

Descongelen a Victor Moreno

Contents

1 Estructuras de Datos	2		
1.1 Unordered Map	2	3.13 Max Matching	16
1.2 Segment tree Recursivo	2	3.14 LCA	16
1.3 Segment Tree Iterativo	2	3.15 Centroid	17
1.4 Segment Tree Lazy Recursivo	3	3.16 HLD	18
1.5 Segment Tree Lazy Iterativo	4		
1.6 Rope	5	4 Flow	19
1.7 Ordered Set	5	4.1 Dinics	19
1.8 Union Find	5	4.2 Flow's Utilities	19
1.9 Segment Tree Persistente	5	4.3 Min cost-Max Flow	20
1.10 Sparse Table	6	4.4 Hungarian	21
1.11 Wavelet Tree	6	4.5 Edmonds-Karp	21
1.12 Trie	7		
1.13 Treap	7	5 Geometria	22
1.14 Segment Tree Dinamico	8	5.1 Puntos y lineas	22
		5.2 Circulos	24
2 Strings	8	5.3 Poligonos	25
2.1 Aho Corasick	8	5.4 Voronoi y Triangulaciones de Delunay	27
2.2 Dynamic Aho Corasick	9	5.5 HalfPlanes	27
2.3 Hashing	10		
2.4 KMP	10	6 Matematicas	30
2.5 Manacher	11	6.1 Exponenciacion Binaria	30
2.6 Suffix Automaton	11	6.2 GCD y LCD	30
		6.3 Euclides extendido e inverso modular	31
3 Graph	11	6.4 Fibonacci	31
3.1 Structs for Graphs	11	6.5 Criba de Primos	31
3.2 Dijkstra	12	6.6 Triangulo de Pascal	32
3.3 Bellman-Ford	12	6.7 Cambio de bases	32
3.4 Floyd Warshall	13	6.8 Factorizacion	32
3.5 Transitive Closure	13		
3.6 Is bipartite?	13	7 Varios	32
3.7 Topological Sort	14	7.1 String a vector int	32
3.8 Has Cycle?	14	7.2 Generar permutaciones	32
3.9 Articulation Bridges	14	7.3 2-Sat	32
3.10 SCC Kosaraju's	15	7.4 Bits	33
3.11 Kruskal	15	7.5 Matrix	33
3.12 Kuhn's Algorithm	15	7.6 Mo's Algorithm	34
		7.7 PBS	34
		7.8 Dates	35
		8 Template	35

1 Estructuras de Datos

1.1 Unordered Map

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3
4 struct custom_hash {
5     static uint64_t splitmix64(uint64_t x) {
6         // http://xorshift.di.unimi.it/splitmix64.c
7         x += 0x9e3779b97f4a7c15;
8         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
9         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
10        return x ^ (x >> 31);
11    }
12
13    size_t operator()(uint64_t x) const {
14        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now()
15            .time_since_epoch().count();
16        return splitmix64(x + FIXED_RANDOM);
17    }
18 };
19 gp_hash_table<int, int, custom_hash> m1;
20
21 //Function count
22 m1.find(x) != m1.end()

```

1.2 Segment tree Recursivo

```

1 // Point updates, range query
2 const int N = 4e5+5;
3 int st[N], arr[N];
4 void build(int l, int r, int i) {
5     if (l == r) {st[i] = arr[l]; return;}
6     int m = l+r>>1;
7     build(l, m, 2*i+1);
8     build(m+1, r, 2*i+2);
9     st[i] = st[2*i+1] + st[2*i+2]; // !#
10 }
11 void update(int l, int r, int idx, int x, int i) {
12     if (l == r) {st[i] += x; return;}
13     int m = l+r>>1;

```

```

14     if (idx <= m) update(l, m, idx, x, i*2+1);
15     else update(m+1, r, idx, x, i*2+2);
16     st[i] = st[i*2+1] + st[i*2+2];
17 }
18 int query(int l, int r, int a, int b, int i) {
19     if (a > r || b < l) return 0;
20     if (a <= l && r <= b) return st[i];
21     int m = l+r>>1;
22     return query(l, m, a, b, 2*i+1) + query(m+1, r, a, b, 2*i+2);
23 } // idx=0, l=0, r=n-1
24
25 // Range update, point query
26 // Use same build function above, but comment #!
27 void update(int l, int r, int a, int b, int x, int i) {
28     if (a > r || b < l) return;
29     if (a <= l && r <= b) {st[i] += x; return;}
30     int m = l+r>>1;
31     update(l, m, a, b, x, i*2+1);
32     update(m+1, r, a, b, x, i*2+2);
33 }
34 ll query(int l, int r, int idx, int i) {
35     if (idx > r || idx < l) return 0;
36     if (idx <= l && r <= idx) return st[i];
37     int m = l+r>>1;
38     return query(l, m, idx, 2*i+1) + query(m+1, r, idx, 2*i+2) + st[i];
39 }

```

1.3 Segment Tree Iterativo

```

1 //Para procesar queries de tipo k-esimo es necesario crear un arbol
2 // binario perfecto (llenar con 0's)
3 template<typename T>
4 struct SegmentTree{
5     int N;
6     vector<T> ST;
7
8     //Creacion a partir de un arreglo O(n)
9     SegmentTree(int N, vector<T> & arr): N(N){
10         ST.resize(N << 1);
11         for(int i = 0; i < N; ++i)
12             ST[N + i] = arr[i]; //Dato normal
13             ST[N + i] = creaNode(); //Dato compuesto
14         for(int i = N - 1; i > 0; --i)

```

```

14     ST[i] = ST[i << 1] + ST[i << 1 | 1];          //Dato normal
15     ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
16 }
17
18 //Actualizacion de un elemento en la posicion i
19 void update(int i, T value){
20     ST[i += N] = value;          //Dato normal
21     ST[i += N] = creaNodo(); //Dato compuesto
22     while(i >= 1)
23         ST[i] = ST[i << 1] + ST[i << 1 | 1];          //Dato normal
24         ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
25 }
26
27 //query en [l, r]
28 T query(int l, int r){
29     T res = 0; //Dato normal
30     nodo resl = creaNodo(), resr = creaNodo(); //Dato compuesto
31     for(l += N, r += N; l <= r; l >= 1, r >= 1){
32         if(l & 1)         res += ST[l++]; //Dato normal
33         if(!(r & 1))      res += ST[r--]; //Dato normal
34
35         if(l & 1)         resl = merge(resl, ST[l++]); //Dato compuesto
36         if(!(r & 1))      resr = merge(ST[r--], resr); //Dato compuesto
37     }
38     return res;          //Dato normal
39     return merge(resl, resr); //Dato compuesto
40 }
41
42 //Para estas querys es necesario que el st tenga el tam de la
43     siguiente potencia de 2
44 //ll nT = 1;
45 // while(nT<n) nT<=1;
46 //vector<int> a(nT,0);
47
48 //Encontrar k-esimo 1 en un st de 1's
49 int Kth_One(int k) {
50     int i = 0, s = N >> 1;
51     for(int p = 2; p < 2 * N; p <= 1, s >= 1) {
52         if(k < ST[p]) continue;
53         k -= ST[p++]; i += s;
54     }
55     return i;
56 }

```

```

56
57 //i del primer elemento >= k en todo el arr
58 int atLeastX(int k){
59     int i = 0, s = N >> 1;
60     for(int p = 2; p < 2 * N; p <= 1, s >= 1) {
61         if(ST[p] < k) p++, i += s;
62     }
63     if(ST[N + i] < k) i = -1;
64     return i;
65 }
66
67 //i del primer elemento >= k en [l,fin]
68 //Uso atLeastX(k,l,1,nT)
69 int atLeastX(int x, int l, int p, int s) {
70     if(ST[p] < x or s <= 1) return -1;
71     if((p < 1) >= 2 * N)
72         return (ST[p] >= x) - 1;
73     int i = atLeastX(x, l, p < 1, s >> 1);
74     if(i != -1) return i;
75     i = atLeastX(x, l - (s >> 1), p < 1 | 1, s >> 1);
76     if(i == -1) return -1;
77     return (s >> 1) + i;
78 }
79 };

```

1.4 Segment Tree Lazy Recursivo

```

1 const int N = 2e5+10;
2 ll st[4*N+10], lazy[4*N+10], arr[N];
3 void build(int l, int r, int i) {
4     lazy[i] = 0;
5     if (l == r) {st[i] = arr[l]; return;}
6     int m = l+r>>1;
7     build(l, m, 2*i+1);
8     build(m+1, r, 2*i+2);
9     st[i] = st[2*i+1] + st[2*i+2];
10 }
11 void push(int l, int r, int i) {
12     if (!lazy[i]) return;
13     st[i] += (r-l+1) * lazy[i];
14     if (l != r) {
15         lazy[2*i+1] += lazy[i];
16         lazy[2*i+2] += lazy[i];

```

```

17     }
18     lazy[i] = 0;
19 }
20 void update(int l, int r, int a, int b, ll x, int i) {
21     push(l, r, i);
22     if (a > r || b < l) return;
23     if (a <= l && r <= b) {
24         lazy[i] += x;
25         push(l, r, i);
26         return;
27     }
28     int m = l+r>>1;
29     update(l, m, a, b, x, 2*i+1);
30     update(m+1, r, a, b, x, 2*i+2);
31     st[i] = st[2*i+1] + st[2*i+2];
32 }
33 ll query(int l, int r, int a, int b, int i) {
34     if (a > r || b < l) return 0;
35     push(l, r, i);
36     if (a <= l && r <= b) return st[i];
37     int m = l+r>>1;
38     return query(l, m, a, b, 2*i+1) + query(m+1, r, a, b, 2*i+2);
39 } // i=0, l=0, r=n-1, x=value, a,b=range query

```

1.5 Segment Tree Lazy Iterativo

```

1 //Lazy propagation con incremento de u en rango y minimo
2 //Hay varias modificaciones necesarias para suma en ambos
3 template<typename T>
4 struct SegmentTreeLazy{
5     int N,h;
6     vector<T> ST, d;
7
8     //Creacion a partir de un arreglo
9     SegmentTreeLazy(int n, vector<T> &a): N(n){
10         //En caso de inicializar en cero o algo similar, revisar que la
            construccion tenga su respectivo neutro mult y 1
11         ST.resize(N << 1);
12         d.resize(N);
13         h = 64 - __builtin_clzll(n);
14
15         for(int i = 0; i < N; ++i)
16             ST[N + i] = a[i];

```

```

17     //Construir el st sobre la query que se necesita
18     for(int i = N - 1; i > 0; --i)
19         ST[i] = min(ST[i << 1], ST[i << 1 | 1]);
20 }
21
22 //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc
23 void apply(int p, T value) {
24     ST[p] += value;
25     if(p<N) d[p]+= value;
26 }
27
28 // Modifica valores de los padres de p
29 //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc y a
    la respectiva query
30 void build(int p){
31     while(p>1){
32         p >>= 1;
33         ST[p] = min(ST[p << 1], ST[p << 1 | 1]) + d[p];
34         //ST[p] = (ST[p << 1] & ST[p << 1 | 1]) | d[p]; Ejemplos con
            bitwise
35     }
36 }
37
38 // Propagacion desde la raiz a p
39 void push(int p){
40     for (int s = h; s > 0; --s) {
41         int i = p >> s;
42         if (d[i] != 0) {
43             apply(i << 1, d[i]);
44             apply(i << 1 | 1, d[i]);
45             d[i] = 0; //Tener cuidado si estoy haciendo multiplicaciones
46         }
47     }
48 }
49
50 // Sumar v a cada elemento en el intervalo [l, r)
51 void increment(int l, int r, T value) {
52     l += N, r += N;
53     int l0 = l, r0 = r;
54     for (; l < r; l >>= 1, r >>= 1) {
55         if(l & 1) apply(l++, value);
56         if(r & 1) apply(--r, value);
57     }

```

```

58     build(l0);
59     build(r0 - 1);
60 }
61
62 // min en el intervalo [l, r)
63 T range_min(int l, int r) {
64     l += N, r += N;
65     push(l);
66     push(r - 1);
67     T res = LLONG_MAX;
68     //T res = (1 << 30) - 1;    Requerir operacion and
69     for (; l < r; l >>= 1, r >>= 1) {
70         if(l & 1) res = min(res, ST[l++]);
71         //if(res >= mod) res -= mod;
72         if(r & 1) res = min(res, ST[--r]);
73         //if(res >= mod) res -= mod;
74     }
75     return res;
76 }
77
78 };

```

1.6 Rope

