

1 Data-Structure

1.1 Treap

```

1 mt19937 mt(hash<string>())("Treap");
2 template<class T>struct Treap{
3     struct node{
4         node *l = NULL,*r = NULL;
5         T key;
6         int pri = mt(),sz = 1;
7         bool rev = 0;
8         node(T x):key(x){}
9         ~node(){
10             for(auto &i:{l,r})
11                 delete i;
12         }
13         void push(){
14             if(!rev)return;
15             swap(l,r);
16             for(auto &i:{l,r})
17                 if(i)i->rev^=1;
18             rev = 0;
19         }
20         void pull(){
21             sz = 1;
22             for(auto i:{l,r})
23                 if(i)sz+=i->sz;
24         }
25     };
26     node *root = NULL;
27     int size(node *a){
28         return a?a->sz:0;
29     }
30     node *merge(node *a,node *b){
31         if(!a or !b)return a?:b;
32         if(a->pri>b->pri){
33             a->push();
34             a->r = merge(a->r,b);
35             a->pull();
36             return a;
37         }
38         else{
39             b->push();
40             b->l = merge(a,b->l);
41             b->pull();
42             return b;
43         }
44     }
45     void split(node *t,int k,node *&a,node *&b){
46         if(!t){a = b = NULL;return;}
47         t->push();
48         if(size(t->l)+1<=k){
49             a = t;
50             split(t->r,k-size(t->l)-1,a->r,b);
51             a->pull();
52         }
53         else{
54             b = t;
55             split(t->l,k,a,b->l);
56             b->pull();
57         }
58     }
59     void split_by_key(node *t,T k,node *&a,node *&b){
60         if(!t){a = b = NULL;return;}
61         t->push();
62         if(t->key<=k){
63             a = t;
64             split_by_key(t->r,k,a->r,b);
65             a->pull();
66         }
67         else{
68             b = t;
69             split_by_key(t->l,k,a,b->l);
70             b->pull();
71         }
72     }
73     void push_back(T x){
74         root = merge(root,new node(x));
75     }
76     void push_front(T x){
77         root = merge(new node(x),root);
78     }
79     void erase(int l,int r){
80         node *a,*b,*c;
81         split(root,l,a,b);
82         split(b,r-l+1,b,c);
83         delete b;
84         root = merge(a,c);
85     }
86     void insert(int idx,T k){
87         node *a,*b;
88         split(root,idx,a,b);
89         root = merge(a,merge(new node(k),b));
90     }

```

```

91     T operator [](int x){
92         node *a,*b,*c;
93         split(root,x,a,b);
94         split(b,1,b,c);
95         root = merge(a,merge(b,c));
96         return b->key;
97     }
98     void reverse(int l,int r){
99         node *a,*b,*c;
100         split(root,l,a,b);
101         split(b,r-l+1,b,c);
102         b->rev^=1;
103         root = merge(a,merge(b,c));
104     }
105 };

```

1.2 Segtree

```

1 template<class S,
2         S (*node_pull)(S, S),
3         S (*node_init)(),
4         class T,
5         S (*mapping)(S, T),
6         T (*tag_pull)(T, T),
7         T (*tag_init)()>
8 struct segment_tree{
9     struct node{
10         S seg;
11         T tag = tag_init();
12         int l,r;
13         node(S _seg = node_init(),int _l = -1,int _r = -1) : seg(
14             _seg), l(_l), r(_r){}
15         friend node operator +(const node &lhs,const node &rhs){
16             if(lhs.l==1)return rhs;
17             if(rhs.l==1)return lhs;
18             return node(node_pull(lhs.seg,rhs.seg),lhs.l,rhs.r);
19         };
20     };
21     vector<node>arr;
22     void all_apply(int idx,T t){
23         arr[idx].seg = mapping(arr[idx].seg, t);
24         arr[idx].tag = tag_pull(arr[idx].tag, t);
25     }
26     void push(int idx){
27         all_apply(idx<<1, arr[idx].tag);
28         all_apply(idx<<1|1, arr[idx].tag);
29         arr[idx].tag = tag_init();
30     }
31     inline void build(const vector<S> &v,const int &l,const int &
32         r,int idx = 1){
33         if(idx==1)arr.resize((r-l+1)<<2);
34         if(l==r){
35             arr[idx].seg = v[l];
36             arr[idx].tag = tag_init();
37             arr[idx].l = arr[idx].r = l;
38             return;
39         }
40         int m = (l+r)>>1;
41         build(v,l,m,idx<<1);
42         build(v,m+1,r,idx<<1|1);
43         arr[idx] = arr[idx<<1]+arr[idx<<1|1];
44     }
45     inline void update(const int &ql,const int &qr,T t,int idx =
46         1){
47         assert(ql<=qr);
48         if(ql<=arr[idx].l and arr[idx].r<=qr){
49             all_apply(idx, t);
50             return;
51         }
52         push(idx);
53         int m = (arr[idx].l+arr[idx].r)>>1;
54         if(ql<=m)update(ql,qr,t,idx<<1);
55         if(qr>m)update(ql,qr,t,idx<<1|1);
56         arr[idx] = arr[idx<<1]+arr[idx<<1|1];
57     }
58     inline S query(const int &ql,const int &qr,int idx = 1){
59         assert(ql<=qr);
60         if(ql<=arr[idx].l and arr[idx].r<=qr){
61             return arr[idx].seg;
62         }
63         push(idx);
64         int m = (arr[idx].l+arr[idx].r)>>1;
65         S ans = node_init(),lhs = node_init(),rhs = node_init();
66         if(ql<=m)lhs = query(ql,qr,idx<<1);
67         if(qr>m)rhs = query(ql,qr,idx<<1|1);
68         ans = node_pull(lhs,rhs);
69         return ans;
70     }
71 };

