

Descongelen a Victor Moreno

Contents

1 Estructuras de Datos	2	6.4 Fibonacci	23
1.1 Unordered Map	2	6.5 Criba de Primos	23
1.2 Segment tree Recursivo	2	6.6 Triangulo de Pascal	23
1.3 Segment Tree Iterativo	2	6.7 Cambio de bases	23
1.4 Segment Tree Lazy Recursivo	3	6.8 Factorizacion	24
1.5 Segment Tree Lazy Iterativo	4	6.9 Factorial mod p	24
1.6 Rope	5		
1.7 Ordered Set	5	7 Varios	24
1.8 Union Find	5	7.1 String a vector<int>	24
1.9 Segment Tree Persistente	5	7.2 Generar permutaciones	24
1.10 Sparce Table	6	7.3 2 Sat	24
1.11 Walvet Tree	6	7.4 Bits	25
1.12 Trie	6	7.5 Matrix	25
1.13 Treap	7	7.6 MO	25
		7.7 PBS	26
2 Strings	8	7.8 Digit DP	26
2.1 Aho Corasick	8		
2.2 Hashing	8	8 Template	27
2.3 KMP	9	8.1 Template	27
2.4 Manacher	9		
2.5 Suffix Automata	9		
3 Graph	10		
4 Flow	16		
4.1 Dinics	16		
4.2 Edmon	16		
5 Geometria	17		
5.1 Puntos y lineas	17		
5.2 Circulos	19		
5.3 Poligonos	20		
6 Matematicas	22		
6.1 Exponenciacion Binaria	22		
6.2 GCD y LCD	22		
6.3 Euclides extendido e inverso modular	22		

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 //Funcion count
22 m1.find(x)!=m1.end()

```

1.2 Segment tree Recursivo

```

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

```

```

14    st[i]=st[i*2+1]+st[i*2+2];
15 }
16 int query(int l, int r, int a, int b, int i){
17     if(a>r||b<l) return 0;
18     if(a<=l&&r<=b) return st[i];
19     int m=l+r>>1;
20     return query(l,m,a,b,2*i+1)+query(m+1,r,a,b,2*i+2);
21 }

```

1.3 Segment Tree Iterativo

```

1 //Para procesar querys de tipo k-esimo es necesario crear un arbol binario
2 //perfector(llevar 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 0(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] = creaNodo(); //Dato compuesto
14        for(int i = N - 1; i > 0; --i)
15            ST[i] = ST[i << 1] + ST[i << 1 | 1]; //Dato normal
16            ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
17    }
18
19    //Actualizacion de un elemento en la posicion i
20    void update(int i, T value){
21        ST[i += N] = value; //Dato normal
22        ST[i += N] = creaNodo(); //Dato compuesto
23        while(i >= 1)
24            ST[i] = ST[i << 1] + ST[i << 1 | 1]; //Dato normal
25            ST[i] = merge(ST[i << 1] , ST[i << 1 | 1]); //Dato compuesto
26    }
27
28    //query en [l, r]
29    T query(int l, int r){
30        T res = 0; //Dato normal
31        nodo resl = creaNodo(), resr = creaNodo(); //Dato compuesto
32        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 siguiente
    potencia de 2
43 //ll nT = 1;
44 // while(nT<n) nT<=1;
45 //vector<int> a(nT,0);
46
47 //Encontrar k-esimo 1 en un st de 1's
48 int Kth_One(int k) {
49     int i = 0, s = N >> 1;
50     for(int p = 2; p < 2 * N; p <= 1, s >= 1) {
51         if(k < ST[p]) continue;
52         k -= ST[p++]; i += s;
53     }
54     return i;
55 }
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);update(m+1,r,a,b,x,2*i+2);
30     st[i]=st[2*i+1]+st[2*i+2];
31 }
32 ll query(int l, int r, int a, int b, int i){
33     if(a>r||b<l)return 0;
34     push(l,r,i);

```