```

1 #include <ext/rope>
2 using namespace __gnu_cxx;
3 rope<int> s;
4 // Sequence with O(log(n)) random access, insert, erase at any position
5 // s.push_back(x);
6 // s.insert(i,r) // insert rope r at position i
7 // s.erase(i,k) // erase subsequence [i,i+k)
8 // s.substr(i,k) // return new rope corresponding to subsequence [i,i+k)
9 // s[i] // access ith element (cannot modify)
10 // s.mutable_reference_at(i) // acces ith element (allows modification)
11 // s.begin() and s.end() are const iterators (use mutable_begin(),
    mutable_end() to allow modification)

```

1.7 Ordered Set

```

1 #include<ext/pb_ds/assoc_container.hpp>
2 #include<ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

```

```

5 // find_by_order(i) -> iterator to ith element
6 // order_of_key(k) -> position (int) of lower_bound of k

```

1.8 Union Find

```

1 vector<pair<int,int>>ds(MAX,{-1,0});
2 // Solo siu requieres los elementos del union find, utiliza
3 // dsex en caso contrario borrarlo
4 list<int>dsex[MAX];
5 void init(int n){
6     for(int i=0;i<n;i++)dsex[i].push_back(i);
7 }
8 int find(int x){
9     if(-1==ds[x].first) return x;
10    return ds[x].first=find(ds[x].first);
11 }
12 bool unionDs(int x, int y){
13     int px=find(x),py=find(y);
14     int &rx=ds[px].second,&ry=ds[py].second;
15     if(px==py) return false;
16     else{
17         if(rx>ry){
18             ds[py].first=px;
19         }
20         else{
21             ds[px].first=py;
22             if(rx==ry) ry+=1;
23         }
24     }
25     return true;
26 }

```

1.9 Segment Tree Persistente

```

1 #define inf INT_MAX
2 const int MAX=5e5+2;
3 typedef pair<ll, ll> item;
4 struct node{
5     item val;
6     node *l, *r;
7     node(): l(nullptr),r(nullptr),val({inf,inf}){};
8     node(node *_l,node *_r):l(_l),r(_r){
9         val=min(l->val,r->val);
10    }

```

```

11     node(ll value,ll pos):r(nullptr),l(nullptr){
12         val=make_pair(value,pos);
13     }
14 };
15 pair<ll,ll>all;
16 vector<node*>versions(MAX,nullptr);
17 node* build(int l,int r){
18     if(l==r)return new node(inf,l);
19     int m=(l+r)/2;
20     return new node(build(l,m),build(m+1,r));
21 }
22
23 node* update(node *root,int l,int r,int pos,int val){
24     if(l==r){
25         return new node(val,pos);}
26     int m=(l+r)/2;
27     if(pos<=m) return new node(update(root->l,l,m,pos,val),root->r);
28     return new node(root->l,update(root->r,m+1,r,pos,val));
29 }
30 item query(node *root,int l,int r,int a,int b){
31     if(a>r || b<l) return all;
32     if(a<=l && r<=b) return root->val;
33     int m=(l+r)/2;
34     return min(query(root->l,l,m,a,b),query(root->r,m+1,r,a,b));
35 }

```

1.10 Sparse Table

```

1 //Se usa para RMQ porque se puede hacer en O(1), no acepta updates
2 vector<int>lg;
3 vector<vector<int>>st;
4 int *nums;
5 void init(int n){
6     int logn=(int) log2(n)+1;
7     lg.assign(n+1,0);
8     st.assign(logn,vector<int>(n+1));
9     for(int i=0;i<n;i++) st[0][i]=nums[i];
10    lg[1]=0;
11    for(int i=2;i<=n;i++) lg[i]=lg[i/2]+1;
12    for(int i=1;i<logn;i++)
13        for(int j=0;j+(1<<i)<n;j++)st[i][j]=min(st[i-1][j],st[i-1][j
14        +(1<<(i-1))]);

```

```

15 int query(int a,int b){
16     int logn=lg[(b-a+1)];
17     cout<<st[logn][a]<<endl;
18     return min(st[logn][a],st[logn][b-(1<<logn)+1]);
19 }

```

1.11 Wavelet Tree

```

1 // indexed in 1
2 // from pointer to first element and to to end
3 // x and y The minimum element and y the max element
4 // If you need only one function or more erase the others
5 // If you need to construct other function you only required to
6 // understand the limit, this
7 // are the same
8 struct wavelet_tree{
9     wavelet_tree *l, *r;
10    vector<int> b;
11    wavelet_tree(int *from, int *to, int x, int y){
12        lo = x, hi = y;
13        if(lo == hi or from >= to) return;
14        int mid = (lo+hi)/2;
15        auto f = [mid](int x){ return x <= mid;};
16        b.reserve(to-from+1);
17        b.pb(0);
18        for(auto it = from; it != to; it++)
19            b.push_back(b.back() + f(*it));
20        auto pivot = stable_partition(from, to, f);
21        l = new wavelet_tree(from, pivot, lo, mid);
22        r = new wavelet_tree(pivot, to, mid+1, hi);
23    }
24    //kth smallest element in [l, r]
25    int kth(int l, int r, int k){
26        if(l > r) return 0;
27        if(lo == hi) return lo;
28        int inLeft = b[r] - b[l-1];
29        int lb = b[l-1];
30        int rb = b[r];
31        if(k <= inLeft) return this->l->kth(lb+1, rb, k);
32        return this->r->kth(lb, rb-rb, k-inLeft);
33    }
34    //count of nos in [l, r] Less than or equal to k

```

```

35 int LTE(int l, int r, int k) {
36     if(l > r or k < lo) return 0;
37     if(hi <= k) return r - l + 1;
38     int lb = b[l-1], rb = b[r];
39     return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
40 }
41 //count of nos in [l, r] equal to k
42 int count(int l, int r, int k) {
43     if(l > r or k < lo or k > hi) return 0;
44     if(lo == hi) return r - l + 1;
45     int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
46     if(k <= mid) return this->l->count(lb+1, rb, k);
47     return this->r->count(l-lb, r-rb, k);
48 }
49 };

```

1.12 Trie

```

1 struct trie{
2     int len,id;
3     int children[26];
4     trie(int _id){
5         len=0,id=_id;
6         for(int i=0;i<26;i++)children[i]=-1;
7     }
8 };vector<trie>Trie;Trie.push_back(trie());
9 void inserString(string str,int root){
10     int aux=root;
11     for(int i=0;i<str.size();i++){
12         int index=str[i]-'a';
13         if(Trie[aux].children[index]==-1){
14             Trie.push_back(trie(Trie.size()));
15             Trie[aux].children[index]=Trie.size()-1;
16         }
17         aux=Trie[aux].children[index];
18     }
19     Trie[aux].len=str.size();
20 }
21 bool existInTrie(string str,int root){
22     int aux=root;
23     for(int i=0;i<str.size();i++){
24         int index=str[i]-'a';
25         if(Trie[aux].children[index]==-1) return false;

```

```

26         aux=Trie[aux].children[index];
27     }
28     return Trie[aux].len;
29 }

```

1.13 Treap

```

1 struct Node {
2     int val=0;
3     ll weight, len=1,lazy=0,sum=0;
4     Node *l, *r;
5     Node(int c) : val(c),weight(rand()), l(NULL), r(NULL) {}
6 } *treap;
7 int size(Node *root) { return root ? root->len : 0; }
8 ll sum(Node *root){ return root? root->sum:0;}
9 void pushDown(Node *&root){
10     if(!root || !root->lazy) return;
11     if(root->l) root->l->lazy+=root->lazy;
12     if(root->r) root->r->lazy+=root->lazy;
13     ll num=root->lazy;num*=size(root);
14     root->sum+=num;root->lazy=0;
15 }
16 void recal(Node *&root){
17     if(!root) return;
18     root->len=1+size(root->l)+size(root->r);
19     root->sum=sum(root->l)+sum(root->r)+root->val;
20     root->val+=root->lazy;
21     pushDown(root);
22 }
23 void split(Node *root, Node *&l, Node *&r, int val) {
24     recal(root);
25     if (!root) l = r = NULL;
26     else if (size(root->l) < val) {
27         split(root->r, root->r, r, val - size(root->l) - 1); l = root; recal
28         (l);
29     } else {
30         split(root->l, l, root->l, val); r = root; recal(r);
31     }
32     recal(root);
33 }
34 void merge(Node *&root, Node *l, Node *r) {
35     recal(l);recal(r);
36     if (!l || !r){root = (!l)?r:l;}

```

```

36 else if (l->weight < r->weight) {
37     merge(l->r, l->r, r); root = l;
38 } else {
39     merge(r->l, l, r->l); root = r;
40 }
41 root->len=1+size(root->l)+size(root->r);
42 }
43 // Not necessary functions indexed in 1
44 void insert(Node *&root, Node *nNode, int pos){
45     Node *l=NULL, *r=NULL, *aux=NULL;
46     split(root, l, r, pos-1);
47     merge(aux, l, nNode);
48     merge(root, aux, r);
49 }
50 void delateRange(Node *&root, int l, int r){
51     Node *l1, *r1, *l2, *r2, *aux2;
52     split(root, l1, r1, l-1);
53     split(r1, r1, r2, r-l+1);
54     merge(root, l1, r2);
55 }
56 // queries if you dont need this you can delete recal and push-down
57 // rembember change the size
58 ll query(Node *&root, int l, int r){
59     Node *l1, *r1, *l2, *r2;
60     split(root, l1, r1, l-1);
61     split(r1, r1, l2, r-l+1);
62     ll res=sum(r1);
63     merge(root, l1, r1); merge(root, root, l2);
64     return res;
65 }
66 void update(Node *&root, int l, int r, ll add){
67     Node *l1, *r1, *l2, *r2, *aux;
68     split(root, l1, r1, l-1);
69     split(r1, r1, r2, r-l+1);
70     r1->lazy+=add;
71     merge(l1, l1, r1); merge(root, l1, r2);
72 }
73 // debugging
74 ostream &operator<<(ostream &os, Node *n) {
75     if (!n) return os;
76     os << n->l;
77     os << n->val;
78     os << n->r;

```

```

79     return os;
80 }

```

1.14 Segment Tree Dinamico

```

1 struct dinamicStree{
2     int l,r;
3     dinamicStree *left=NULLptr,*right=NULLptr;
4     ll sum=0;
5     dinamicStree(int l1,int r1){
6         l=l1,r=r1;
7     }
8 };
9 void updateD(int l,int r,int idx,ll x,dinamicStree *node){
10     if(l==r){ node->sum+=x;return;}
11     int m=(l+r)>>1;
12     ll sum=0;
13     if(idx<=m){
14         node->left=(node->left==NULLptr?new dinamicStree(l,m):node->left
15             );
16         updateD(l,m,idx,x,node->left);
17     }
18     else{
19         node->right=(node->right==NULLptr?new dinamicStree(m+1,r):node->
20             right);
21         updateD(m+1,r,idx,x,node->right);
22     }
23     node->sum=(node->left!=NULLptr?node->left->sum:0)+(node->right!=
24         NULLptr?node->right->sum:0);
25 }
26 ll queryD(int a,int b,dinamicStree *node){
27     if(node==NULLptr) return 0;
28     if(a>node->r || b<node->l) return 0;
29     if(a<=node->l && node->r<=b) return node->sum;
30     return queryD(a,b,node->left)+queryD(a,b,node->right);
31 }

```

2 Strings

2.1 Aho Corasick

```

1 int K, I = 1;
2 struct node {

```



```

3     int fail, ch[26] = {};
4     vector<int> lens;
5 } T[500005];
6
7 void add(string s) {
8     int x = 1;
9     for (int i = 0; i < s.size(); i++) {
10         if (T[x].ch[s[i] - 'a'] == 0)
11             T[x].ch[s[i] - 'a'] = ++I;
12         x = T[x].ch[s[i] - 'a'];
13     }
14     T[x].lens.PB(s.size());
15 }
16
17 void build() {
18     queue<int> Q;
19     int x = 1;
20     T[1].fail = 1;
21     for (int i = 0; i < 26; i++) {
22         if (T[x].ch[i])
23             T[T[x].ch[i]].fail = x, Q.push(T[x].ch[i]);
24         else
25             T[x].ch[i] = 1;
26     }
27     while (!Q.empty()) {
28         x = Q.front(); Q.pop();
29         for (int i = 0; i < 26; i++) {
30             if (T[x].ch[i])
31                 T[T[x].ch[i]].fail = T[T[x].fail].ch[i], Q.push(T[x].ch[i]);
32             else
33                 T[x].ch[i] = T[T[x].fail].ch[i];
34         }
35     }
36 }

```

2.2 Dynamic Aho Corasick

```

1 const int MX = 300005, SIG = 26, LMX = 19;
2
3 struct aho_corasick {
4     struct Node {
5         Node *sig[SIG], *fail;

```