```

1.3 DsuUndo

```

1 struct dsu_undo{
2     vector<int>sz,p;
3     int comps;
4     dsu_undo(int n){
5         sz.assign(n+5,1);
6         p.resize(n+5);
7         for(int i = 1;i<=n;++i)p[i] = i;
8         comps = n;
9     }
10    vector<pair<int,int>>opt;
11    int Find(int x){
12        return x==p[x]?x:Find(p[x]);
13    }
14    bool Union(int a,int b){
15        int pa = Find(a),pb = Find(b);
16        if(pa==pb)return 0;
17        if(sz[pa]<sz[pb])swap(pa,pb);
18        sz[pa]+=sz[pb];
19        p[pb] = pa;
20        opt.push_back({pa,pb});
21        comps--;
22        return 1;
23    }
24    void undo(){
25        auto [pa,pb] = opt.back();
26        opt.pop_back();
27        p[pb] = pb;
28        sz[pa]-=sz[pb];
29        comps++;
30    }
31 };

```

1.4 DSU

```

1 struct DSU{
2     vector<int>sz;
3     int n;
4     DSU(int _n):n(_n){
5         sz.assign(n+1,-1);
6     }
7     int Find(int x){
8         return sz[x]<0?x:sz[x] = Find(sz[x]);
9     }
10    bool Union(int a,int b){
11        int pa = Find(a),pb = Find(b);
12        if(pa==pb)return 0;
13        if((-sz[pa])<(-sz[pb]))swap(pa,pb);
14        sz[pa]+=sz[pb];
15        sz[pb] = pa;
16        return 1;
17    }
18 };

```

1.5 Fenwick

```

1 template<class T>struct fenwick_tree{
2     int n;
3     vector<T>arr;
4     inline int lowbit(int x){
5         return x&(-x);
6     }
7     fenwick_tree(int _n) : n(_n){
8         arr.assign(n+5,0);
9     }
10    T query(int x){
11        T ans = 0;
12        for(int i = x;i>0;i-=lowbit(i)){
13            ans+=arr[i];
14        }
15        return ans;
16    }
17    void update(int x,T y){
18        for(int i = x;i<=n;i+=lowbit(i)){
19            arr[i]+=y;
20        }
21    }
22 };

```

1.6 Persistent DSU

```

1 int rk[200001] = {};
2 struct Persistent_DSU{
3     rope<int>*p;
4     int n;
5     Persistent_DSU(int _n = 0):n(_n){
6         if(n==0)return;
7         p = new rope<int>;

```

```

8         int tmp[n+1] = {};
9         for(int i = 1;i<=n;++i)tmp[i] = i;
10        p->append(tmp,n+1);
11    }
12    Persistent_DSU(const Persistent_DSU &tmp){
13        p = new rope<int>(*tmp.p);
14        n = tmp.n;
15    }
16    int Find(int x){
17        int px = p->at(x);
18        return px==x?x:Find(px);
19    }
20    bool Union(int a,int b){
21        int pa = Find(a),pb = Find(b);
22        if(pa==pb)return 0;
23        if(rk[pa]<rk[pb])swap(pa,pb);
24        p->replace(pb,pa);
25        if(rk[pa]==rk[pb])rk[pa]++;
26        return 1;
27    }
28 };

```

1.7 TimingSegtree

```

1 template<class T,class D>struct timing_segment_tree{
2     struct node{
3         int l,r;
4         vector<T>opt;
5     };
6     vector<node>arr;
7     void build(int l,int r,int idx = 1){
8         if(idx==1)arr.resize((r-l+1)<<2);
9         if(l==r){
10            arr[idx].l = arr[idx].r = l;
11            arr[idx].opt.clear();
12            return;
13        }
14        int m = (l+r)>>1;
15        build(l,m,idx<<1);
16        build(m+1,r,idx<<1|1);
17        arr[idx].l = l,arr[idx].r = r;
18        arr[idx].opt.clear();
19    }
20    void update(int ql,int qr,T k,int idx = 1){
21        if(ql<=arr[idx].l and arr[idx].r<=qr){
22            arr[idx].opt.push_back(k);
23            return;
24        }
25        int m = (arr[idx].l+arr[idx].r)>>1;
26        if(ql<=m)update(ql,qr,k,idx<<1);
27        if(qr>m)update(ql,qr,k,idx<<1|1);
28    }
29    void dfs(D &d,vector<int>&ans,int idx = 1){
30        int cnt = 0;
31        for(auto [a,b]:arr[idx].opt){
32            if(d.Union(a,b))cnt++;
33        }
34        if(arr[idx].l==arr[idx].r)ans[arr[idx].l] = d.comps;
35        else{
36            dfs(d,ans,idx<<1);
37            dfs(d,ans,idx<<1|1);
38        }
39        while(cnt-->0)d.undo();
40    }
41 };