```

35     if(a<=l&& r<=b) return st[i];
36     int m=(l+r)>>1;
37     return query(l,m,a,b,2*i+1)+query(m+1,r,a,b,2*i+2);
38 }

```

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
11         //construccion tenga su respectivo neutro mult y 1
12         ST.resize(N << 1);
13         d.resize(N);
14         h = 64 - __builtin_clzll(n);
15
16         for(int i = 0; i < N; ++i)
17             ST[N + i] = a[i];
18         //Construir el st sobre la query que se necesita
19         for(int i = N - 1; i > 0; --i)
20             ST[i] = min(ST[i << 1] , ST[i << 1 | 1]);
21     }
22
23     //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc
24     void apply(int p, T value) {
25         ST[p] += value;
26         if(p<N) d[p] += value;
27     }
28
29     // Modifica valores de los padres de p
30     //Modificar de acuerdo al tipo modificacion requerida, +,*,|,^,etc y a la
31     //respectiva query
32     void build(int p){
33         while(p>1){
34             p >>= 1;
35             ST[p] = min(ST[p << 1], ST[p << 1 | 1]) + d[p];
36             //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; Requiere 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 si requieres los elementos del union find, utiliza
3 // dsext en caso contrario borrarlo
4 list<int>dsex[ds(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     }
27     int m=(l+r)/2;
28     if(pos<=m) return new node(update(root->l,l,m,pos,val),root->r);
29     return new node(root->l,update(root->r,m+1,r,pos,val));
30 }

```

```

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 Sparce 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+(1<<(i-1))]);
14 }
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 Walvet Tree

```

1 // indexed in 1
2 // from pointer to first element and to to end
3 // x and y The minimun element and y the max element
4 // If you need only one function or more erase the others
5 // If you need tu construct other function you only required to undertand
6 // the limit, this
7 // are the same
8 struct wavelet_tree{
9     int lo, hi;
10    wavelet_tree *l, *r;
11    vector<int> b;
12    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(l-lb, r-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 insertString(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(l)
28         ;
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;}
37     else if (l->weight < r->weight) {
38         merge(l->r, l->r, r); root = l;
39     } else {
40         merge(r->l, l, r->l); root = r;
41     }
42     root->len=1+size(root->l)+size(root->r);
43 }
44 // Not necessary functions indexed in 1
45 void insert(Node *&root,Node *nNode,int pos){
46     Node *l=NULL,*r=NULL,*aux=NULL;
47     split(root,l,r,pos-1);
48     merge(aux,l,nNode);
49     merge(root,aux,r);
50 }
51 void delateRange(Node *&root,int l, int r){
52     Node *l1,*r1,*l2,*r2,*aux2;
53     split(root,l1,r1,l-1);
54     split(r1,r1,r2,r-l+1);
55     merge(root,l1,r2);

```

```

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-1+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-1+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 }

```

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 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 getHashInInterval(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};

```

2.3 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 }

```

2.4 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 }

```

2.5 Suffix Automata

```

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

```

1 struct disjointSet{
2     int N;
3     vector<short int> rank;
4     vi parent, count;
5
6     disjointSet(int N): N(N), parent(N), count(N), rank(N){}
7
8     void makeSet(int v){
9         count[v] = 1;
10        parent[v] = v;
11    }
12
13    int findSet(int v){
14        if(v == parent[v]) return v;
15        return parent[v] = findSet(parent[v]);
16    }
17
18    void unionSet(int a, int b){
19        a = findSet(a), b = findSet(b);
20        if(a == b) return;
21        if(rank[a] < rank[b]){
22            parent[a] = b;
23            count[b] += count[a];
24        }else{
25            parent[b] = a;
26            count[a] += count[b];
27            if(rank[a] == rank[b]) ++rank[a];
28        }
29    }
30 };
31
32 struct edge{
33     int source, dest, cost;

```

```

34     edge(): source(0), dest(0), cost(0){}
35
36     edge(int dest, int cost): dest(dest), cost(cost){}
37
38     edge(int source, int dest, int cost): source(source), dest(dest), cost(
39         cost){}
40
41     bool operator==(const edge & b) const{
42         return source == b.source && dest == b.dest && cost == b.cost;
43     }
44     bool operator<(const edge & b) const{
45         return cost < b.cost;
46     }
47     bool operator>(const edge & b) const{
48         return cost > b.cost;
49     }
50 };
51
52 struct path{
53     int cost = inf;
54     deque<int> vertices;
55     int size = 1;
56     int prev = -1;
57 };
58
59 struct graph{
60     vector<vector<edge>> adjList;
61     vector<vb> adjMatrix;
62     vector<vi> costMatrix;
63     vector<edge> edges;
64     int V = 0;
65     bool dir = false;
66
67     graph(int n, bool dir): V(n), dir(dir), adjList(n), edges(n), adjMatrix(n
68         , vb(n)), costMatrix(n, vi(n)){
69         for(int i = 0; i < n; ++i)
70             for(int j = 0; j < n; ++j)
71                 costMatrix[i][j] = (i == j ? 0 : inf);
72     }
73
74     void add(int source, int dest, int cost){
75         adjList[source].emplace_back(source, dest, cost);

```