```

6     int finish, cnt;
7     Node () : fail(this), finish(0), cnt(0) {
8         for (int i = 0; i < SIG; i++)
9             sig[i] = this;
10    }
11    Node (Node *root) : fail(root), finish(0), cnt(0) {
12        for (int i = 0; i < SIG; i++)
13            sig[i] = root;
14    }
15 };
16 Node *root;
17 aho_corasick() { reset(); }
18 void reset () {
19     root = new Node;
20 }
21 void insert (string &s, int ind) {
22     Node *u = root;
23     for (char c : s) {
24         c -= 'a';
25         if (u->sig[c] == root) {
26             u->sig[c] = new Node(root);
27             u->sig[c]->finish = -1;
28         }
29         u = u->sig[c];
30     }
31     u->finish = ind;
32     u->cnt++;
33 }
34 Node* getFail (Node *u, int c) {
35     while (u != root && u->sig[c] == root)
36         u = u->fail;
37     return u->sig[c];
38 }
39 void build () {
40     queue<Node*> q;
41     for (int i = 0; i < SIG; i++)
42         if (root->sig[i] != root)
43             q.push(root->sig[i]);
44     while (q.size()) {
45         Node *u = q.front();
46         q.pop();
47         for (int i = 0; i < SIG; i++) {
48             Node *v = u->sig[i];

```

```

49     if (v != root) {
50         v->fail = getFail(u->fail, i);
51         v->cnt += v->fail->cnt;
52         q.push(v);
53     }}}}
54 int match (string &t) {
55     Node *u = root;
56     int res = 0;
57     for (int i = 0; i < t.size(); i++) {
58         char c = t[i] - 'a';
59         if (u->sig[c] != root)
60             u = u->sig[c];
61         else
62             u = getFail(u->fail, c);
63         res += u->cnt;
64     }
65     return res;
66 }
67 };
68
69 typedef vector<string*> vs;
70 struct dynamic_aho_corasick {
71     aho_corasick ac[LMX];
72     vs s[LMX];
73     int exi;
74     dynamic_aho_corasick () : exi(0) {}
75     int insert (string &str) {
76         int j = 0;
77         while (exi & (1 << j)) j++;
78         s[j].push_back(new string(str));
79         for (int i = 0; i < j; i++) {
80             for (string *t : s[i]) s[j].push_back(t);
81             s[i].clear();
82             ac[i].reset();
83         }
84         for (string *t : s[j])
85             ac[j].insert(*t, 1);
86         ac[j].build();
87         exi++;
88     }
89     int match (string &t) {
90         int res = 0;
91         for (int i = 0; i < LMX; i++)

```

```

92         if (exi & (1 << i))
93             res += ac[i].match(t);
94         return res;
95     }
96 };

```

2.3 Hashing

```

1 struct Hash{
2     const int mod=1e9+123;
3     const int p=257;
4     vector<int> prefix;
5     static vector<int> pow;
6     Hash(string str){
7         int n=str.size();
8         while(pow.size()<=n){
9             pow.push_back(1LL*pow.back()*p%mod);
10        }
11        vector<int> aux(n+1);
12        prefix=aux;
13        for(int i=0;i<n;i++){
14            prefix[i+1]=(prefix[i]+1LL*str[i]*pow[i])%mod;
15        }
16    }
17    inline int getHashInInerval(int i,int len,int MxPow){
18        int hashing=prefix[i+len]-prefix[i];
19        if(hashing<0) hashing+=mod;
20        hashing=1LL*hashing*pow[MxPow-(len+i-1)]%mod;
21        return hashing;
22    }
23 };
24 vector<int> Hash::pow{1};

```

2.4 KMP

```

1 vector<int> kmp(string s){
2     int n=s.size();
3     vector<int> pi(n);
4     for(int i=1;i<n;i++){
5         int j=pi[i-1];
6         while(j>0 && s[i]!=s[j])j=pi[j-1];
7         if(s[i]==s[j]) j++;
8         pi[i]=j;
9     }

```

```

10     return pi;
11 }

```

2.5 Manacher

```

1 vector<int> manacher_odd(string s) {
2     int n = s.size();
3     s = "$" + s + "^";
4     vector<int> p(n + 2);
5     int l = 1, r = 1;
6     for(int i = 1; i <= n; i++) {
7         p[i] = max(0, min(r - i, p[l + (r - i)]));
8         while(s[i - p[i]] == s[i + p[i]]) {
9             p[i]++;
10        }
11        if(i + p[i] > r) {
12            l = i - p[i], r = i + p[i];
13        }
14    }
15    return vector<int>(begin(p) + 1, end(p) - 1);
16 }
17 vector<int> manacher_even(string s){
18     string even;
19     for(auto c:s){
20         even+='#'+c;
21     }
22     even+='#';
23     return manacher_odd(even);
24 }

```

2.6 Suffix Automaton

```

1 struct node{
2     map<char,int>edges;
3     int link,length,terminal=0;
4     node(int link,int length): link(link),length(length){};
5 };vector<node>sa;
6 // init in main with sa.push_back(node(-1,0));
7 int last=0;
8 // add one by one chars in order
9 void addChar(char s, int pos){
10     sa.push_back(node(0,pos+1));
11     int r=sa.size()-1;
12     int p=last;

```

```

13     while(p >= 0 && sa[p].edges.find(s) == sa[p].edges.end()) {
14         sa[p].edges[s] = r;
15         p = sa[p].link;
16     }
17     if(p != -1) {
18         int q = sa[p].edges[s];
19         if(sa[p].length + 1 == sa[q].length) {
20             sa[r].link = q;
21         } else {
22             sa.push_back(node(sa[q].link,sa[p].length+1));
23             sa[sa.size()-1].edges=sa[q].edges;
24             int qq = sa.size()-1;
25             sa[q].link = qq;
26             sa[r].link= qq;
27             while(p >= 0 && sa[p].edges[s] == q) {
28                 sa[p].edges[s] = qq;
29                 p = sa[p].link;
30             }
31         }
32     }
33     last = r;
34 }
35 // Not necessary functions
36 void findTerminals(){
37     int p = last;
38     while(p > 0) {
39         sa[p].terminal=1;
40         p = sa[p].link;
41     }
42 }

```

3 Graph

3.1 Structs for Graphs

```

1 struct edge{
2     int source, dest, cost;
3     edge(): source(0), dest(0), cost(0){}
4     edge(int dest, int cost): dest(dest), cost(cost){}
5     edge(int source, int dest, int cost): source(source), dest(dest), cost
        (cost){}
6     bool operator==(const edge & b) const{
7         return source == b.source && dest == b.dest && cost == b.cost;

```

```

8   }
9   bool operator<(const edge & b) const{
10      return cost < b.cost;
11   }
12   bool operator>(const edge & b) const{
13      return cost > b.cost;
14   }
15 };
16
17 struct path{
18     int cost = inf;
19     deque<int> vertices;
20     int size = 1;
21     int prev = -1;
22 };
23
24 struct graph{
25     vector<vector<edge>> adjList;
26     vector<vb> adjMatrix;
27     vector<vi> costMatrix;
28     vector<edge> edges;
29     int V = 0;
30     bool dir = false;
31     graph(int n, bool dir): V(n), dir(dir), adjList(n), edges(n),
32         adjMatrix(n, vb(n)), costMatrix(n, vi(n)){
33         for(int i = 0; i < n; ++i)
34             for(int j = 0; j < n; ++j)
35                 costMatrix[i][j] = (i == j ? 0 : inf);
36     }
37     void add(int source, int dest, int cost){
38         adjList[source].emplace_back(source, dest, cost);
39         edges.emplace_back(source, dest, cost);
40         adjMatrix[source][dest] = true;
41         costMatrix[source][dest] = cost;
42         if(!dir){
43             adjList[dest].emplace_back(dest, source, cost);
44             adjMatrix[dest][source] = true;
45             costMatrix[dest][source] = cost;
46         }
47     }
48     void buildPaths(vector<path> & paths){
49         for(int i = 0; i < V; i++){
50             int u = i;

```

```

50         for(int j = 0; j < paths[i].size; j++){
51             paths[i].vertices.push_front(u);
52             u = paths[u].prev;
53         }
54     }
55 }
56 };

```

3.2 Dijkstra

```

1 vector<path> dijkstra(int start){
2     priority_queue<edge, vector<edge>, greater<edge>> cola;
3     vector<path> paths(V);
4     cola.emplace(start, 0);
5     paths[start].cost = 0;
6     while(!cola.empty()){
7         int u = cola.top().dest; cola.pop();
8         for(edge & current : adjList[u]){
9             int v = current.dest;
10            int nuevo = paths[u].cost + current.cost;
11            if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].
12                size){
13                paths[v].prev = u;
14                paths[v].size = paths[u].size + 1;
15            }else if(nuevo < paths[v].cost){
16                paths[v].prev = u;
17                paths[v].size = paths[u].size + 1;
18                cola.emplace(v, nuevo);
19                paths[v].cost = nuevo;
20            }
21        }
22    }
23    buildPaths(paths); // !# - Copy function from above
24    return paths;
25 }

```

3.3 Bellman-Ford

```

1 vector<path> bellmanFord(int start){
2     vector<path> paths(V, path());
3     vi processed(V);
4     vb inQueue(V);
5     queue<int> Q;
6     paths[start].cost = 0;

```

```

7   Q.push(start);
8   while(!Q.empty()){
9       int u = Q.front(); Q.pop(); inQueue[u] = false;
10      if(paths[u].cost == inf) continue;
11      ++processed[u];
12      if(processed[u] == V){
13          cout << "Negative_cycle\n";
14          return {};
15      }
16      for(edge & current : adjList[u]){
17          int v = current.dest;
18          int nuevo = paths[u].cost + current.cost;
19          if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
20              paths[v].prev = u;
21              paths[v].size = paths[u].size + 1;
22          }else if(nuevo < paths[v].cost){
23              if(!inQueue[v]){
24                  Q.push(v);
25                  inQueue[v] = true;
26              }
27              paths[v].prev = u;
28              paths[v].size = paths[u].size + 1;
29              paths[v].cost = nuevo;
30          }
31      }
32  }
33  buildPaths(paths); // !# - Copy function from above
34  return paths;
35 }

```

3.4 Floyd Warshall

```

1   vector<vi> floyd(){
2       vector<vi> tmp = costMatrix;
3       for(int k = 0; k < V; ++k)
4           for(int i = 0; i < V; ++i)
5               for(int j = 0; j < V; ++j)
6                   if(tmp[i][k] != inf && tmp[k][j] != inf)
7                       tmp[i][j] = min(tmp[i][j], tmp[i][k] + tmp[k][j]);
8       return tmp;
9   }

```

3.5 Transitive Closure

```

1   vector<vb> transitiveClosure(){
2       vector<vb> tmp = adjMatrix;
3       for(int k = 0; k < V; ++k)
4           for(int i = 0; i < V; ++i)
5               for(int j = 0; j < V; ++j)
6                   tmp[i][j] = tmp[i][j] || (tmp[i][k] && tmp[k][j]);
7       return tmp;
8   }
9
10  vector<vb> transitiveClosureDFS(){
11      vector<vb> tmp(V, vb(V));
12      function<void(int, int)> dfs = [&](int start, int u){
13          for(edge & current : adjList[u]){
14              int v = current.dest;
15              if(!tmp[start][v]){
16                  tmp[start][v] = true;
17                  dfs(start, v);
18              }
19          }
20      };
21      for(int u = 0; u < V; u++)
22          dfs(u, u);
23      return tmp;
24  }

```

3.6 Is bipartite?

```

1   bool isBipartite(){
2       vi side(V, -1);
3       queue<int> q;
4       for (int st = 0; st < V; ++st){
5           if(side[st] != -1) continue;
6           q.push(st);
7           side[st] = 0;
8           while(!q.empty()){
9               int u = q.front();
10              q.pop();
11              for (edge & current : adjList[u]){
12                  int v = current.dest;
13                  if(side[v] == -1) {
14                      side[v] = side[u] ^ 1;
15                      q.push(v);
16                  }else{

```

```

17         if(side[v] == side[u]) return false;
18     }
19 }
20 }
21 }
22 return true;
23 }

```

3.7 Topological Sort

```

1 vi topologicalSort(){
2     int visited = 0;
3     vi order, indegree(V);
4     for(auto & node : adjList){
5         for(edge & current : node){
6             int v = current.dest;
7             ++indegree[v];
8         }
9     }
10    queue<int> Q;
11    for(int i = 0; i < V; ++i){
12        if(indegree[i] == 0) Q.push(i);
13    }
14    while(!Q.empty()){
15        int source = Q.front();
16        Q.pop();
17        order.push_back(source);
18        ++visited;
19        for(edge & current : adjList[source]){
20            int v = current.dest;
21            --indegree[v];
22            if(indegree[v] == 0) Q.push(v);
23        }
24    }
25    if(visited == V) return order;
26    else return {};
27 }

```

3.8 Has Cycle?

