxima** (conjunto de vértices sem caminhos entre eles).
- **Teorema do Casamento de Hall:** Um grafo bipartido possui um matching completo do lado X se:

$$\forall W \subseteq X, \quad |W| \leq |\text{vizinhos}(W)|$$

- **Fluxo Viável com Capacidades Inferiores e Superiores:** Para rede sem fonte e sumidouro:
  - Substituir a capacidade de cada aresta por $c_{\text{upper}} - c_{\text{lower}}$
  - Criar nova fonte S e sumidouro T
  - Para cada vértice v, compute:

$$M[v] = \sum_{\text{arestas entrando}} c_{\text{lower}} - \sum_{\text{arestas saindo}} c_{\text{lower}}$$

  - Se $M[v] > 0$, adicione aresta (S, v) com capacidade $M[v]$; se $M[v] < 0$, adicione (v, T) com capacidade $-M[v]$.
  - Se todas as arestas de S estão saturadas no fluxo máximo, então um fluxo viável existe. O fluxo viável final é o fluxo computado mais os valores de c_{lower}.

8.4 DP

- **Divide and Conquer Optimization:** Utilizada em problemas do tipo:

$$dp[i][j] = \min_{k<j}\{dp[i-1][k] + C[k][j]\}$$

onde o objetivo é dividir o subsegmento até j em i segmentos com algum custo. A otimização é válida se:

$$A[i][j] \leq A[i][j+1]$$

onde $A[i][j]$ é o valor de k que minimiza a transição.

- **Knuth Optimization:** Aplicável quando:

$$dp[i][j] = \min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$$

e a condição de monotonicidade é satisfeita:

$$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$$

com $A[i][j]$ sendo o índice k que minimiza a transição.

- **Slope Trick:** Técnica usada para lidar com funções lineares por partes e convexas. A função é representada por pontos onde a derivada muda, que podem ser manipulados com `multiset` ou `heap`. Útil para manter o mínimo de funções acumuladas em forma de envelopes convexos.
- **Outras Técnicas e Truques Importantes:**
  - **FFT (Fast Fourier Transform):** Convolução eficiente de vetores.
  - **CHT (Convex Hull Trick):** Otimização para DP com funções lineares e monotonicidade.
  - **Aliens Trick:** Técnica para binarizar o custo em problemas de otimização paramétrica (geralmente em problemas com limite no número de grupos/segmentos).
  - **Bitset:** Utilizado para otimizações de espaço e tempo em DP de subconjuntos ou somas parciais, especialmente em problemas de mochila.

9 Extra

9.1 stressTest

```
P=code   #mude pro filename do codigo
Q=brute  #mude pro filename do brute [correto]
g++ ${P}.cpp -o sol -O2 || exit 1
g++ ${Q}.cpp -o ans -O2 || exit 1
g++ gen.cpp  -o gen -O2 || exit 1
for ((i = 1; ; i++)) do
  echo $i
  ./gen $i > in
  ./sol < in > out
  ./ans < in > out2
  if (! cmp -s out out2) then
    echo "--> entrada:"
    cat in
    echo "--> saida sol:"
    cat out
    echo "--> saida ans:"
    cat out2
    break;
  fi
done
```

9.2 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 whith } 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 }
```