```

75 edges.emplace_back(source, dest, cost);
76 adjMatrix[source][dest] = true;
77 costMatrix[source][dest] = cost;
78 if(!dir){
79     adjList[dest].emplace_back(dest, source, cost);
80     adjMatrix[dest][source] = true;
81     costMatrix[dest][source] = cost;
82 }
83 }
84
85 void buildPaths(vector<path> & paths){
86     for(int i = 0; i < V; i++){
87         int u = i;
88         for(int j = 0; j < paths[i].size; j++){
89             paths[i].vertices.push_front(u);
90             u = paths[u].prev;
91         }
92     }
93 }
94
95 vector<path> dijkstra(int start){
96     priority_queue<edge, vector<edge>, greater<edge>> cola;
97     vector<path> paths(V);
98     cola.emplace(start, 0);
99     paths[start].cost = 0;
100     while(!cola.empty()){
101         int u = cola.top().dest; cola.pop();
102         for(edge & current : adjList[u]){
103             int v = current.dest;
104             int nuevo = paths[u].cost + current.cost;
105             if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
106                 paths[v].prev = u;
107                 paths[v].size = paths[u].size + 1;
108             }else if(nuevo < paths[v].cost){
109                 paths[v].prev = u;
110                 paths[v].size = paths[u].size + 1;
111                 cola.emplace(v, nuevo);
112                 paths[v].cost = nuevo;
113             }
114         }
115     }
116     buildPaths(paths);
117     return paths;

```

```

118 }
119
120 vector<path> bellmanFord(int start){
121     vector<path> paths(V, path());
122     vi processed(V);
123     vb inQueue(V);
124     queue<int> Q;
125     paths[start].cost = 0;
126     Q.push(start);
127     while(!Q.empty()){
128         int u = Q.front(); Q.pop(); inQueue[u] = false;
129         if(paths[u].cost == inf) continue;
130         ++processed[u];
131         if(processed[u] == V){
132             cout << "Negative cycle\n";
133             return {};
134         }
135         for(edge & current : adjList[u]){
136             int v = current.dest;
137             int nuevo = paths[u].cost + current.cost;
138             if(nuevo == paths[v].cost && paths[u].size + 1 < paths[v].size){
139                 paths[v].prev = u;
140                 paths[v].size = paths[u].size + 1;
141             }else if(nuevo < paths[v].cost){
142                 if(!inQueue[v]){
143                     Q.push(v);
144                     inQueue[v] = true;
145                 }
146                 paths[v].prev = u;
147                 paths[v].size = paths[u].size + 1;
148                 paths[v].cost = nuevo;
149             }
150         }
151     }
152     buildPaths(paths);
153     return paths;
154 }
155
156
157
158
159
160 vector<vi> floyd(){

```

```

161     vector<vi> tmp = costMatrix;
162     for(int k = 0; k < V; ++k)
163         for(int i = 0; i < V; ++i)
164             for(int j = 0; j < V; ++j)
165                 if(tmp[i][k] != inf && tmp[k][j] != inf)
166                     tmp[i][j] = min(tmp[i][j], tmp[i][k] + tmp[k][j]);
167     return tmp;
168 }
169
170 vector<vb> transitiveClosure(){
171     vector<vb> tmp = adjMatrix;
172     for(int k = 0; k < V; ++k)
173         for(int i = 0; i < V; ++i)
174             for(int j = 0; j < V; ++j)
175                 tmp[i][j] = tmp[i][j] || (tmp[i][k] && tmp[k][j]);
176     return tmp;
177 }
178
179 vector<vb> transitiveClosureDFS(){
180     vector<vb> tmp(V, vb(V));
181     function<void(int, int)> dfs = [&](int start, int u){
182         for(edge & current : adjList[u]){
183             int v = current.dest;
184             if(!tmp[start][v]){
185                 tmp[start][v] = true;
186                 dfs(start, v);
187             }
188         }
189     };
190     for(int u = 0; u < V; u++){
191         dfs(u, u);
192     }
193     return tmp;
194 }
195
196 bool isBipartite(){
197     vi side(V, -1);
198     queue<int> q;
199     for (int st = 0; st < V; ++st){
200         if(side[st] != -1) continue;
201         q.push(st);
202         side[st] = 0;
203         while(!q.empty()){
204             int u = q.front();

```