```

1 bool hasCycle(){
2     vi color(V);
3     function<bool(int, int)> dfs = [&](int u, int parent){
4         color[u] = 1;

```

```

5         bool ans = false;
6         int ret = 0;
7         for(edge & current : adjList[u]){
8             int v = current.dest;
9             if(color[v] == 0)
10                 ans |= dfs(v, u);
11             else if(color[v] == 1 && (dir || v != parent || ret++))
12                 ans = true;
13         }
14         color[u] = 2;
15         return ans;
16     };
17     for(int u = 0; u < V; ++u)
18         if(color[u] == 0 && dfs(u, -1))
19             return true;
20     return false;
21 }

```

3.9 Articulation Bridges

```

1 pair<vb, vector<edge>> articulationBridges(){
2     vi low(V), label(V);
3     vb points(V);
4     vector<edge> bridges;
5     int time = 0;
6     function<int(int, int)> dfs = [&](int u, int p){
7         label[u] = low[u] = ++time;
8         int hijos = 0, ret = 0;
9         for(edge & current : adjList[u]){
10             int v = current.dest;
11             if(v == p && !ret++) continue;
12             if(!label[v]){
13                 ++hijos;
14                 dfs(v, u);
15                 if(label[u] <= low[v])
16                     points[u] = true;
17                 if(label[u] < low[v])
18                     bridges.push_back(current);
19                 low[u] = min(low[u], low[v]);
20             }
21             low[u] = min(low[u], label[v]);
22         }
23         return hijos;

```

```

24 };
25 for(int u = 0; u < V; ++u)
26     if(!label[u])
27         points[u] = dfs(u, -1) > 1;
28 return make_pair(points, bridges);
29 }

```

3.10 SCC Kosaraju's

```

1 vector<vi> scc(){
2     vi low(V), label(V);
3     int time = 0;
4     vector<vi> ans;
5     stack<int> S;
6     function<void(int)> dfs = [&](int u){
7         label[u] = low[u] = ++time;
8         S.push(u);
9         for(edge & current : adjList[u]){
10             int v = current.dest;
11             if(!label[v]) dfs(v);
12             low[u] = min(low[u], low[v]);
13         }
14         if(label[u] == low[u]){
15             vi comp;
16             while(S.top() != u){
17                 comp.push_back(S.top());
18                 low[S.top()] = V + 1;
19                 S.pop();
20             }
21             comp.push_back(S.top());
22             S.pop();
23             ans.push_back(comp);
24             low[u] = V + 1;
25         }
26     };
27     for(int u = 0; u < V; ++u)
28         if(!label[u]) dfs(u);
29     return ans;
30 }

```

3.11 Kruskal

```

1 vector<edge> kruskal(){
2     sort(edges.begin(), edges.end());

```

```

3     vector<edge> MST;
4     disjointSet DS(V);
5     for(int u = 0; u < V; ++u)
6         DS.makeSet(u);
7     int i = 0;
8     while(i < edges.size() && MST.size() < V - 1){
9         edge current = edges[i++];
10        int u = current.source, v = current.dest;
11        if(DS.findSet(u) != DS.findSet(v)){
12            MST.push_back(current);
13            DS.unionSet(u, v);
14        }
15    }
16    return MST;
17 }

```

3.12 Kuhn's Algorithm

```

1 bool tryKuhn(int u, vb & used, vi & left, vi & right){
2     if(used[u]) return false;
3     used[u] = true;
4     for(edge & current : adjList[u]){
5         int v = current.dest;
6         if(right[v] == -1 || tryKuhn(right[v], used, left, right)){
7             right[v] = u;
8             left[u] = v;
9             return true;
10        }
11    }
12    return false;
13 }
14 bool augmentingPath(int u, vb & used, vi & left, vi & right){
15     used[u] = true;
16     for(edge & current : adjList[u]){
17         int v = current.dest;
18         if(right[v] == -1){
19             right[v] = u;
20             left[u] = v;
21             return true;
22         }
23    }
24    for(edge & current : adjList[u]){
25        int v = current.dest;

```

```

26     if(!used[right[v]] && augmentingPath(right[v], used, left, right)){
27         right[v] = u;
28         left[u] = v;
29         return true;
30     }
31 }
32 return false;
33 }

```

3.13 Max Matching

```

1 //vertices from the left side numbered from 0 to l-1
2 //vertices from the right side numbered from 0 to r-1
3 //graph[u] represents the left side
4 //graph[u][v] represents the right side
5 //we can use tryKuhn() or augmentingPath()
6 vector<pair<int, int>> maxMatching(int l, int r){
7     vi left(l, -1), right(r, -1);
8     vb used(l);
9     for(int u = 0; u < l; ++u){
10         tryKuhn(u, used, left, right);
11         fill(used.begin(), used.end(), false);
12     }
13     vector<pair<int, int>> ans;
14     for(int u = 0; u < r; ++u){
15         if(right[u] != -1){
16             ans.emplace_back(right[u], u);
17         }
18     }
19     return ans;
20 }
21
22 void dfs(int u, vi & status, vi & parent){
23     status[u] = 1;
24     for(edge & current : adjList[u]){
25         int v = current.dest;
26         if(status[v] == 0){ //not visited
27             parent[v] = u;
28             dfs(v, status, parent);
29         }else if(status[v] == 1){ //explored
30             if(v == parent[u]){
31                 //bidirectional node u<-->v
32             }else{

```

```

33         //back edge u-v
34     }
35     }else if(status[v] == 2){ //visited
36         //forward edge u-v
37     }
38 }
39 status[u] = 2;
40 }

```

3.14 LCA

```

1 struct tree{
2     vi parent, level, weight;
3     vector<vi> dists, DP;
4     int n, root;
5
6     void dfs(int u, graph & G){
7         for(edge & curr : G.adjList[u]){
8             int v = curr.dest;
9             int w = curr.cost;
10            if(v != parent[u]){
11                parent[v] = u;
12                weight[v] = w;
13                level[v] = level[u] + 1;
14                dfs(v, G);
15            }
16        }
17    }
18
19    tree(int n, int root): n(n), root(root), parent(n), level(n), weight(n),
20        dists(n, vi(20)), DP(n, vi(20)){
21        parent[root] = root;
22    }
23
24    tree(graph & G, int root): n(G.V), root(root), parent(G.V), level(G.V),
25        weight(G.V), dists(G.V, vi(20)), DP(G.V, vi(20)){
26        parent[root] = root;
27        dfs(root, G);
28    }
29
30    void pre(){
31        for(int u = 0; u < n; u++){
32            DP[u][0] = parent[u];

```



```

31     dists[u][0] = weight[u];
32 }
33 for(int i = 1; (1 << i) <= n; ++i){
34     for(int u = 0; u < n; ++u){
35         DP[u][i] = DP[DP[u][i - 1]][i - 1];
36         dists[u][i] = dists[u][i - 1] + dists[DP[u][i - 1]][i - 1];
37     }
38 }
39 }
40
41 int ancestor(int p, int k){
42     int h = level[p] - k;
43     if(h < 0) return -1;
44     int lg;
45     for(lg = 1; (1 << lg) <= level[p]; ++lg);
46     lg--;
47     for(int i = lg; i >= 0; --i){
48         if(level[p] - (1 << i) >= h){
49             p = DP[p][i];
50         }
51     }
52     return p;
53 }
54
55 int lca(int p, int q){
56     if(level[p] < level[q]) swap(p, q);
57     int lg;
58     for(lg = 1; (1 << lg) <= level[p]; ++lg);
59     lg--;
60     for(int i = lg; i >= 0; --i){
61         if(level[p] - (1 << i) >= level[q]){
62             p = DP[p][i];
63         }
64     }
65     if(p == q) return p;
66
67     for(int i = lg; i >= 0; --i){
68         if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
69             p = DP[p][i];
70             q = DP[q][i];
71         }
72     }
73     return parent[p];

```

```

74 }
75
76 int dist(int p, int q){
77     if(level[p] < level[q]) swap(p, q);
78     int lg;
79     for(lg = 1; (1 << lg) <= level[p]; ++lg);
80     lg--;
81     int sum = 0;
82     for(int i = lg; i >= 0; --i){
83         if(level[p] - (1 << i) >= level[q]){
84             sum += dists[p][i];
85             p = DP[p][i];
86         }
87     }
88     if(p == q) return sum;
89
90     for(int i = lg; i >= 0; --i){
91         if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
92             sum += dists[p][i] + dists[q][i];
93             p = DP[p][i];
94             q = DP[q][i];
95         }
96     }
97     sum += dists[p][0] + dists[q][0];
98     return sum;
99 }
100 };

```

3.15 Centroid

```

1 vector<int> g[MAXN]; int n;
2 bool tk[MAXN];
3 int fat[MAXN]; // father in centroid decomposition
4 int szt[MAXN]; // size of subtree
5 int calcsz(int x, int f){
6     szt[x] = 1;
7     for(auto y: g[x]) if(y != f && !tk[y]) szt[x] += calcsz(y, x);
8     return szt[x];
9 }
10 void cdfs(int x = 0, int f = -1, int sz = -1) { // O(n log n)
11     if(sz < 0) sz = calcsz(x, -1);
12     for(auto y: g[x]) if(!tk[y] && szt[y] * 2 >= sz) {
13         szt[x] = 0; cdfs(y, x, sz); return;

```

```

14 }
15 tk[x]=true;fat[x]=f;
16 for(auto y:g[x])if(!tk[y])cdfs(y,x);
17 }
18 void centroid(){memset(tk,false,sizeof(tk));cdfs();}

```

3.16 HLD

```

1 struct DT{
2     int n;
3     vll st,lazy;
4     DT(int _n):n(_n),st(4*_n+1),lazy(4*_n+1){}
5     void push(int l, int r, int i) {
6         if (!lazy[i]) return;
7         st[i] += (r-l+1) * lazy[i];
8         if (l != r) {
9             lazy[2*i+1] += lazy[i];
10            lazy[2*i+2] += lazy[i];
11        }
12        lazy[i] = 0;
13    }
14    void update(int l, int r, int a, int b, ll x, int i) {
15        push(l, r, i);
16        if (a > r || b < l) return;
17        if (a <= l && r <= b) {
18            lazy[i] += x;
19            push(l, r, i);
20            return;
21        }
22        int m = l+r>>1;
23        update(l, m, a, b, x, 2*i+1);
24        update(m+1, r, a, b, x, 2*i+2);
25        st[i] = st[2*i+1] + st[2*i+2];
26    }
27    ll query(int l, int r, int a, int b, int i) {
28        if (a > r || b < l) return 0;
29        push(l, r, i);
30        if (a <= l && r <= b) return st[i];
31        int m = l+r>>1;
32        return query(l, m, a, b, 2*i+1) + query(m+1, r, a, b, 2*i+2);
33    }
34 };
35 struct hld{

```

```

36     int n,timer;
37     vi *T,deep,CH,SZE,PT,P,PTN;
38     DT st;
39     hld(int _n):n(_n),deep(_n+1),CH(_n+1),SZE(_n+1),PT(_n+1),PTN(_n+1),P
40         (_n+1),st(_n){
41         timer=1;
42         T=new vector<int>[_n+1];
43     }
44     void dfs(int u,int p){
45         P[u]=p,SZE[u]+=1,deep[u]=deep[p]+1;
46         for(auto v:T[u]){
47             if(v==p)continue;
48             dfs(v,u);
49             SZE[u]+=SZE[v];
50         }
51     }
52     void dfsHLD(int u,int p,int head){
53         int mv=-1,cmv=-1;
54         PTN[u]=timer,PT[timer++]=u,CH[u]=head;
55         for(auto v:T[u]){
56             if(v==p) continue;
57             if(cmv<SZE[v]) mv=v, cmv=SZE[v];
58         }
59         if(mv!=-1) dfsHLD(mv,u,head);
60         for(auto v:T[u]){
61             if(v==p || mv==v)continue;
62             dfsHLD(v,u,v);
63         }
64     }
65     void update(int u,int v,int k){
66         while(CH[u]!=CH[v]){
67             if(deep[CH[u]]<deep[CH[v]])swap(u,v);
68             st.update(0,n,PTN[CH[u]],PTN[u],k,0);
69             u=P[CH[u]];
70         }
71         if(PTN[u]>PTN[v]) swap(u,v);
72         st.update(0,n,PTN[u],PTN[v],k,0);
73     }
74     ll query(int u,int v){
75         ll res=0;
76         while(CH[u]!=CH[v]){
77             if(deep[CH[u]]<deep[CH[v]])swap(u,v);
78             res+=st.query(0,n,PTN[CH[u]],PTN[u],0);

```