```

1.8 AreaOfRectangles

```

1 long long AreaOfRectangles(vector<tuple<int,int,int,int>>>v){
2     vector<tuple<int,int,int,int>>tmp;
3     int L = INT_MAX,R = INT_MIN;
4     for(auto [x1,y1,x2,y2]:v){
5         tmp.push_back({x1,y1+1,y2,1});
6         tmp.push_back({x2,y1+1,y2,-1});
7         R = max(R,y2);
8         L = min(L,y1);
9     }
10    vector<long long>seg((R-L+1)<<2),tag((R-L+1)<<2);
11    sort(tmp.begin(),tmp.end());
12    function<void(int,int,int,int,int,int)>update = [&](int ql,
13        int qr,int val,int l,int r,int idx){
14        if(ql<=l and r<=qr){
15            tag[idx]+=val;
16            if(tag[idx])seg[idx] = r-l+1;
17            else if(l==r)seg[idx] = 0;
18            else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
19            return;
20        }
21        int m = (l+r)>>1;
22        if(ql<=m)update(ql,qr,val,l,m,idx<<1);
23        if(qr>m)update(ql,qr,val,m+1,r,idx<<1|1);

```

```

23     if(tag[idx])seg[idx] = r-1+1;
24     else seg[idx] = seg[idx<<1]+seg[idx<<1|1];
25 };
26 long long last_pos = 0,ans = 0;
27 for(auto [pos,l,r,val]:tmp){
28     ans+=(pos-last_pos)*seg[l];
29     update(l,r,val,L,R,1);
30     last_pos = pos;
31 }
32 return ans;
33 }

```

1.9 SparseTable

```

1 template<class T, T (*op)(T, T)> struct sparse_table {
2     int n;
3     vector<vector<T>> mat;
4     sparse_table() : n(0) {}
5     sparse_table(const vector<T>& a) {
6         n = static_cast<int>(a.size());
7         int max_log = 32 - __builtin_clz(n);
8         mat.resize(max_log);
9         mat[0] = a;
10        for(int j = 1; j < max_log; ++j) {
11            mat[j].resize(n - (1 << j) + 1);
12            for(int i = 0; i <= n - (1 << j); ++i) {
13                mat[j][i] = op(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
14            }
15        }
16    }
17    inline T prod(int from, int to) const {
18        assert(0 <= from && from <= to && to <= n - 1);
19        int lg = 31 - __builtin_clz(to - from + 1);
20        return op(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
21    }
22 };

```

1.10 DynamicSegtree

```

1 template<class T>struct dynamic_segment_tree{
2     struct node{
3         node *l = NULL,*r = NULL;
4         T sum;
5         node(T k = 0): sum(k){}
6         node(node *p){if(p)*this = *p;}
7         ~node(){
8             for(auto &i:{l,r})
9                 if(i)delete i;
10        }
11        void pull(){
12            sum = 0;
13            for(auto i:{l,r})
14                if(i)sum+=i->sum;
15        }
16    }*root = NULL;
17    int n;
18    dynamic_segment_tree(){}
19    dynamic_segment_tree(const dynamic_segment_tree<T>&tmp){root
20        = new node(tmp.root);}
21    void update(node *t,int pos,T k,int l,int r){
22        if(!t)t = new node();
23        if(l==r)return t = new node(k),void();
24        int m = (l+r)>>1;
25        t = new node(t);
26        if(pos<=m)update(t->l,pos,k,l,m);
27        else update(t->r,pos,k,m+1,r);
28        t->pull();
29    }void update(int pos,T k,int l = -1e9,int r = 1e9){update(
30        root,pos,k,l,r);}
31    T query(node *t,int ql,int qr,int l,int r){
32        if(!t)return 0;
33        if(ql<=l and r<=qr)return t->sum;
34        int m = (l+r)>>1;
35        T ans = 0;
36        if(ql<=m)ans+=query(t->l,ql,qr,l,m);
37        if(qr>m)ans+=query(t->r,ql,qr,m+1,r);
38        return ans;
39    }T query(int ql,int qr,int l = -1e9,int r = 1e9){return query
40        (root,ql,qr,l,r);}
41 };

```

1.11 ZkwSegtree

```

1 template<class S,
2     S (*node_pull)(S, S),

```

```

3     S (*node_init)(),
4     class F,
5     S (*mapping)(S, F),
6     F (*tag_pull)(F, F),
7     F (*tag_init)()>
8 class segment_tree {
9 public:
10    segment_tree() : segment_tree(0) {}
11    explicit segment_tree(int _n) : segment_tree(vector<S>(_n,
12        node_init())) {}
13    explicit segment_tree(const vector<S>& v) : n((int) v.size())
14    {
15        log = std::__lg(2 * n - 1);
16        size = 1 << log;
17        d = vector<S>(size << 1, node_init());
18        lz = vector<F>(size, tag_init());
19        for(int i = 0; i < n; i++) {
20            d[size + i] = v[i];
21        }
22        for(int i = size - 1; i; --i) {
23            update(i);
24        }
25    }
26    void set(int p, S x) {
27        assert(0 <= p && p < n);
28        p += size;
29        for(int i = log; i; --i) {
30            push(p >> i);
31        }
32        d[p] = x;
33        for(int i = 1; i <= log; ++i) {
34            update(p >> i);
35        }
36    }
37    S get(int p) {
38        assert(0 <= p && p < n);
39        p += size;
40        for(int i = log; i; i--) {
41            push(p >> i);
42        }
43        return d[p];
44    }
45    S operator[](int p) {
46        return get(p);
47    }
48    S query(int l, int r) {
49        r++;
50        assert(l<=r);
51        l += size;
52        r += size;
53        for(int i = log; i; i--) {
54            if(((l >> i) << i) != 1) {
55                push(l >> i);
56            }
57            if(((r >> i) << i) != r) {
58                push(r >> i);
59            }
60        }
61        S sml = node_init(), smr = node_init();
62        while(l < r) {
63            if(l & 1) {
64                sml = node_pull(sml, d[l++]);
65            }
66            if(r & 1) {
67                smr = node_pull(d[--r], smr);
68            }
69            l >>= 1;
70            r >>= 1;
71        }
72        return node_pull(sml, smr);
73    }
74    void apply(int p, F f) {
75        assert(0 <= p && p < n);
76        p += size;
77        for(int i = log; i; i--) {
78            push(p >> i);
79        }
80        d[p] = mapping(f, d[p]);
81        for(int i = 1; i <= log; ++i) {
82            update(p >> i);
83        }
84    }
85    void update(int l, int r, F f) {
86        r++;
87        assert(l<=r);
88        l += size;
89        r += size;
90        for(int i = log; i; i--) {
91            if(((l >> i) << i) != 1) {
92                push(l >> i);
93            }
94            if(((r >> i) << i) != r) {
95                push((r - 1) >> i);
96            }
97        }
98    }
99    {
100        int l2 = l, r2 = r;