```

204             q.pop();
205             for (edge & current : adjList[u]){
206                 int v = current.dest;
207                 if(side[v] == -1) {
208                     side[v] = side[u] ^ 1;
209                     q.push(v);
210                 }else{
211                     if(side[v] == side[u]) return false;
212                 }
213             }
214         }
215     }
216     return true;
217 }
218
219 vi topologicalSort(){
220     int visited = 0;
221     vi order, indegree(V);
222     for(auto & node : adjList){
223         for(edge & current : node){
224             int v = current.dest;
225             ++indegree[v];
226         }
227     }
228     queue<int> Q;
229     for(int i = 0; i < V; ++i){
230         if(indegree[i] == 0) Q.push(i);
231     }
232     while(!Q.empty()){
233         int source = Q.front();
234         Q.pop();
235         order.push_back(source);
236         ++visited;
237         for(edge & current : adjList[source]){
238             int v = current.dest;
239             --indegree[v];
240             if(indegree[v] == 0) Q.push(v);
241         }
242     }
243     if(visited == V) return order;
244     else return {};
245 }
246

```

```

247 bool hasCycle(){
248     vi color(V);
249     function<bool(int, int)> dfs = [&](int u, int parent){
250         color[u] = 1;
251         bool ans = false;
252         int ret = 0;
253         for(edge & current : adjList[u]){
254             int v = current.dest;
255             if(color[v] == 0)
256                 ans |= dfs(v, u);
257             else if(color[v] == 1 && (dir || v != parent || ret++))
258                 ans = true;
259         }
260         color[u] = 2;
261         return ans;
262     };
263     for(int u = 0; u < V; ++u)
264         if(color[u] == 0 && dfs(u, -1))
265             return true;
266     return false;
267 }
268
269 pair<vb, vector<edge>> articulationBridges(){
270     vi low(V), label(V);
271     vb points(V);
272     vector<edge> bridges;
273     int time = 0;
274     function<int(int, int)> dfs = [&](int u, int p){
275         label[u] = low[u] = ++time;
276         int hijos = 0, ret = 0;
277         for(edge & current : adjList[u]){
278             int v = current.dest;
279             if(v == p && !ret++) continue;
280             if(!label[v]){
281                 ++hijos;
282                 dfs(v, u);
283                 if(label[u] <= low[v])
284                     points[u] = true;
285                 if(label[u] < low[v])
286                     bridges.push_back(current);
287                 low[u] = min(low[u], low[v]);
288             }
289             low[u] = min(low[u], label[v]);

```

```

290         }
291         return hijos;
292     };
293     for(int u = 0; u < V; ++u)
294         if(!label[u])
295             points[u] = dfs(u, -1) > 1;
296     return make_pair(points, bridges);
297 }
298
299 vector<vi> scc(){
300     vi low(V), label(V);
301     int time = 0;
302     vector<vi> ans;
303     stack<int> S;
304     function<void(int)> dfs = [&](int u){
305         label[u] = low[u] = ++time;
306         S.push(u);
307         for(edge & current : adjList[u]){
308             int v = current.dest;
309             if(!label[v]) dfs(v);
310             low[u] = min(low[u], low[v]);
311         }
312         if(label[u] == low[u]){
313             vi comp;
314             while(S.top() != u){
315                 comp.push_back(S.top());
316                 low[S.top()] = V + 1;
317                 S.pop();
318             }
319             comp.push_back(S.top());
320             S.pop();
321             ans.push_back(comp);
322             low[u] = V + 1;
323         }
324     };
325     for(int u = 0; u < V; ++u)
326         if(!label[u]) dfs(u);
327     return ans;
328 }
329
330 vector<edge> kruskal(){
331     sort(edges.begin(), edges.end());
332     vector<edge> MST;

```

```

333 disjointSet DS(V);
334 for(int u = 0; u < V; ++u)
335     DS.makeSet(u);
336 int i = 0;
337 while(i < edges.size() && MST.size() < V - 1){
338     edge current = edges[i++];
339     int u = current.source, v = current.dest;
340     if(DS.findSet(u) != DS.findSet(v)){
341         MST.push_back(current);
342         DS.unionSet(u, v);
343     }
344 }
345 return MST;
346 }
347
348 bool tryKuhn(int u, vb & used, vi & left, vi & right){
349     if(used[u]) return false;
350     used[u] = true;
351     for(edge & current : adjList[u]){
352         int v = current.dest;
353         if(right[v] == -1 || tryKuhn(right[v], used, left, right)){
354             right[v] = u;
355             left[u] = v;
356             return true;
357         }
358     }
359     return false;
360 }
361
362 bool augmentingPath(int u, vb & used, vi & left, vi & right){
363     used[u] = true;
364     for(edge & current : adjList[u]){
365         int v = current.dest;
366         if(right[v] == -1){
367             right[v] = u;
368             left[u] = v;
369             return true;
370         }
371     }
372     for(edge & current : adjList[u]){
373         int v = current.dest;
374         if(!used[right[v]] && augmentingPath(right[v], used, left, right)){
375             right[v] = u;

```