```

78         u=P[CH[u]];
79     }
80     if(PTN[u]>PTN[v]) swap(u,v);
81     return res+st.query(0,n,PTN[u],PTN[v],0);
82 }
83 };

```

4 Flow

4.1 Dinics

```

1 struct Dinic {
2     int nodes, src, dst;
3     vector<int> dist, q, work;
4     struct edge {
5         int to, rev;
6         ll f, cap;
7     };
8     vector<vector<edge>> g;
9     Dinic(int x) : nodes(x), g(x), dist(x), q(x), work(x) {}
10    void add_edge(int s, int t, ll cap) {
11        g[s].pb((edge){t, sz(g[t]), 0, cap});
12        g[t].pb((edge){s, sz(g[s]) - 1, 0, 0});
13    }
14    bool dinic_bfs() {
15        fill(all(dist), -1);
16        dist[src] = 0;
17        int qt = 0;
18        q[qt++] = src;
19        for (int qh = 0; qh < qt; qh++) {
20            int u = q[qh];
21            rep(i, 0, sz(g[u])) {
22                edge &e = g[u][i];
23                int v = g[u][i].to;
24                if (dist[v] < 0 && e.f < e.cap)
25                    dist[v] = dist[u] + 1, q[qt++] = v;
26            }
27        }
28        return dist[dst] >= 0;
29    }
30    ll dinic_dfs(int u, ll f) {
31        if (u == dst) return f;
32        for (int &i = work[u]; i < sz(g[u]); i++) {

```

```

33            edge &e = g[u][i];
34            if (e.cap <= e.f) continue;
35            int v = e.to;
36            if (dist[v] == dist[u] + 1) {
37                ll df = dinic_dfs(v, min(f, e.cap - e.f));
38                if (df > 0) {
39                    e.f += df;
40                    g[v][e.rev].f -= df;
41                    return df;
42                }
43            }
44        }
45        return 0;
46    }
47    ll max_flow(int _src, int _dst) {
48        src = _src, dst = _dst;
49        ll result = 0;
50        while (dinic_bfs()) {
51            fill(all(work), 0);
52            while (ll delta = dinic_dfs(src, 1e18)) result += delta;
53        }
54        return result;
55    }
56 };

```

4.2 Flow's Utilities

```

1 // Get path of max flow
2 void dfs_max_flow(int u, int v) {
3     each(i, g[u]) {
4         if (i.f > 0 && i.f < 1e9 && i.f < i.cap && i.to != v) {
5             res[u][i.to % n] = i.f;
6             i.f = 0;
7             dfs_max_flow(i.to, u);
8         }
9     }
10 }
11 // Convert a 2D matrix as a bipartite graph with 2 nodes (in/out)
12 void matrix_to_bipartite_graph(int n, int m) {
13     int s, t, dx[] = {1, -1, 0, 0}, dy[] = {0, 0, 1, -1};
14     Dinic nf(2 * n * m + 2);
15     rep(i, 0, n) {
16         rep(j, 0, m) {

```

```

17     char c = matrix[i][j];
18     int u = 2 * (n * j + i), cap = 1e9;
19     if(c == '#') continue;
20     else if(c == '.') cap = 1;
21     else if(c == 'A') s = u;
22     else if(c == 'B') t = u;
23     nf.add_edge(u, u+1, cap);
24     rep(k,0,4) {
25         int x = i+dx[k], y = j+dy[k], v = 2*(n*y+x);
26         if(x<0 || x>=n || y<0 || y>=m) continue;
27         nf.add_edge(u+1, v, cap);
28     }
29 }
30 }
31 ll mx=nf.max_flow(s,t+1);
32 }
33 // Get min cut
34 void dfs_min_cut(int u){ // Mark saturated nodes from source
35     vis[u] = 1;
36     each(i, g[u])
37         if(!vis[i.to] && i.f < i.cap)
38             dfs_min_cut(i.to);
39 }
40 void print_min_cut(int s) {
41     dfs_min_cut(s);
42     rep(i,0,n) { // Check for not saturated nodes from
43         rep(j,0,m) { // saturated nodes and mark them as part
44             int u = 2 * (n * j + i); // of the answer.
45             if(nf.vis[u]) {
46                 each(v, nf.g[u]){
47                     if(!nf.vis[v.to] && v.cap > 0)
48                         res[i][j] = v.to;
49                 }
50             }
51         }
52     }
53 }

```

4.3 Min cost-Max Flow

```

1 typedef ll tf;
2 typedef ll tc;
3 const tf INFFLOW=1e9;

```

```

4 const tc INFCOST=1e9;
5 struct MCF{
6     int n;
7     vector<tc> prio, pot; vector<tf> curflow; vector<int> prevedge,
8         prevnode;
9     priority_queue<pair<tc, int>, vector<pair<tc, int>>, greater<pair<tc,
10         int>>> q;
11     struct edge{int to, rev; tf f, cap; tc cost;};
12     vector<vector<edge>> g;
13     MCF(int n):n(n),prio(n),curflow(n),prevedge(n),prevnode(n),pot(n),g(n)
14         {}
15     void add_edge(int s, int t, tf cap, tc cost) {
16         g[s].pb((edge){t,sz(g[t]),0,cap,cost});
17         g[t].pb((edge){s,sz(g[s])-1,0,0,-cost});
18     }
19     pair<tf,tc> get_flow(int s, int t) {
20         tf flow=0; tc flowcost=0;
21         while(1){
22             q.push({0, s});
23             fill(ALL(prio),INFCOST);
24             prio[s]=0; curflow[s]=INFFLOW;
25             while(!q.empty()) {
26                 auto cur=q.top();
27                 tc d=cur.fst;
28                 int u=cur.snd;
29                 q.pop();
30                 if(d!=prio[u]) continue;
31                 for(int i=0; i<sz(g[u]); ++i) {
32                     edge &e=g[u][i];
33                     int v=e.to;
34                     if(e.cap<=e.f) continue;
35                     tc nprio=prio[u]+e.cost+pot[u]-pot[v];
36                     if(prio[v]>nprio) {
37                         prio[v]=nprio;
38                         q.push({nprio, v});
39                         prevnode[v]=u; prevedge[v]=i;
40                         curflow[v]=min(curflow[u], e.cap-e.f);
41                     }
42                 }
43             }
44             if(prio[t]==INFCOST) break;
45             fore(i,0,n) pot[i]+=prio[i];
46             tf df=min(curflow[t], INFFLOW-flow);

```

```

44     flow+=df;
45     for(int v=t; v!=s; v=prevnode[v]) {
46         edge &e=g[prevnode[v]][prevedge[v]];
47         e.f+=df; g[v][e.rev].f-=df;
48         flowcost+=df*e.cost;
49     }
50 }
51 return {flow,flowcost};
52 }
53 };

```

4.4 Hungarian

```

1  typedef long double td; typedef vector<int> vi; typedef vector<td> vd;
2  const td INF=1e100;//for maximum set INF to 0, and negate costs
3  bool zero(td x){return fabs(x)<1e-9;}//change to x==0, for ints//
4  struct Hungarian{
5      int n; vector<vd> cs; vi L, R;
6      Hungarian(int N, int M):n(max(N,M)),cs(n,vd(n)),L(n),R(n){
7          fore(x,0,N)fore(y,0,M)cs[x][y]=INF;
8      }
9      void set(int x,int y,td c){cs[x][y]=c;}
10     td assign() {
11         int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
12         fore(i,0,n)u[i]=*min_element(ALL(cs[i]));
13         fore(j,0,n){v[j]=cs[0][j]-u[0];fore(i,1,n)v[j]=min(v[j],cs[i][j]-u[i]);}
14         L=R=vi(n, -1);
15         fore(i,0,n)fore(j,0,n)
16             if(R[j]==-1&&zero(cs[i][j]-u[i]-v[j])){L[i]=j;R[j]=i;mat++;break;}
17         for(;mat<n;mat++){
18             int s=0, j=0, i;
19             while(L[s] != -1)s++;
20             fill(ALL(dad),-1);fill(ALL(sn),0);
21             fore(k,0,n)ds[k]=cs[s][k]-u[s]-v[k];
22             for(;;){
23                 j = -1;
24                 fore(k,0,n)if(!sn[k]&&(j==-1||ds[k]<ds[j]))j=k;
25                 sn[j] = 1; i = R[j];
26                 if(i == -1) break;
27                 fore(k,0,n)if(!sn[k]){
28                     auto new_ds=ds[j]+cs[i][k]-u[i]-v[k];
29                     if(ds[k] > new_ds){ds[k]=new_ds;dad[k]=j;}

```

```

30     }
31 }
32 fore(k,0,n)if(k!=j&&sn[k]){auto w=ds[k]-ds[j];v[k]+=w,u[R[k]]-=w;
33     };
34 u[s] += ds[j];
35 while(dad[j]>=0){int d = dad[j];R[j]=R[d];L[R[j]]=j;j=d;}
36 R[j]=s;L[s]=j;
37 }
38 td value=0;fore(i,0,n)value+=cs[i][L[i]];
39 return value;
40 };

```

4.5 Edmonds-Karps

```

1  struct Edmons{
2      #define ll long long
3      int n;
4      vector<int>d;
5      vector<tuple<int,ll,ll>>edges;
6      vector<vector<int>> adj;
7      vector<pair<int,int>>cam;
8      Edmons(int _n):adj(_n+1,n(_n){}
9      ll sentFlow(int s,int t,ll f){
10         if(s==t)return f;
11         auto &[u,idx]=cam[t];
12         auto cap=get<1>(edges[idx]),&flow=get<2>(edges[idx]);
13         ll push=sentFlow(s,u,min(cap-flow,f));
14         flow+=push;
15         auto &flowr=get<2>(edges[idx^1]);
16         flowr-=push;
17         return push;
18     }
19     bool bfs(int s,int t){
20         d.assign(n+1,-1); d[s]=0;
21         cam.assign(n+1,{-1,-1});
22         queue<int> q({s});
23         while(!q.empty()){
24             int u=q.front();
25             q.pop();
26             for(auto idx:adj[u]){
27                 auto &v=get<0>(edges[idx]);auto &cap=get<1>(edges[idx]),

```

```

28         if(cap-flow>0 && d[v]==-1) d[v]=d[u]+1,cam[v]={u,idx},q.
           push(v);
29     }
30 }
31 return d[t]!=-1;
32 }
33 ll maxFlow(int s,int t){
34     ll flow=0;
35     while(bfs(s,t)){
36         ll push=sentFlow(s,t,1e18);
37         if(!push) return flow;
38         flow+=push;
39     }
40     return flow;
41 }
42 void addEdge(int u,int v, ll c, bool dire=true){
43     if(u==v) return;
44     edges.emplace_back(v,c,0);
45     adj[u].push_back(edges.size()-1);
46     edges.emplace_back(u,(dire?0:c),0);
47     adj[v].push_back(edges.size()-1);
48 }
49 };

```

5 Geometria

5.1 Puntos y lineas

```

1 using ld = long double;
2 const ld eps = 1e-9, inf = numeric_limits<ld>::max(), pi = acos(-1);
3 // For use with integers, just set eps=0 and everything remains the same
4 bool geq(ld a, ld b){return a-b >= -eps;} //a >= b
5 bool leq(ld a, ld b){return b-a >= -eps;} //a <= b
6 bool ge(ld a, ld b){return a-b > eps;} //a > b
7 bool le(ld a, ld b){return b-a > eps;} //a < b
8 bool eq(ld a, ld b){return abs(a-b) <= eps;} //a == b
9 bool neq(ld a, ld b){return abs(a-b) > eps;} //a != b
10
11 struct point{
12     ld x, y;
13     point(): x(0), y(0){}
14     point(ld x, ld y): x(x), y(y){}
15

```

```

16     point operator+(const point & p) const{return point(x + p.x, y + p.y)
17         ;}
18     point operator-(const point & p) const{return point(x - p.x, y - p.y)
19         ;}
20     point operator*(const ld & k) const{return point(x * k, y * k);}
21     point operator/(const ld & k) const{return point(x / k, y / k);}
22
23     point operator+=(const point & p){*this = *this + p; return *this;}
24     point operator-=(const point & p){*this = *this - p; return *this;}
25     point operator*=(const ld & p){*this = *this * p; return *this;}
26     point operator/=(const ld & p){*this = *this / p; return *this;}
27
28     point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*
29         sin(a) + y*cos(a));}
30     point perp() const{return point(-y, x);}
31     ld ang() const{
32         ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
33     }
34     ld dot(const point & p) const{return x * p.x + y * p.y;}
35     ld cross(const point & p) const{return x * p.y - y * p.x;}
36     ld norm() const{return x * x + y * y;}
37     ld length() const{return sqrtl(x * x + y * y);}
38     point unit() const{return (*this) / length();}
39
40     bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y)
41         ;}
42     bool operator!=(const point & p) const{return !(*this == p);}
43     bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x)
44         && le(y, p.y));}
45     bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x)
46         && ge(y, p.y));}
47     bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p
48         .cross(*this), 0) && le(p.dot(*this), 0));}
49 };
50
51 istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
52 ostream &operator<<(ostream &os, const point & p){return os << "(" << p.
53     x << ", " << p.y << ")";}
54
55 int sgn(ld x){
56     if(ge(x, 0)) return 1;
57     if(le(x, 0)) return -1;
58     return 0;
59 }

```