```

```

98     while(l < r) {
99         if(l & 1) {
100             all_apply(l++, f);
101         }
102         if(r & 1) {
103             all_apply(--r, f);
104         }
105         l >>= 1;
106         r >>= 1;
107     }
108     l = l2;
109     r = r2;
110 }
111 for(int i = 1; i <= log; i++) {
112     if(((l >> i) << i) != 1) {
113         update(l >> i);
114     }
115     if(((r >> i) << i) != r) {
116         update((r - 1) >> i);
117     }
118 }
119 }
120 private:
121 int n, size, log;
122 vector<S> d;
123 vector<F> lz;
124 inline void update(int k) { d[k] = node_pull(d[k << 1], d[k
    << 1 | 1]); }
125 void all_apply(int k, F f) {
126     d[k] = mapping(d[k], f);
127     if(k < size) {
128         lz[k] = tag_pull(lz[k], f);
129     }
130 }
131 void push(int k) {
132     all_apply(k << 1, lz[k]);
133     all_apply(k << 1 | 1, lz[k]);
134     lz[k] = tag_init();
135 }
136 };

```

1.12 MoAlgo

```

1 struct qry{
2     int ql,qr,id;
3 };
4 template<class T>struct Mo{
5     int n,m;
6     vector<pii>ans;
7     Mo(int _n,int _m): n(_n),m(_m){
8         ans.resize(m);
9     }
10    void solve(vector<T>&v,vector<qry>&q){
11        int l = 0,r = -1;
12        vector<int>cnt,cntcnt;
13        cnt.resize(n+5);
14        cntcnt.resize(n+5);
15        int mx = 0;
16        function<void(int)>add = [&](int pos){
17            cntcnt[cnt[v[pos]]]--;
18            cnt[v[pos]]++;
19            cntcnt[cnt[v[pos]]]++;
20            mx = max(mx,cnt[v[pos]]);
21        };
22        function<void(int)>sub = [&](int pos){
23            if(--cntcnt[cnt[v[pos]]] and cnt[v[pos]]==mx)mx--;
24            cnt[v[pos]]--;
25            cntcnt[cnt[v[pos]]]++;
26            mx = max(mx,cnt[v[pos]]);
27        };
28        sort(all(q),[&](qry a,qry b){
29            static int B = max((int)1,n/max((int)sqrt(m),(int)1));
30            if(a.ql/B!=b.ql/B)return a.ql<b.ql;
31            if((a.ql/B)&1)return a.qr>b.qr;
32            return a.qr<b.qr;
33        });
34        for(auto [ql,qr,id]:q){
35            while(l>ql)add(--l);
36            while(r<qr)add(++r);
37            while(l<ql)sub(l++);
38            while(r>qr)sub(r--);
39            ans[id] = {mx,cntcnt[mx]};
40        }
41    }
42 };

```

1.13 Hash

```

1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;

```

```

4         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7     }
8     size_t operator()(uint64_t x) const {
9         static const uint64_t FIXED_RANDOM = chrono::steady_clock::
            now().time_since_epoch().count();
10        return splitmix64(x + FIXED_RANDOM);
11    }
12    size_t operator()(pair<uint64_t,uint64_t> x) const {
13        static const uint64_t FIXED_RANDOM = chrono::steady_clock::
            now().time_since_epoch().count();
14        return splitmix64(3*x.first + x.second + FIXED_RANDOM);
15    }
16 };
17 template<class T,class U>using hash_map = gp_hash_table<T,U,
    custom_hash>;

```

1.14 RedBlackTree

```

1 template<class T, typename cmp=less<>>struct _tree{//#include<
    bits/extc++.h>
2     tree<pair<T,int>,null_type,cmp,rb_tree_tag,
        tree_order_statistics_node_update>st;
3     int id = 0;
4     void insert(T x){st.insert({x,id++});}
5     void erase(T x){st.erase(st.lower_bound({x,0}));}
6     int order_of_key(T x){return st.order_of_key(*st.lower_bound
            ({x,0}));}
7     T find_by_order(int x){return st.find_by_order(x)->first;}
8     T lower_bound(T x){return st.lower_bound({x,0})->first;}
9     T upper_bound(T x){return st.upper_bound({x,(int)1e9+7})->
        first;}
10    T smaller_bound(T x){return (--st.lower_bound({x,0}))->first
        ;}
11 };