```

376         left[u] = v;
377         return true;
378     }
379 }
380 return false;
381 }
382
383 //vertices from the left side numbered from 0 to l-1
384 //vertices from the right side numbered from 0 to r-1
385 //graph[u] represents the left side
386 //graph[u][v] represents the right side
387 //we can use tryKuhn() or augmentingPath()
388 vector<pair<int, int>> maxMatching(int l, int r){
389     vi left(l, -1), right(r, -1);
390     vb used(l);
391     for(int u = 0; u < l; ++u){
392         tryKuhn(u, used, left, right);
393         fill(used.begin(), used.end(), false);
394     }
395     vector<pair<int, int>> ans;
396     for(int u = 0; u < r; ++u){
397         if(right[u] != -1){
398             ans.emplace_back(right[u], u);
399         }
400     }
401     return ans;
402 }
403
404 void dfs(int u, vi & status, vi & parent){
405     status[u] = 1;
406     for(edge & current : adjList[u]){
407         int v = current.dest;
408         if(status[v] == 0){ //not visited
409             parent[v] = u;
410             dfs(v, status, parent);
411         }else if(status[v] == 1){ //explored
412             if(v == parent[u]){
413                 //bidirectional node u<-->v
414             }else{
415                 //back edge u-v
416             }
417         }else if(status[v] == 2){ //visited
418             //forward edge u-v

```

```

419     }
420     }
421     status[u] = 2;
422 }
423 };
424
425 struct tree{
426     vi parent, level, weight;
427     vector<vi> dists, DP;
428     int n, root;
429
430     void dfs(int u, graph & G){
431         for(edge & curr : G.adjList[u]){
432             int v = curr.dest;
433             int w = curr.cost;
434             if(v != parent[u]){
435                 parent[v] = u;
436                 weight[v] = w;
437                 level[v] = level[u] + 1;
438                 dfs(v, G);
439             }
440         }
441     }
442
443     tree(int n, int root): n(n), root(root), parent(n), level(n), weight(n),
444         dists(n, vi(20)), DP(n, vi(20)){
445         parent[root] = root;
446     }
447
448     tree(graph & G, int root): n(G.V), root(root), parent(G.V), level(G.V),
449         weight(G.V), dists(G.V, vi(20)), DP(G.V, vi(20)){
450         parent[root] = root;
451         dfs(root, G);
452     }
453
454     void pre(){
455         for(int u = 0; u < n; u++){
456             DP[u][0] = parent[u];
457             dists[u][0] = weight[u];
458         }
459         for(int i = 1; (1 << i) <= n; ++i){
460             for(int u = 0; u < n; ++u){
461                 DP[u][i] = DP[DP[u][i - 1]][i - 1];

```

```

460         dists[u][i] = dists[u][i - 1] + dists[DP[u][i - 1]][i - 1];
461     }
462 }
463 }
464
465 int ancestor(int p, int k){
466     int h = level[p] - k;
467     if(h < 0) return -1;
468     int lg;
469     for(lg = 1; (1 << lg) <= level[p]; ++lg);
470     lg--;
471     for(int i = lg; i >= 0; --i){
472         if(level[p] - (1 << i) >= h){
473             p = DP[p][i];
474         }
475     }
476     return p;
477 }
478
479 int lca(int p, int q){
480     if(level[p] < level[q]) swap(p, q);
481     int lg;
482     for(lg = 1; (1 << lg) <= level[p]; ++lg);
483     lg--;
484     for(int i = lg; i >= 0; --i){
485         if(level[p] - (1 << i) >= level[q]){
486             p = DP[p][i];
487         }
488     }
489     if(p == q) return p;
490
491     for(int i = lg; i >= 0; --i){
492         if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
493             p = DP[p][i];
494             q = DP[q][i];
495         }
496     }
497     return parent[p];
498 }
499
500 int dist(int p, int q){
501     if(level[p] < level[q]) swap(p, q);
502     int lg;

```