```

51 }
52
53 void polarSort(vector<point> & P, const point & o, const point & v){
54     //sort points in P around o, taking the direction of v as first angle
55     sort(P.begin(), P.end(), [&](const point & a, const point & b){
56         return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).
57             cross(b - o));
58     });
59 }
60
61 bool pointInLine(const point & a, const point & v, const point & p){
62     //line a+tv, point p
63     return eq((p - a).cross(v), 0);
64 }
65
66 bool pointInSegment(const point & a, const point & b, const point & p){
67     //segment ab, point p
68     return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
69 }
70
71 int intersectLinesInfo(const point & a1, const point & v1, const point &
72     a2, const point & v2){
73     //lines a1+tv1 and a2+tv2
74     ld det = v1.cross(v2);
75     if(eq(det, 0)){
76         if(eq((a2 - a1).cross(v1), 0)){
77             return -1; //infinity points
78         }else{
79             return 0; //no points
80         }
81     }else{
82         return 1; //single point
83     }
84 }
85
86 point intersectLines(const point & a1, const point & v1, const point &
87     a2, const point & v2){
88     //lines a1+tv1, a2+tv2
89     //assuming that they intersect
90     ld det = v1.cross(v2);
91     return a1 + v1 * ((a2 - a1).cross(v2) / det);
92 }

```

```

91 int intersectLineSegmentInfo(const point & a, const point & v, const
92     point & c, const point & d){
93     //line a+tv, segment cd
94     point v2 = d - c;
95     ld det = v.cross(v2);
96     if(eq(det, 0)){
97         if(eq((c - a).cross(v), 0)){
98             return -1; //infinity points
99         }else{
100             return 0; //no point
101         }
102     }else{
103         return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point
104         , 0: no point
105     }
106 }
107
108 int intersectSegmentsInfo(const point & a, const point & b, const point
109     & c, const point & d){
110     //segment ab, segment cd
111     point v1 = b - a, v2 = d - c;
112     int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
113     if(t == u){
114         if(t == 0){
115             if(pointInSegment(a, b, c) || pointInSegment(a, b, d) ||
116                 pointInSegment(c, d, a) || pointInSegment(c, d, b)){
117                 return -1; //infinity points
118             }else{
119                 return 0; //no point
120             }
121         }else{
122             return 0; //no point
123         }
124     }else{
125         return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single
126         point, 0: no point
127     }
128 }
129
130 ld distancePointLine(const point & a, const point & v, const point & p){
131     //line: a + tv, point p
132     return abs(v.cross(p - a)) / v.length();
133 }

```

5.2 Circulos

```

1 ld distancePointCircle(const point & c, ld r, const point & p){
2     //point p, circle with center c and radius r
3     return max((ld)0, (p - c).length() - r);
4 }
5
6 point projectionPointCircle(const point & c, ld r, const point & p){
7     //point p (outside the circle), circle with center c and radius r
8     return c + (p - c).unit() * r;
9 }
10
11 pair<point, point> pointsOfTangency(const point & c, ld r, const point &
12     p){
13     //point p (outside the circle), circle with center c and radius r
14     point v = (p - c).unit() * r;
15     ld d2 = (p - c).norm(), d = sqrt(d2);
16     point v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r*r) / d);
17     return {c + v1 - v2, c + v1 + v2};
18 }
19
20 vector<point> intersectLineCircle(const point & a, const point & v,
21     const point & c, ld r){
22     //line a+tv, circle with center c and radius r
23     ld h2 = r*r - v.cross(c - a) * v.cross(c - a) / v.norm();
24     point p = a + v * v.dot(c - a) / v.norm();
25     if(eq(h2, 0)) return {p}; //line tangent to circle
26     else if(1e(h2, 0)) return {}; //no intersection
27     else{
28         point u = v.unit() * sqrt(h2);
29         return {p - u, p + u}; //two points of intersection (chord)
30     }
31 }
32
33 vector<point> intersectSegmentCircle(const point & a, const point & b,
34     const point & c, ld r){
35     //segment ab, circle with center c and radius r
36     vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
37     for(const point & p : P){
38         if(pointInSegment(a, b, p)) ans.push_back(p);
39     }
40     return ans;
41 }

```

```

39
40 pair<point, ld> getCircle(const point & m, const point & n, const point
41     & p){
42     //find circle that passes through points p, q, r
43     point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p
44         - n).perp());
45     ld r = (c - m).length();
46     return {c, r};
47 }
48
49 vector<point> intersectionCircles(const point & c1, ld r1, const point &
50     c2, ld r2){
51     //circle 1 with center c1 and radius r1
52     //circle 2 with center c2 and radius r2
53     point d = c2 - c1;
54     ld d2 = d.norm();
55     if(eq(d2, 0)) return {}; //concentric circles
56     ld pd = (d2 + r1*r1 - r2*r2) / 2;
57     ld h2 = r1*r1 - pd*pd/d2;
58     point p = c1 + d*pd/d2;
59     if(eq(h2, 0)) return {p}; //circles touch at one point
60     else if(1e(h2, 0)) return {}; //circles don't intersect
61     else{
62         point u = d.perp() * sqrt(h2/d2);
63         return {p - u, p + u};
64     }
65 }
66
67 int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2)
68 {
69     //test if circle 2 is inside circle 1
70     //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if
71     they overlap
72     ld l = r1 - r2 - (c1 - c2).length();
73     return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
74 }
75
76 int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2)
77 {
78     //test if circle 2 is outside circle 1
79     //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if
80     they overlap
81     ld l = (c1 - c2).length() - (r1 + r2);

```



```

75     return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
76 }
77
78 int pointInCircle(const point & c, ld r, const point & p){
79     //test if point p is inside the circle with center c and radius r
80     //returns "0" if it's outside, "-1" if it's in the perimeter, "1" if
        it's inside
81     ld l = (p - c).length() - r;
82     return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
83 }
84
85 vector<vector<point>> tangents(const point & c1, ld r1, const point & c2
        , ld r2, bool inner){
86     //returns a vector of segments or a single point
87     if(inner) r2 = -r2;
88     point d = c2 - c1;
89     ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
90     if(eq(d2, 0) || le(h2, 0)) return {};
91     point v = d*dr/d2;
92     if(eq(h2, 0)) return {{c1 + v*r1}};
93     else{
94         point u = d.perp()*sqrt(h2)/d2;
95         return {{c1 + (v - u)*r1, c2 + (v - u)*r2}, {c1 + (v + u)*r1, c2 + (
            v + u)*r2}};
96     }
97 }
98
99 ld signed_angle(const point & a, const point & b){
100     return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length() * b.length()));
101 }
102
103 ld intersectPolygonCircle(const vector<point> & P, const point & c, ld r
    ){
104     //Gets the area of the intersection of the polygon with the circle
105     int n = P.size();
106     ld ans = 0;
107     for(int i = 0; i < n; ++i){
108         point p = P[i], q = P[(i+1)%n];
109         bool p_inside = (pointInCircle(c, r, p) != 0);
110         bool q_inside = (pointInCircle(c, r, q) != 0);
111         if(p_inside && q_inside){
112             ans += (p - c).cross(q - c);
113         }else if(p_inside && !q_inside){

```

```

114         point s1 = intersectSegmentCircle(p, q, c, r)[0];
115         point s2 = intersectSegmentCircle(c, q, c, r)[0];
116         ans += (p - c).cross(s1 - c) + r*r * signed_angle(s1 - c, s2 - c);
117     }else if(!p_inside && q_inside){
118         point s1 = intersectSegmentCircle(c, p, c, r)[0];
119         point s2 = intersectSegmentCircle(p, q, c, r)[0];
120         ans += (s2 - c).cross(q - c) + r*r * signed_angle(s1 - c, s2 - c);
121     }else{
122         auto info = intersectSegmentCircle(p, q, c, r);
123         if(info.size() <= 1){
124             ans += r*r * signed_angle(p - c, q - c);
125         }else{
126             point s2 = info[0], s3 = info[1];
127             point s1 = intersectSegmentCircle(c, p, c, r)[0];
128             point s4 = intersectSegmentCircle(c, q, c, r)[0];
129             ans += (s2 - c).cross(s3 - c) + r*r * (signed_angle(s1 - c, s2 -
                c) + signed_angle(s3 - c, s4 - c));
130         }
131     }
132 }
133 return abs(ans)/2;
134 }

```

5.3 Poligonos

```

1 ld perimeter(vector<point> & P){
2     int n = P.size();
3     ld ans = 0;
4     for(int i = 0; i < n; i++){
5         ans += (P[i] - P[(i + 1) % n]).length();
6     }
7     return ans;
8 }
9
10 ld area(vector<point> & P){
11     int n = P.size();
12     ld ans = 0;
13     for(int i = 0; i < n; i++){
14         ans += P[i].cross(P[(i + 1) % n]);
15     }
16     return abs(ans / 2);
17 }
18

```

```

19 vector<point> convexHull(vector<point> P){
20     sort(P.begin(), P.end());
21     vector<point> L, U;
22     for(int i = 0; i < P.size(); i++){
23         while(L.size() >= 2 && leq((L[L.size() - 2] - P[i]).cross(L[L.size()
24             - 1] - P[i]), 0)){
25             L.pop_back();
26         }
27         L.push_back(P[i]);
28     }
29     for(int i = P.size() - 1; i >= 0; i--){
30         while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size()
31             - 1] - P[i]), 0)){
32             U.pop_back();
33         }
34         U.push_back(P[i]);
35     }
36     L.pop_back();
37     U.pop_back();
38     L.insert(L.end(), U.begin(), U.end());
39     return L;
40 }
41
42 bool pointInPerimeter(const vector<point> & P, const point & p){
43     int n = P.size();
44     for(int i = 0; i < n; i++){
45         if(pointInSegment(P[i], P[(i + 1) % n], p)){
46             return true;
47         }
48     }
49     return false;
50 }
51
52 bool crossesRay(const point & a, const point & b, const point & p){
53     return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) >
54         0;
55 }
56
57 int pointInPolygon(const vector<point> & P, const point & p){
58     if(pointInPerimeter(P, p)){
59         return -1; //point in the perimeter
60     }
61     int n = P.size();

```

```

59     int rays = 0;
60     for(int i = 0; i < n; i++){
61         rays += crossesRay(P[i], P[(i + 1) % n], p);
62     }
63     return rays & 1; //0: point outside, 1: point inside
64 }
65
66 //point in convex polygon in O(log n)
67 //make sure that P is convex and in ccw
68 //before the queries, do the preprocess on P:
69 // rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
70 // int right = max_element(P.begin(), P.end()) - P.begin();
71 //returns 0 if p is outside, 1 if p is inside, -1 if p is in the
    perimeter
72 int pointInConvexPolygon(const vector<point> & P, const point & p, int
    right){
73     if(p < P[0] || P[right] < p) return 0;
74     int orientation = sgn((P[right] - P[0]).cross(p - P[0]));
75     if(orientation == 0){
76         if(p == P[0] || p == P[right]) return -1;
77         return (right == 1 || right + 1 == P.size()) ? -1 : 1;
78     }else if(orientation < 0){
79         auto r = lower_bound(P.begin() + 1, P.begin() + right, p);
80         int det = sgn((p - r[-1]).cross(r[0] - r[-1])) - 1;
81         if(det == -2) det = 1;
82         return det;
83     }else{
84         auto l = upper_bound(P.rbegin(), P.rend() - right - 1, p);
85         int det = sgn((p - l[0]).cross((l == P.rbegin() ? P[0] : l[-1]) - l
86             [0])) - 1;
87         if(det == -2) det = 1;
88         return det;
89     }
90 }
91
92 vector<point> cutPolygon(const vector<point> & P, const point & a, const
    point & v){
93     //returns the part of the convex polygon P on the left side of line a +
94     tv
95     int n = P.size();
96     vector<point> lhs;
97     for(int i = 0; i < n; ++i){
98         if(geq(v.cross(P[i] - a), 0)){

```