```

2 Geometry

2.1 Theorem

- Pick's Theorem

$$A = I + \frac{B}{2} - 1$$

$$A := \text{Area}$$

$$i := \text{PointsInside}$$

$$B := \text{PointsBoundary}$$

2.2 PointInPolygon

```

1 template<class T>
2 int PointInPolygon(const vector<Point<T>> &Poly, const Point<T>
    p){
3     int ans = 0;
4     for(auto a = --Poly.end(),b = Poly.begin();b!=Poly.end();a =
        b++){
5         if(PointOnSegment(*a,*b,p)){
6             return -1;
7         }
8         if(seg_intersect(p,p+Point<T>(2e9+7,1),*a,*b)){
9             ans = !ans;
10        }
11    }
12    return ans;
13 }

```

2.3 PointInConvex

```

1 template<class T>
2 int PointInConvex(const vector<Point<T>>&C,const Point<T>&p){
3     if(btw(C[0],C[1],p) || btw(C[0],C.back(),p))return -1;
4     int l = 0,r = (int)C.size()-1;
5     while(l<r){
6         int m = (l+r)>>1;
7         auto a1 = (C[m]-C[0])^(p-C[0]);
8         auto a2 = (C[(m+1)%C.size()]-C[0])^(p-C[0]);
9         if(a1>=0 and a2<=0){
10            auto res = (C[(m+1)%C.size()]-C[m])^(p-C[m]);
11            return res > 0 ? 1 : (res >= 0 ? -1 : 0);
12        }
13        if(a1 < 0) r = m-1;
14        else l = m+1;
15    }

```

```

16 | return 0;
17 | }

```

2.4 MaximumDistance

```

1 | template<class T>
2 | T MaximumDistance(vector<Point<T>>&p){
3 |     vector<Point<T>>C = ConvexHull(p,0);
4 |     int n = C.size(),t = 2;
5 |     T ans = 0;
6 |     for(int i = 0;i<n;i++){
7 |         while(((C[i] - C[t]) ^ (C[(i+1)%n] - C[t])) < ((C[i] - C[(t
8 |             +1)%n]) ^ (C[(i+1)%n] - C[(t+1)%n]))) t = (t + 1)%n;
9 |         ans = max({ans, abs2(C[i] - C[t]), abs2(C[(i+1)%n] - C[t])
10 |             });
11 |     }
12 |     return ans;
13 | }

```

2.5 PolarAngleSort

```

1 | template<class T>
2 | bool cmp(const Point<T> &a,const Point<T> &b){
3 |     int lhs = (a.y < 0 || a.y==0 && a.x > 0) ? 0 : (1 + (a.x != 0
4 |         || a.y != 0));
5 |     int rhs = (b.y < 0 || b.y==0 && b.x > 0) ? 0 : (1 + (b.x != 0
6 |         || b.y != 0));
7 |     if(lhs != rhs) {
8 |         return lhs < rhs;
9 |     }
10 |     long long area = (a^b);
11 |     return area ? area > 0 : abs(a.x) + abs(a.y) < abs(b.x) + abs
12 |         (b.y);
13 | }

```

2.6 MinimumDistance

```

1 | template<class T>
2 | T MinimumDistance(vector<Point<T>>&p,int l = -1,int r = -1){
3 |     if(l==l and r==r){
4 |         sort(p.begin(),p.end(),[(Point<T> a,Point<T> b){
5 |             return a.x<b.x;
6 |         }]);
7 |         return MinimumDistance(p,0,p.size()-1);
8 |     }
9 |     if(l==r)return numeric_limits<T>::max();
10 |     int m = (l+r)>>1,mid_pos = p[m].x;
11 |     T ans = min(MinimumDistance(p,l,m),MinimumDistance(p,m+1,r));
12 |     vector<Point<T>>tmp((r-l+1),Point<T>(0,0));
13 |     merge(p.begin()+l,p.begin()+m+1, p.begin()+m+1,p.begin()+r+1,
14 |         tmp.begin(), [(Point<T> a,Point<T> b){return a.y<b.y;});
15 |     for(int i = l;i<r;i++)p[i] = tmp[i-l];
16 |     tmp.clear();
17 |     for(int i = l;i<r;i++){
18 |         if(abs(p[i].x-mid_pos)<=ans){
19 |             tmp.push_back(p[i]);
20 |         }
21 |     }
22 |     int n = tmp.size();
23 |     for(int i = 0;i<n;i++){
24 |         for(int j = i+1;j<n;j++){
25 |             ans = min(ans,abs2(tmp[i]-tmp[j]));
26 |             if(((tmp[i].y-tmp[j].y)*(tmp[i].y-tmp[j].y))>ans){
27 |                 break;
28 |             }
29 |         }
30 |     }
31 |     return ans;
32 | }

```

2.7 ConvexHull

```

1 | template<class T>
2 | vector<Point<T>> ConvexHull(vector<Point<T>> v,bool Boundary =
3 |     1){
4 |     sort(begin(v),end(v),[(Point<T> &a,Point<T> &b){
5 |         if(a.x!=b.x)return a.x<b.x;
6 |         return a.y<b.y;
7 |     }]);
8 |     vector<Point<T>>ans;
9 |     int t = 1;
10 |     auto add = [(Point<T> &p){