```

503     for(lg = 1; (1 << lg) <= level[p]; ++lg);
504     lg--;
505     int sum = 0;
506     for(int i = lg; i >= 0; --i){
507         if(level[p] - (1 << i) >= level[q]){
508             sum += dists[p][i];
509             p = DP[p][i];
510         }
511     }
512     if(p == q) return sum;
513
514     for(int i = lg; i >= 0; --i){
515         if(DP[p][i] != -1 && DP[p][i] != DP[q][i]){
516             sum += dists[p][i] + dists[q][i];
517             p = DP[p][i];
518             q = DP[q][i];
519         }
520     }
521     sum += dists[p][0] + dists[q][0];
522     return sum;
523 }
524 };

```

4 Flow

4.1 Dinics

```

1 struct Dinic{
2     int nodes,src,dst;
3     vector<int> dist,q,work;
4     struct edge {int to,rev;ll f,cap;};
5     vector<vector<edge>> g;
6     Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
7     void add_edge(int s, int t, ll cap){
8         g[s].pb((edge){t,SZ(g[t]),0,cap});
9         g[t].pb((edge){s,SZ(g[s])-1,0,0});
10    }
11    bool dinic_bfs(){
12        fill(ALL(dist),-1);dist[src]=0;
13        int qt=0;q[qt++]=src;
14        for(int qh=0;qh<qt;qh++){
15            int u=q[qh];
16            fore(i,0,SZ(g[u])){

```

```

17                edge &e=g[u][i];int v=g[u][i].to;
18                if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
19            }
20        }
21        return dist[dst]>=0;
22    }
23    ll dinic_dfs(int u, ll f){
24        if(u==dst)return f;
25        for(int &i=work[u];i<SZ(g[u]);i++){
26            edge &e=g[u][i];
27            if(e.cap<=e.f)continue;
28            int v=e.to;
29            if(dist[v]==dist[u]+1){
30                ll df=dinic_dfs(v,min(f,e.cap-e.f));
31                if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
32            }
33        }
34        return 0;
35    }
36    ll max_flow(int _src, int _dst){
37        src=_src;dst=_dst;
38        ll result=0;
39        while(dinic_bfs()){
40            fill(ALL(work),0);
41            while(ll delta=dinic_dfs(src,INF))result+=delta;
42        }
43        return result;
44    }
45 };

```

4.2 Edmon

```

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                 flow=get<2>(edges[idx]);
29             if(cap-flow>0 && d[v]==-1) d[v]=d[u]+1,cam[v]={u,idx},q.
30                 push(v);
31         }
32     }
33     return d[t]!=-1;
34 }
35 ll maxFlow(int s,int t){
36     ll flow=0;
37     while(bfs(s,t)){
38         ll push=sentFlow(s,t,1e18);
39         if(!push) return flow;
40         flow+=push;
41     }
42     return flow;
43 }
44 void addEdge(int u,int v, ll c, bool dire=true){
45     if(u==v) return;
46     edges.emplace_back(v,c,0);
47     adj[u].push_back(edges.size()-1);
48     edges.emplace_back(u,(dire?0:c),0);
49     adj[v].push_back(edges.size()-1);
50 }

```

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     point operator-(const point & p) const{return point(x - p.x, y - p.y);}
18     point operator*(const ld & k) const{return point(x * k, y * k);}
19     point operator/(const ld & k) const{return point(x / k, y / k);}
20
21     point operator+=(const point & p){*this = *this + p; return *this;}
22     point operator-=(const point & p){*this = *this - p; return *this;}
23     point operator*=(const ld & p){*this = *this * p; return *this;}
24     point operator/=(const ld & p){*this = *this / p; return *this;}
25
26     point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*sin(
27         a) + y*cos(a));}
28     point perp() const{return point(-y, x);}
29     ld ang() const{
30         ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
31     }
32     ld dot(const point & p) const{return x * p.x + y * p.y;}
33     ld cross(const point & p) const{return x * p.y - y * p.x;}
34     ld norm() const{return x * x + y * y;}
35     ld length() const{return sqrtl(x * x + y * y);}
36     point unit() const{return (*this) / length();}
37
38     bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y);}
39     bool operator!=(const point & p) const{return !(*this == p);}