```

97     lhs.push_back(P[i]);
98 }
99 if(intersectLineSegmentInfo(a, v, P[i], P[(i+1)%n]) == 1){
100     point p = intersectLines(a, v, P[i], P[(i+1)%n] - P[i]);
101     if(p != P[i] && p != P[(i+1)%n]){
102         lhs.push_back(p);
103     }
104 }
105 }
106 return lhs;
107 }

```

5.4 HalfPlanes

```

1 struct plane{
2     point a, v;
3     plane(): a(), v(){}
4     plane(const point& a, const point& v): a(a), v(v){}
5
6     point intersect(const plane& p) const{
7         ld t = (p.a - a).cross(p.v) / v.cross(p.v);
8         return a + v*t;
9     }
10
11     bool outside(const point& p) const{ // test if point p is strictly
12         outside
13         return le(v.cross(p - a), 0);
14     }
15
16     bool inside(const point& p) const{ // test if point p is inside or in
17         the boundary
18         return geq(v.cross(p - a), 0);
19     }
20
21     bool operator<(const plane& p) const{ // sort by angle
22         auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
23         auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
24         return lhs < rhs;
25     }
26
27     bool operator==(const plane& p) const{ // paralell and same directions
28         , not really equal
29         return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
30     }
31 }

```

```

27 }
28 };
29
30 vector<point> halfPlaneIntersection(vector<plane> planes){
31     planes.push_back({{0, -inf}, {1, 0}});
32     planes.push_back({{inf, 0}, {0, 1}});
33     planes.push_back({{0, inf}, {-1, 0}});
34     planes.push_back({{-inf, 0}, {0, -1}});
35     sort(planes.begin(), planes.end());
36     planes.erase(unique(planes.begin(), planes.end()), planes.end());
37     deque<plane> ch;
38     deque<point> poly;
39     for(const plane& p : planes){
40         while(ch.size() >= 2 && p.outside(poly.back())) ch.pop_back(), poly.
41             pop_back();
42         while(ch.size() >= 2 && p.outside(poly.front())) ch.pop_front(),
43             poly.pop_front();
44         if(p.v.half({1, 0}) && poly.empty()) return {};
45         ch.push_back(p);
46         if(ch.size() >= 2) poly.push_back(ch[ch.size()-2].intersect(ch[ch.
47             size()-1]));
48     }
49     while(ch.size() >= 3 && ch.front().outside(poly.back())) ch.pop_back()
50         , poly.pop_back();
51     while(ch.size() >= 3 && ch.back().outside(poly.front())) ch.pop_front
52         (), poly.pop_front();
53     poly.push_back(ch.back().intersect(ch.front()));
54     return vector<point>(poly.begin(), poly.end());
55 }

```

5.5 Voronoi y Triangulaciones de Delunay

```

1 /*
2     Functions required
3     getCircle, InteseccLine, sgn
4     ConvexHull Lineal and Noraml
5 */
6 //
7 const point inf_pt(inf, inf);
8 struct QuadEdge{
9     point origin;
10    QuadEdge* rot = nullptr;
11    QuadEdge* onext = nullptr;

```

```

12  bool used = false;
13  QuadEdge* rev() const{return rot->rot;}
14  QuadEdge* lnext() const{return rot->rev()->onext->rot;}
15  QuadEdge* oprev() const{return rot->onext->rot;}
16  point dest() const{return rev()->origin;}
17  };
18
19  QuadEdge* make_edge(const point & from, const point & to){
20      QuadEdge* e1 = new QuadEdge;
21      QuadEdge* e2 = new QuadEdge;
22      QuadEdge* e3 = new QuadEdge;
23      QuadEdge* e4 = new QuadEdge;
24      e1->origin = from;
25      e2->origin = to;
26      e3->origin = e4->origin = inf_pt;
27      e1->rot = e3;
28      e2->rot = e4;
29      e3->rot = e2;
30      e4->rot = e1;
31      e1->onext = e1;
32      e2->onext = e2;
33      e3->onext = e4;
34      e4->onext = e3;
35      return e1;
36  }
37
38  void splice(QuadEdge* a, QuadEdge* b){
39      swap(a->onext->rot->onext, b->onext->rot->onext);
40      swap(a->onext, b->onext);
41  }
42
43  void delete_edge(QuadEdge* e){
44      splice(e, e->oprev());
45      splice(e->rev(), e->rev()->oprev());
46      delete e->rev()->rot;
47      delete e->rev();
48      delete e->rot;
49      delete e;
50  }
51
52  QuadEdge* connect(QuadEdge* a, QuadEdge* b){
53      QuadEdge* e = make_edge(a->dest(), b->origin);
54      splice(e, a->lnext());

```

```

55      splice(e->rev(), b);
56      return e;
57  }
58
59  bool left_of(const point & p, QuadEdge* e){
60      return ge((e->origin - p).cross(e->dest() - p), 0);
61  }
62
63  bool right_of(const point & p, QuadEdge* e){
64      return le((e->origin - p).cross(e->dest() - p), 0);
65  }
66
67  __int128_t det3(__int128_t a1, __int128_t a2, __int128_t a3, __int128_t
        b1, __int128_t b2, __int128_t b3, __int128_t c1, __int128_t c2,
        __int128_t c3) {
68      return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1
        * c2 - c1 * b2);
69  }
70
71  bool in_circle(const point & a, const point & b, const point & c, const
        point & d) {
72      __int128_t det = -det3(b.x, b.y, b.norm(), c.x, c.y, c.norm(), d.x, d.
        y, d.norm());
73      det += det3(a.x, a.y, a.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm()
        );
74      det -= det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), d.x, d.y, d.norm()
        );
75      det += det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), c.x, c.y, c.norm()
        );
76      return det > 0;
77  }
78
79  pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<point> & P){
80      if(r - l + 1 == 2){
81          QuadEdge* res = make_edge(P[l], P[r]);
82          return make_pair(res, res->rev());
83      }
84      if(r - l + 1 == 3){
85          QuadEdge *a = make_edge(P[l], P[l + 1]), *b = make_edge(P[l + 1], P[
            r]);
86          splice(a->rev(), b);
87          int sg = sgn((P[l + 1] - P[l]).cross(P[r] - P[l]));
88          if(sg == 0)

```

```

89     return make_pair(a, b->rev());
90     QuadEdge* c = connect(b, a);
91     if(sg == 1)
92         return make_pair(a, b->rev());
93     else
94         return make_pair(c->rev(), c);
95 }
96 int mid = (l + r) / 2;
97 QuadEdge *ldo, *ldi, *rdo, *rdi;
98 tie(ldo, ldi) = build_tr(l, mid, P);
99 tie(rdi, rdo) = build_tr(mid + 1, r, P);
100 while(true){
101     if(left_of(rdi->origin, ldi)){
102         ldi = ldi->lnext();
103         continue;
104     }
105     if(right_of(ldi->origin, rdi)){
106         rdi = rdi->rev()->onext;
107         continue;
108     }
109     break;
110 }
111 QuadEdge* basel = connect(rdi->rev(), ldi);
112 auto valid = [&basel](QuadEdge* e){return right_of(e->dest(), basel)
113     ;};
114 if(ldi->origin == ldo->origin)
115     ldo = basel->rev();
116 if(rdi->origin == rdo->origin)
117     rdo = basel;
118 while(true){
119     QuadEdge* lcand = basel->rev()->onext;
120     if(valid(lcand)){
121         while(in_circle(basel->dest(), basel->origin, lcand->dest(), lcand
122             ->onext->dest())){
123             QuadEdge* t = lcand->onext;
124             delete_edge(lcand);
125             lcand = t;
126         }
127     }
128     QuadEdge* rcand = basel->oprev();
129     if(valid(rcand)){
130         while(in_circle(basel->dest(), basel->origin, rcand->dest(), rcand
131             ->oprev()->dest())){

```

```

129         QuadEdge* t = rcand->oprev();
130         delete_edge(rcand);
131         rcand = t;
132     }
133 }
134 if(!valid(lcand) && !valid(rcand))
135     break;
136 if(!valid(lcand) || (valid(rcand) && in_circle(lcand->dest(), lcand
137     ->origin, rcand->origin, rcand->dest())))
138     basel = connect(rcand, basel->rev());
139 else
140     basel = connect(basel->rev(), lcand->rev());
141 }
142 return make_pair(ldo, rdo);
143 }
144 vector<vector<point>> delaunay(vector<point> P){
145     sort(P.begin(), P.end());
146     auto res = build_tr(0, (int)P.size() - 1, P);
147     QuadEdge* e = res.first;
148     vector<QuadEdge*> edges = {e};
149     while(le((e->dest() - e->onext->dest()).cross(e->origin - e->onext->
150         dest()), 0))
151         e = e->onext;
152     auto add = [&P, &e, &edges]() {
153         QuadEdge* curr = e;
154         do{
155             curr->used = true;
156             P.push_back(curr->origin);
157             edges.push_back(curr->rev());
158             curr = curr->lnext();
159         }while(curr != e);
160     };
161     add();
162     P.clear();
163     int kek = 0;
164     while(kek < (int)edges.size())
165         if(!(e = edges[kek++])->used)
166             add();
167     vector<vector<point>> ans;
168     for(int i = 0; i < (int)P.size(); i += 3){
169         ans.push_back({P[i], P[i + 1], P[i + 2]});

```

```

170     return ans;
171 }
172 vector<vector<point>> voronoi (vector<point> P) {
173     vector<vector<point>> cells(P.size());
174     point border[4];
175     ld minSizeBorder[4];
176     minSizeBorder[0]=minSizeBorder[1]=minSizeBorder[2]=minSizeBorder[3]=1
        e15;
177     rep(i, 0,P.size()) P[i].idx = i;
178     auto ch = convexHull(P);
179     if (ch.size() > 2) {
180         auto dt = delaunay(P);
181         for (auto &tri : dt) {
182             point c = getCircle(tri[0], tri[1], tri[2]);
183             for (auto &p : tri) cells[p.idx].pb(c);
184         }
185         ch = convexHullWithColinear(P);
186         rep (i, 0,ch.size()) {
187             point a = ch[i];point b = ch[(i + 1)% ch.size()];
188             point mid = (a + b)/2;
189             point c = (mid+(a - mid).perp()*inf_coord);
190             cells[a.idx].pb(c);
191             cells[b.idx].pb(c);
192         }
193     } else if (ch.size() == 2) {
194         sort(all(P));
195         rep (i, 0,(int)P.size() - 1) {
196             point mid = (P[i + 1] + P[i]) / 2;
197             point a =(mid+ (P[i] - mid).perp()*inf_coord);
198             point b =(mid+ (P[i] - mid).perp()*-inf_coord);
199             rep(j, 0,2) {
200                 cells[P[i + j].idx].pb(a);
201                 cells[P[i + j].idx].pb(b);
202             }
203         }
204     } else {
205         cells[0] = {
206             rec[0],rec[1],rec[2],rec[3]
207         };
208     }
209     for (auto &cell : cells) {
210         cell = convexHull(cell);
211     }

```

```

212     return cells;
213 }

```

6 Matematicas

6.1 Exponenciacion Binaria

```

1  ll binpow(ll a, ll b, ll mod) {
2      a %= mod;
3      ll res = 1;
4      while (b > 0) {
5          if (b & 1)
6              res = res * a % mod;
7          a = a * a % mod;
8          b >>= 1;
9      }
10     return res;
11 }
12
13 ll binpow(ll a, ll b) {
14     if (b == 0)
15         return 1;
16     ll res = binpow(a, b / 2);
17     if (b % 2)
18         return res * res * a;
19     else
20         return res * res;
21 }

```

6.2 GCD y LCD

```

1  ll gcd(ll a, ll b){
2      ll r;
3      while(b != 0) r = a % b, a = b, b = r;
4      return a;
5  }
6
7  ll lcm(ll a, ll b){
8      return b * (a / gcd(a, b));
9  }
10
11 ll gcd(const vector<ll>& nums){
12     ll ans = 0;

```

```

13   for(ll num : nums) ans = gcd(ans, num);
14   return ans;
15 }
16
17 ll lcm(const vector<ll>& nums){
18     ll ans = 1;
19     for(ll num : nums) ans = lcm(ans, num);
20     return ans;
21 }

```

6.3 Euclides extendido e inverso modular

```

1 tuple<lli, lli, lli> extendedGcd(lli a, lli b){
2     if(b == 0){
3         if(a > 0) return {a, 1, 0};
4         else return {-a, -1, 0};
5     }else{
6         auto[d, x, y] = extendedGcd(b, a%b);
7         return {d, y, x - y*(a/b)};
8     }
9 }
10
11 lli modularInverse(lli a, lli m){
12     auto[d, x, y] = extendedGcd(a, m);
13     if(d != 1) return -1; // inverse doesn't exist
14     if(x < 0) x += m;
15     return x;
16 }

```

6.4 Fibonacci