```

```

10 | while(ans.size() > t and ((p - ans[ans.size() - 2])^(ans.
11 |     back() - ans[ans.size() - 2])) > (Boundary ? 0 : 0-eps)
12 | )
13 |     ans.pop_back();
14 |     ans.push_back(p);
15 | };
16 | for(int i = 0; i < v.size(); ++i) add(v[i]);
17 | t = ans.size();
18 | for(int i = (int)(v.size()-2; i >= 0; --i) add(v[i]);
19 | if(v.size() > 1) ans.pop_back();
20 | return ans;
21 | }

```

2.8 Template

```

1 | template<class T>
2 | struct Point{
3 |     T x,y;
4 |     Point(T x = 0,T y = 0) : x(x), y(y) {}
5 |     Point operator + (const Point &b) const {
6 |         return Point(x + b.x,y + b.y);
7 |     }
8 |     Point operator - (const Point &b) const {
9 |         return Point(x - b.x,y - b.y);
10 |    }
11 |    Point operator * (T b) const {
12 |        return Point(x*b,y*b);
13 |    }
14 |    Point operator / (T b) const {
15 |        return Point(x/b,y/b);
16 |    }
17 |    T operator * (const Point &b) const {
18 |        return x * b.x + y * b.y;
19 |    }
20 |    T operator ^ (const Point &b) const {
21 |        return x * b.y - y * b.x;
22 |    }
23 | };
24 | int sign(double a){
25 |     return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;
26 | }
27 | template<class T>
28 | double abs(const Point<T>&p){
29 |     return sqrtl(p*p);
30 | }
31 | template<class T>
32 | T abs2(const Point<T>&p){
33 |     return p*p;
34 | }
35 | template<class T>
36 | int ori(Point<T> a,Point<T> b,Point<T> c){
37 |     return sign((b-a)^(c-a));
38 | }
39 | template<class T>
40 | bool collinearity(Point<T> p1,Point<T> p2,Point<T> p3){
41 |     return sign((p1-p3)^(p2-p3)) == 0;
42 | }
43 | template<class T>
44 | bool btw(Point<T> p1,Point<T> p2,Point<T> p3) {
45 |     if(!collinearity(p1, p2, p3)) return 0;
46 |     return sign((p1-p3)*(p2-p3)) <= 0;
47 | }
48 | template<class T>
49 | bool PointOnSegment(const Point<T> &p1,const Point<T> &p2,
50 |     const Point<T> &p3){
51 |     return collinearity(p1,p2,p3) && btw(p1,p2,p3);
52 | }
53 | template<class T>
54 | bool seg_intersect(Point<T> p1, Point<T> p2, Point<T> p3, Point
55 |     <T> p4) {
56 |     int a123 = ori(p1, p2, p3);
57 |     int a124 = ori(p1, p2, p4);
58 |     int a341 = ori(p3, p4, p1);
59 |     int a342 = ori(p3, p4, p2);
60 |     if(a123 == 0 && a124 == 0)
61 |         return btw(p1, p2, p3) || btw(p1, p2, p4) || btw(p3, p4, p1
62 |             ) || btw(p3, p4, p2);
63 |     return a123 * a124 <= 0 && a341 * a342 <= 0;
64 | }
65 | template<class T>
66 | double area(vector<Point<T>> v){
67 |     if(v.size()<=2)return 0;
68 |     double ans = 0;
69 |     for(int i = 1;i<v.size()-1;i++){
70 |         ans+=((v[i]-v[0])^(v[i+1]-v[0]));
71 |     }
72 |     return abs(ans)/2.;
73 | }

```

3 Graph

3.1 HLD

```

1 struct HLD{
2     int n,root;
3     vector<int>dep,father,sz,mxson,topf,id;
4     HLD(int _n,int _root,vector<vector<int>>&g): n(_n),root(_root)
5     {
6         dep.resize(n+5);
7         father.resize(n+5);
8         sz.resize(n+5);
9         mxson.resize(n+5);
10        topf.resize(n+5);
11        id.resize(n+5);
12        function<void(int,int)>dfs = [&](int u,int p){
13            dep[u] = dep[p]+1;
14            father[u] = p;
15            sz[u] = 1;
16            mxson[u] = 0;
17            for(auto v:g[u]){
18                if(v!=p){
19                    dfs(v,u);
20                    sz[u]+=sz[v];
21                    if(sz[v]>sz[mxson[u]])mxson[u] = v;
22                }
23            };
24            function<void(int,int)>dfs2 = [&](int u,int top){
25                static int idn = 0;
26                topf[u] = top;
27                id[u] = ++idn;
28                if(mxson[u])dfs2(mxson[u],top);
29                for(auto v:g[u]){
30                    if(v!=father[u] and v!=mxson[u]){
31                        dfs2(v,v);
32                    }
33                };
34                dfs(root,0);
35                dfs2(root,root);
36            }
37            int query(int u,int v,const auto &qry,const auto &op){
38                int ans = 0;
39                while(topf[u]!=topf[v]){
40                    if(dep[topf[u]]<dep[topf[v]])swap(u,v);
41                    ans = op(ans,qry(id[topf[u]],id[u]));
42                    u = father[topf[u]];
43                }
44                if(id[u]>id[v])swap(u,v);
45                ans = op(ans,qry(id[u],id[v]));
46                return ans;
47            }
48            void update(int u,int v,int val,const auto &upd){
49                while(topf[u]!=topf[v]){
50                    if(dep[topf[u]]<dep[topf[v]])swap(u,v);
51                    upd(id[topf[u]],id[u],val);
52                    u = father[topf[u]];
53                }
54                if(id[u]>id[v])swap(u,v);
55                upd(id[u],id[v],val);
56            }
57        };
58    };