```

```

39 bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x) &&
    le(y, p.y));}
40 bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x) &&
    ge(y, p.y));}
41 bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p.
    cross(*this), 0) && le(p.dot(*this), 0));}
42 };
43
44 istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
45 ostream &operator<<(ostream &os, const point & p){return os << "(" << p.x
    << ", " << p.y << ")};
46
47 int sgn(ld x){
48     if(ge(x, 0)) return 1;
49     if(le(x, 0)) return -1;
50     return 0;
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).cross
            (b - o));
57     });
58 }
59
60 bool pointInLine(const point & a, const point & v, const point & p){
61     //line a+tv, point p
62     return eq((p - a).cross(v), 0);
63 }
64
65 bool pointInSegment(const point & a, const point & b, const point & p){
66     //segment ab, point p
67     return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
68 }
69
70 int intersectLinesInfo(const point & a1, const point & v1, const point & a2
    , const point & v2){
71     //lines a1+tv1 and a2+tv2
72     ld det = v1.cross(v2);
73     if(eq(det, 0)){
74         if(eq((a2 - a1).cross(v1), 0)){
75             return -1; //infinity points

```

```

76     }else{
77         return 0; //no points
78     }
79 }else{
80     return 1; //single point
81 }
82 }
83
84 point intersectLines(const point & a1, const point & v1, const point & a2,
    const point & v2){
85     //lines a1+tv1, a2+tv2
86     //assuming that they intersect
87     ld det = v1.cross(v2);
88     return a1 + v1 * ((a2 - a1).cross(v2) / det);
89 }
90
91 int intersectLineSegmentInfo(const point & a, const point & v, const point
    & c, const point & d){
92     //line a+tv, segment cd
93     point v2 = d - c;
94     ld det = v.cross(v2);
95     if(eq(det, 0)){
96         if(eq((c - a).cross(v), 0)){
97             return -1; //infinity points
98         }else{
99             return 0; //no point
100         }
101     }else{
102         return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point,
            0: no point
103     }
104 }
105
106 int intersectSegmentsInfo(const point & a, const point & b, const point & c
    , const point & d){
107     //segment ab, segment cd
108     point v1 = b - a, v2 = d - c;
109     int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
110     if(t == u){
111         if(t == 0){
112             if(pointInSegment(a, b, c) || pointInSegment(a, b, d) ||
                pointInSegment(c, d, a) || pointInSegment(c, d, b)){
113                 return -1; //infinity points

```

```

114     }else{
115         return 0; //no point
116     }
117     }else{
118         return 0; //no point
119     }
120     }else{
121         return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single point,
122         0: no point
123     }
124 }
125 ld distancePointLine(const point & a, const point & v, const point & p){
126     //line: a + tv, point p
127     return abs(v.cross(p - a)) / v.length();
128 }

```

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 & p)
12 {
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, const
21 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();

```

```

23     if(eq(h2, 0)) return {p}; //line tangent to circle
24     else if(1e(h2, 0)) return {}; //no intersection
25     else{
26         point u = v.unit() * sqrt(h2);
27         return {p - u, p + u}; //two points of intersection (chord)
28     }
29 }
30
31 vector<point> intersectSegmentCircle(const point & a, const point & b,
32 const point & c, ld r){
33     //segment ab, circle with center c and radius r
34     vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
35     for(const point & p : P){
36         if(pointInSegment(a, b, p)) ans.push_back(p);
37     }
38     return ans;
39 }
40
41 pair<point, ld> getCircle(const point & m, const point & n, const point & p
42 ){
43     //find circle that passes through points p, q, r
44     point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p - n)
45         .perp());
46     ld r = (c - m).length();
47     return {c, r};
48 }
49
50 vector<point> intersectionCircles(const point & c1, ld r1, const point & c2
51 , ld r2){
52     //circle 1 with center c1 and radius r1
53     //circle 2 with center c2 and radius r2
54     point d = c2 - c1;
55     ld d2 = d.norm();
56     if(eq(d2, 0)) return {}; //concentric circles
57     ld pd = (d2 + r1*r1 - r2*r2) / 2;
58     ld h2 = r1*r1 - pd*pd/d2;
59     point p = c1 + d*pd/d2;
60     if(eq(h2, 0)) return {p}; //circles touch at one point
61     else if(1e(h2, 0)) return {}; //circles don't intersect
62     else{
63         point u = d.perp() * sqrt(h2/d2);
64         return {p - u, p + u};
65     }

```