```

1 //very fast fibonacci
2 inline void modula(lli & n, lli mod){
3     while(n >= mod) n -= mod;
4 }
5
6 lli fibo(lli n, lli mod){
7     array<lli, 2> F = {1, 0};
8     lli p = 1;
9     for(lli v = n; v >= 1; p <= 1);
10    array<lli, 4> C;
11    do{
12        int d = (n & p) != 0;
13        C[0] = C[3] = 0;

```

```

14    C[d] = F[0] * F[0] % mod;
15    C[d+1] = (F[0] * F[1] << 1) % mod;
16    C[d+2] = F[1] * F[1] % mod;
17    F[0] = C[0] + C[2] + C[3];
18    F[1] = C[1] + C[2] + (C[3] << 1);
19    modula(F[0], mod), modula(F[1], mod);
20 }while(p >= 1);
21 return F[1];
22 }
23
24 const long M = 1000000007; // modulo
25 map<long, long> F;
26
27 long f(long n) {
28     if (F.count(n)) return F[n];
29     long k=n/2;
30     if (n%2==0) { // n=2*k
31         return F[n] = (f(k)*f(k) + f(k-1)*f(k-1)) % M;
32     } else { // n=2*k+1
33         return F[n] = (f(k)*f(k+1) + f(k-1)*f(k)) % M;
34     }
35 }
36
37 main(){
38     long n;
39     F[0]=F[1]=1;
40     while (cin >> n)
41         cout << (n==0 ? 0 : f(n-1)) << endl;
42 }

```

6.5 Criba de Primos

```

1 vector<int> linearPrimeSieve(int n){
2     vector<int> primes;
3     vector<bool> isPrime(n+1, true);
4     for(int i = 2; i <= n; ++i){
5         if(isPrime[i])
6             primes.push_back(i);
7         for(int p : primes){
8             int d = i * p;
9             if(d > n) break;
10            isPrime[d] = false;
11            if(i % p == 0) break;

```

```

12     }
13 }
14 return primes;
15 }

```

6.6 Triangulo de Pascal

```

1 vector<vector<lli>> ncrSieve(int n){
2     vector<vector<lli>> Ncr(n+1);
3     Ncr[0] = {1};
4     for(int i = 1; i <= n; ++i){
5         Ncr[i].resize(i + 1);
6         Ncr[i][0] = Ncr[i][i] = 1;
7         for(int j = 1; j <= i / 2; j++){
8             Ncr[i][i - j] = Ncr[i][j] = Ncr[i - 1][j - 1] + Ncr[i - 1][j];
9         }
10    return Ncr;
11 }

```

6.7 Cambio de bases

```

1 string decimalToBaseB(lli n, lli b){
2     string ans = "";
3     lli d;
4     do{
5         d = n % b;
6         if(0 <= d && d <= 9) ans = (char)(48 + d) + ans;
7         else if(10 <= d && d <= 35) ans = (char)(55 + d) + ans;
8         n /= b;
9     }while(n != 0);
10    return ans;
11 }
12
13 lli baseBtoDecimal(const string & n, lli b){
14     lli ans = 0;
15     for(const char & d : n){
16         if(48 <= d && d <= 57) ans = ans * b + (d - 48);
17         else if(65 <= d && d <= 90) ans = ans * b + (d - 55);
18         else if(97 <= d && d <= 122) ans = ans * b + (d - 87);
19     }
20    return ans;
21 }

```

6.8 Factorizacion

```

1 vector<pair<lli, int>> factorize(lli n){
2     vector<pair<lli, int>> f;
3     for(lli p : primes){
4         if(p * p > n) break;
5         int pot = 0;
6         while(n % p == 0){
7             pot++;
8             n /= p;
9         }
10        if(pot) f.emplace_back(p, pot);
11    }
12    if(n > 1) f.emplace_back(n, 1);
13    return f;
14 }

```

7 Varios

7.1 String a vector int

```

1 //Convertir una cadena de numeros separados por " " en vector de enteros
2 //Leer varias de esas querys
3 cin.ignore();
4 while(q--){
5     string s;
6     getline(cin, s);
7     vector<int> qr;
8     stringstream ss(s);
9     int num;
10    while (ss >> num) qr.push_back(num);
11 }

```

7.2 Generar permutaciones

```

1 //Generar todas las permutaciones de un arreglo
2 sort(all(a));
3 do{
4     //hacer lo que quieras con la perm generada
5 }while(next_permutation(all(a)));

```

7.3 2-Sat

```

1 struct twoSat{
2     int s;
3     vector<vector<int>> g,gr;

```



```

4   vector<int> visited,ids,topologic_sort,val;
5   twoSat(int n){
6       s=n;
7       g.assign(n*2+1,vector<int>());
8       gr.assign(n*2+1,vector<int>());
9       visited.assign(n*2+1,0);
10      ids.assign(n*2+1,0);
11      val.assign(n+1,0);
12  }
13  void addEdge(int a,int b){
14      g[a].push_back(b);
15      gr[b].push_back(a);
16  }
17  void addOr(int a,bool ba,int b,bool bb){
18      addEdge(a+(ba?s:0),b+(bb?0:s));
19      addEdge(b+(bb?s:0),a+(ba?0:s));
20  }
21  void addXor(int a,bool ba,int b,bool bb){
22      addOr(a,ba,b,bb);
23      addOr(a,!ba,b,!bb);
24  }
25  void addAnd(int a,bool ba,int b,bool bb){
26      addXor(a,!ba,b,bb);
27  }
28  void dfs(int u){
29      if(visited[u]!=0) return;
30      visited[u]=1;
31      for(int node:g[u])dfs(node);
32      topologic_sort.push_back(u);
33  }
34  void dfsr(int u,int id){
35      if(visited[u]!=0) return;
36      visited[u]=1;
37      ids[u]=id;
38      for(int node:gr[u])dfsr(node,id);
39  }
40  bool algo(){
41      for(int i=0;i<s*2;i++) if(visited[i]==0) dfs(i);
42      fill(visited.begin(),visited.end(),0);
43      reverse(topologic_sort.begin(),topologic_sort.end());
44      int id=0;
45      for(int i=0;i<topologic_sort.size();i++){
46          if(visited[topologic_sort[i]]==0)dfsr(topologic_sort[i],id

```

```

        ++);
47     }
48     for(int i=0;i<s;i++){
49         if(ids[i]==ids[i+s]) return false;
50         val[i]=(ids[i]>ids[i+s]?0:1);
51     }
52     return true;
53 }
54 };

```

7.4 Bits

```

1 // Return the numbers the numbers of 1-bit in x
2 int __builtin_popcount (unsigned int x)
3 // Returns the number of trailing 0-bits in x. x=0 is undefined.
4 int __builtin_ctz (unsigned int x)
5 // Returns the number of leading 0-bits in x. x=0 is undefined.
6 int __builtin_clz (unsigned int x)
7 // x of type long long just add 'll' at the end of the function.
8 int __builtin_popcountll (unsigned long long x)
9 // Get the value of the least significant bit that is one.
10 v=(x&(-x))

```

7.5 Matrix

```

1 const int N=100, MOD=1e9+7;
2 struct Matrix {
3     ll a[N][N];
4     Matrix() {memset(a,0,sizeof(a));}
5     Matrix operator *(Matrix other) { // Product of a matrix
6         Matrix product=Matrix();
7         rep(i,0,N) rep(j,0,N) rep(k,0,N) {
8             product.a[i][k]+=a[i][j]*other.a[j][k];
9             product.a[i][k]%=MOD;
10        }
11        return product;
12    }
13 };
14 Matrix expo_power(Matrix a, ll n) { // Matrix exponentiation
15     Matrix res=Matrix();
16     rep(i,0,N) res.a[i][i]=1; // Matriz identidad
17     while(n){
18         if(n&1) res=res*a;
19         n>>=1;

```

```

20     a=a*a;
21 }
22 return res;
23 } // Ej. Matrix M=Matrix(); M.a[0][0]=1; M=M*M; Matrix res=
    expo_power(M,k);

```

7.6 Mo's Algorithm

```

1 void remove(idx); // TODO: remove value at idx from data structure
2 void add(idx); // TODO: add value at idx from data structure
3 int get_answer(); // TODO: extract the current answer of the data
    structure
4
5 int block_size;//Recomended sqrt(n)
6
7 struct Query {
8     int l, r, idx;
9     bool operator<(Query other) const
10    {
11        return make_pair(l / block_size, r) <
12            make_pair(other.l / block_size, other.r);
13    }
14 };
15
16 vector<int> mo_s_algorithm(vector<Query> queries) {
17     vector<int> answers(queries.size());
18     sort(queries.begin(), queries.end());
19
20     // TODO: initialize data structure
21
22     int cur_l = 0;
23     int cur_r = -1;
24     // invariant: data structure will always reflect the range [cur_l,
25         cur_r]
26     for (Query q : queries) {
27         while (cur_l > q.l) {
28             cur_l--;
29             add(cur_l);
30         }
31         while (cur_r < q.r) {
32             cur_r++;
33             add(cur_r);
34         }
35     }
36 }

```

```

34 while (cur_l < q.l) {
35     remove(cur_l);
36     cur_l++;
37 }
38 while (cur_r > q.r) {
39     remove(cur_r);
40     cur_r--;
41 }
42 answers[q.idx] = get_answer();
43 }
44 return answers;
45 }

```

7.7 PBS

```

1
2 1.Crear un arreglo con para procesar
3 2.Para cada elemento inicializar 1 l y en q+1 r;
4 for(int i=1;i<=n;i++){
5     m[i].x=1,m[i].y=q+1;
6 }
7 bool flag=true;
8 while(flag){
9     flag=false;
10    // limpiar la estructura de datos
11    for(int i=0;i<=4*n+5;i++)st[i]=0,lazy[i]=0;
12    for(int i=1;i<=n;i++){
13        //Si es diefente l!=r se procesa;
14        if(m[i].x!=m[i].y){ flag=true;tocheck[(m[i].x+m[i].y)/2].
15            push_back(i);}
16        for(int i=1;i<=q;i++){
17            if(!flag)break;
18            // Se aplican las queries
19            update(0,n-1,qs[i].x,qs[i].y,qs[i].z,0);
20            update(0,n-1,qs[i].x,qs[i].x,qs[i].k,0);
21            while(tocheck[i].size()){
22                int id=tocheck[i].back();
23                tocheck[i].pop_back();
24                // Se obserba si se cumblío la caondicion para el
25                elemeto
26                if(ai[id]<=query(0,n-1,S[id],S[id],0)) m[id].y=i;
27                else m[id].x=i+1;
28            }
29        }
30    }
31 }

```

```

27     }
28 }
29 // Solo se imprime
30 for(int i=1;i<=n;i++){
31     if(m[i].x<=q) cout<<m[i].x<<endl;
32     else cout<<-1<<endl;
33 }

```

7.8 Dates

```

1 int dateToInt(int y, int m, int d){
2     return 1461*(y+4800+(m-14)/12)/4+367*(m-2-(m-14)/12*12)/12-
3         3*((y+4900+(m-14)/12)/100)/4+d-32075;
4 }
5 void intToDate(int jd, int& y, int& m, int& d){
6     int x,n,i,j;x=jd+68569;
7     n=4*x/146097;x-=(146097*n+3)/4;
8     i=(4000*(x+1))/1461001;x-=1461*i/4-31;
9     j=80*x/2447;d=x-2447*j/80;
10    x=j/11;m=j+2-12*x;y=100*(n-49)+i+x;
11 }
12 int DayOfWeek(int d, int m, int y){ //starting on Sunday
13     static int ttt[]={0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
14     y-=m<3;
15     return (y+y/4-y/100+y/400+ttt[m-1]+d)%7;
16 }

```

8 Template

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 #define forn(i,n)      for(int i=0; i<n; i++)
5 #define forr(i,a,n)    for(int i=a; i<n; i++)
6 #define fore(i,a,n)    for(int i=a; i<=n; i++)
7 #define each(a,b)      for(auto a: b)
8 #define all(v)          v.begin(),v.end()
9 #define sz(a)           (int)a.size()
10 #define debln(a)        cout << a << "\n"
11 #define deb(a)          cout << a << " "
12 #define pb              push_back
13
14 typedef long long ll;

```

```

15 typedef vector<int> vi;
16 typedef pair<int,int> ii;
17
18 void sol(){
19
20 }
21
22 int main(){
23     ios::sync_with_stdio(false);cin.tie(0);
24
25     int t=1;
26     cin>>t;
27     while(t--){
28         sol();
29     }
30
31     return 0;
32 }

```