```

3.2 Bridges

```

1 vector<pii> findBridges(const vector<vector<int>>& g) {
2     int n = (int) g.size();
3     vector<int> id(n, -1), low(n);
4     vector<pii> bridges;
5     function<void(int, int)> dfs = [&](int u, int p) {
6         static int cnt = 0;
7         id[u] = low[u] = cnt++;
8         for(auto v : g[u]) {
9             if(v == p) continue;
10            if(id[v] != -1) low[u] = min(low[u], id[v]);
11            else {
12                dfs(v, u);
13                low[u] = min(low[u], low[v]);
14                if(low[v] > id[u]) bridges.emplace_back(u, v);
15            }
16        }
17    };
18    for(int i = 0; i < n; ++i) {
19        if(id[i] == -1) dfs(i, -1);
20    }
21    return bridges;
22 }

```

3.3 MCMF

```

1 template<class Cap_t, class Cost_t>
2 class MCMF {
3 public:
4     struct Edge {
5         int from;
6         int to;
7         Cap_t cap;
8         Cost_t cost;
9         Edge(int u, int v, Cap_t _cap, Cost_t _cost) : from(u), to(v), cap(_cap), cost(_cost) {}
10    };
11
12    static constexpr Cap_t EPS = static_cast<Cap_t>(1e-9);
13
14    int n;
15    vector<Edge> edges;
16    vector<vector<int>> g;
17    vector<Cost_t> d;
18    vector<bool> in_queue;
19    vector<int> previous_edge;
20
21    MCMF() {}
22    MCMF(int _n) : n(_n+1), g(_n+1), d(_n+1), in_queue(_n+1), previous_edge(_n+1) {}
23
24    void add_edge(int u, int v, Cap_t cap, Cost_t cost) {
25        assert(0 <= u && u < n);
26        assert(0 <= v && v < n);
27        g[u].push_back(edges.size());
28        edges.emplace_back(u, v, cap, cost);
29        g[v].push_back(edges.size());
30        edges.emplace_back(v, u, 0, -cost);
31    }
32
33    bool spfa(int s, int t) {
34        bool found = false;
35        fill(d.begin(), d.end(), numeric_limits<Cost_t>::max());
36        d[s] = 0;
37        in_queue[s] = true;
38        queue<int> que;
39        que.push(s);
40        while(!que.empty()) {
41            int u = que.front();
42            que.pop();
43            if(u == t) {
44                found = true;
45            }
46            in_queue[u] = false;
47            for(auto& id : g[u]) {
48                const Edge& e = edges[id];
49                if(e.cap > EPS && d[u] + e.cost < d[e.to]) {
50                    d[e.to] = d[u] + e.cost;
51                    previous_edge[e.to] = id;
52                    if(!in_queue[e.to]) {
53                        que.push(e.to);
54                        in_queue[e.to] = true;
55                    }
56                }
57            }
58        }
59        return found;
60    }
61
62    pair<Cap_t, Cost_t> flow(int s, int t, Cap_t f = numeric_limits<Cap_t>::max()) {
63        assert(0 <= s && s < n);
64        assert(0 <= t && t < n);
65        Cap_t cap = 0;
66        Cost_t cost = 0;
67        while(f > 0 && spfa(s, t)) {
68            Cap_t send = f;
69            int u = t;
70            while(u != s) {
71                const Edge& e = edges[previous_edge[u]];
72                send = min(send, e.cap);
73                u = e.from;
74            }
75            u = t;
76            while(u != s) {
77                Edge& e = edges[previous_edge[u]];
78                e.cap -= send;
79                Edge& b = edges[previous_edge[u] ^ 1];
80                b.cap += send;
81                u = e.from;
82            }
83            cap += send;
84            f -= send;
85            cost += send * d[t];
86        }
87        return make_pair(cap, cost);
88    }
89 };

```


3.4 LCA

```

1 vector<vector<int>>>g,dp;
2 vector<int>deep;
3 void build(int root,int n){
4     dp.assign(25,vector<int>(n+5));
5     deep.assign(n+5,0);
6     function<void(int,int,int)>dfs = [&](int u,int p,int dis){
7         dp[0][u] = p;
8         deep[u] = dis;
9         for(auto v:g[u]){
10             if(v==p)continue;
11             dfs(v,u,dis+1);
12         }
13     };
14     dfs(root,0,1);
15     for(int i = 1;i<=20;++i){
16         for(int j = 1;j<=n;++j){
17             dp[i][j] = dp[i-1][dp[i-1][j]];
18         }
19     }
20 }
21 int LCA(int u,int v){
22     if(deep[u]<deep[v])swap(u,v);
23     for(int i = 20;i>=0;--i){
24         if(deep[dp[i][u]]>=deep[v])
25             u = dp[i][u];
26     }
27     if(u==v)return u;
28     for(int i = 20;i>=0;--i){
29         if(dp[i][u]!=dp[i][v])u = dp[i][u],v = dp[i][v];
30     }
31     return dp[0][u];
32 }