```

62 }
63
64 int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2){
65     //test if circle 2 is inside circle 1
66     //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if
        they overlap
67     ld l = r1 - r2 - (c1 - c2).length();
68     return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
69 }
70
71 int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2){
72     //test if circle 2 is outside circle 1
73     //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if
        they overlap
74     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() - 1] - P[i]), 0)){
24             L.pop_back();
25         }
26         L.push_back(P[i]);
27     }
28     for(int i = P.size() - 1; i >= 0; i--){
29         while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size() - 1] - P[i]), 0)){
30             U.pop_back();
31         }
32         U.push_back(P[i]);
33     }
34     L.pop_back();
35     U.pop_back();
36     L.insert(L.end(), U.begin(), U.end());
37     return L;
38 }
39
40 bool pointInPerimeter(const vector<point> & P, const point & p){
41     int n = P.size();
42     for(int i = 0; i < n; i++){
43         if(pointInSegment(P[i], P[(i + 1) % n], p)){
44             return true;
45         }

```

```

46     }
47     return false;
48 }
49
50 bool crossesRay(const point & a, const point & b, const point & p){
51     return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) > 0;
52 }
53
54 int pointInPolygon(const vector<point> & P, const point & p){
55     if(pointInPerimeter(P, p)){
56         return -1; //point in the perimeter
57     }
58     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[0])
            ) - 1;
86         if(det == -2) det = 1;

```

```

87     return det;
88 }
89 }
90
91 vector<point> cutPolygon(const vector<point> & P, const point & a, const
    point & v){
92     //returns the part of the convex polygon P on the left side of line a+tv
93     int n = P.size();
94     vector<point> lhs;
95     for(int i = 0; i < n; ++i){
96         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 }

```

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 }

```

6.9 Factorial mod p

```

1  int factmod(int n, int p) {
2      vector<int> f(p);
3      f[0] = 1;
4      for (int i = 1; i < p; i++)
5          f[i] = f[i-1] * i % p;
6  }

```

```

7   int res = 1;
8   while (n > 1) {
9       if ((n/p) % 2)
10           res = p - res;
11       res = res * f[n%p] % p;
12       n /= p;
13   }
14   return res;
15 }

```

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 |__builtin_popcount(maks) // Count the numbers of on bits

```

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

```

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         while (cur_l < q.l) {
36             remove(cur_l);
37             cur_l++;
38         }
39         while (cur_r > q.r) {
40             remove(cur_r);
41             cur_r--;
42         }
43         answers[q.idx] = get_answer();
44     }
45     return answers;
46 }

```

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 elemeto
25                if(ai[id]<=query(0,n-1,S[id],S[id],0)) m[id].y=i;
26                else m[id].x=i+1;
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 Digit DP

```

1 res = solve(b) - solve(a-1);
2 vector<int>num;
3 int dp[20][20][2];
4
5 int solve(lli b){
6     num.clear();

```

```

7   while(b>0){
8       num.push_back(b%10);
9       b/=10;
10  }
11  reverse(num.begin(), num.end());
12
13  memset(dp, -1, sizeof(dp));
14  lli res = mem(0, 0, 0);
15  return res;
16 }
17
18 //Numeros con a los mas 3 digitos distintos de cero
19 //4, 200000, 10203
20 int mem(int pos, int cant, int goodAll){
21     if(cant>3) return 0;
22     if(pos==num.size()){
23         if(cant<=3) return 1;
24         return 0;
25     }
26
27     int &a = dp[pos][cant][goodAll];
28     if(a!=-1) return a;
29     a = 0;
30
31     int limite = goodAll==0?num[pos]:9;
32     fore(dig,0,limite){
33         int nG = goodAll;
34         int nCant = cant;
35         if(goodAll==0 && dig<limite)    nG=1;
36         if(dig!=0)    nCant++;
37         if(nCant<=3)    a+=mem(pos+1,nCant,nG);
38     }
39
40     return a;
41 }
42
43 //Numeros donde el digito d ocurre exactamente k veces
44 int call(int pos, int cnt, int f){
45     if(cnt > k) return 0;
46
47     if(pos == num.size()){
48         if(cnt == k)    return 1;
49         return 0;

```

```

50 }
51
52 if(DP[pos][cnt][f] != -1)    return DP[pos][cnt][f];
53 int res = 0;
54
55 int LMT;
56
57 if(f == 0)    LMT = num[pos];
58 else        LMT = 9;
59
60 for(int dgt = 0; dgt<=LMT; dgt++){
61     int nf = f;
62     int ncnt = cnt;
63     if(f == 0 && dgt < LMT) nf = 1;
64     if(dgt == d)    ncnt++;
65     if(ncnt <= k)    res += call(pos+1, ncnt, nf);
66 }
67
68 DP[pos][cnt][f] = res;
69 return DP[pos][cnt][f];
70 }

```

8 Template

8.1 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 }
```