```

3.5 CentroidDecomposition

```

1 vector<vector<int>>>g;
2 vector<int>sz,tmp;
3 vector<bool>vis; //visit_centroid
4 int tree_centroid(int u,int n){
5     function<void(int,int)>dfs1 = [&](int u,int p){
6         sz[u] = 1;
7         for(auto v:g[u]){
8             if(v==p)continue;
9             if(vis[v])continue;
10            dfs1(v,u);
11            sz[u]+=sz[v];
12        }
13    };
14    function<int(int,int)>dfs2 = [&](int u,int p){
15        for(auto v:g[u]){
16            if(v==p)continue;
17            if(vis[v])continue;
18            if(sz[v]*2<n)continue;
19            return dfs2(v,u);
20        }
21        return u;
22    };
23    dfs1(u,-1);
24    return dfs2(u,-1);
25 }
26 int cal(int u,int p = -1,int deep = 1){
27     int ans = 0;
28     tmp.pb(deep);
29     sz[u] = 1;
30     for(auto v:g[u]){
31         if(v==p)continue;
32         if(vis[v])continue;
33         ans+=cal(v,u,deep+1);
34         sz[u]+=sz[v];
35     }
36     //calcuat the answer
37     return ans;
38 }
39 int centroid_decomposition(int u,int tree_size){
40     int center = tree_centroid(u,tree_size);
41     vis[center] = 1;
42     int ans = 0;
43     for(auto v:g[center]){
44         if(vis[v])continue;
45         ans+=cal(v);
46         for(int i = sz[tmp]-sz[v];i<sz[tmp];++i){
47             //update
48         }
49     }
50     while(!tmp.empty()){
51         //roll_back(tmp.back())
52         tmp.pop_back();
53     }
54     for(auto v:g[center]){
55         if(vis[v])continue;
56         ans+=centroid_decomposition(v,sz[v]);

```

```

57     }
58     return ans;
59 }

```

3.6 BCC AP

```

1 struct BCC_AP{
2     int dfn_cnt = 0,bcc_cnt = 0,n;
3     vector<int>dfn,low,ap,bcc_id;
4     stack<int>st;
5     vector<bool>vis,is_ap;
6     vector<vector<int>>>bcc;
7     BCC_AP(int _n):n(_n){
8         dfn.resize(n+5),low.resize(n+5),bcc.resize(n+5),vis.resize(
9             n+5),is_ap.resize(n+5),bcc_id.resize(n+5);
10    }
11    inline void build(const vector<vector<int>>>&g,int u,int p =
12        -1){
13        int child = 0;
14        dfn[u] = low[u] = ++dfn_cnt;
15        st.push(u);
16        vis[u] = 1;
17        if(g[u].empty() and p==-1){
18            bcc_id[u] = ++bcc_cnt;
19            bcc[bcc_cnt].push_back(u);
20            return;
21        }
22        for(auto v:g[u]){
23            if(v==p)continue;
24            if(!dfn[v]){
25                build(g,v,u);
26                child++;
27                if(dfn[u]<=low[v]){
28                    is_ap[u] = 1;
29                    bcc_id[u] = ++bcc_cnt;
30                    bcc[bcc_cnt].push_back(u);
31                    while(vis[v]){
32                        bcc_id[st.top()] = bcc_cnt;
33                        bcc[bcc_cnt].push_back(st.top());
34                        vis[st.top()] = 0;
35                        st.pop();
36                    }
37                }
38                low[u] = min(low[u],low[v]);
39            }
40            low[u] = min(low[u],dfn[v]);
41        }
42        if(p==-1 and child<2)is_ap[u] = 0;
43        if(is_ap[u])ap.push_back(u);
44    };

```

3.7 SCC

```

1 struct SCC{
2     int n,cnt = 0,dfn_cnt = 0;
3     vector<int>sz,scc,low,dfn;
4     stack<int>st;
5     vector<bool>vis;
6     SCC(int _n):n(_n){
7         sz.resize(n+5),scc.resize(n+5),low.resize(n+5),dfn.resize(n
8             +5),vis.resize(n+5);
9     }
10    inline void build(const vector<vector<int>>>&g,int u,int dis =
11        1){
12        low[u] = dfn[u] = ++dfn_cnt,vis[u] = 1;
13        st.push(u);
14        for(auto v:g[u]){
15            if(!dfn[v]){
16                build(g,v,dis+1);
17                low[u] = min(low[u],low[v]);
18            }
19            else if(vis[v]){
20                low[u] = min(low[u],dfn[v]);
21            }
22        }
23        if(low[u]==dfn[u]){
24            ++cnt;
25            while(vis[u]){
26                auto v = st.top();
27                st.pop();
28                vis[v] = 0;
29                scc[v] = cnt;
30                sz[cnt]++;
31            }
32        }
33    }
34    vector<vector<int>> compress(const vector<pii>&e,vector<int>&
35        ind){
36        vector<vector<int>>>ans(n+5);
37        for(auto [u,v]:e){

```

```

35     if(scc[u]==scc[v])continue;
36     ans[scc[u]].pb(scc[v]);
37     ind[scc[v]]++;
38 }
39 return ans;
40 }
41 };

```

3.8 Dinic

```

1  template<class T>
2  struct Dinic{
3      struct edge{
4          int from, to;
5          T cap;
6          edge(int _from, int _to, T _cap) : from(_from), to(_to),
7              cap(_cap) {}
8      };
9      int n;
10     vector<edge> edges;
11     vector<vector<int>> g;
12     vector<int> cur, h;
13     Dinic(int _n) : n(_n+1), g(_n+1) {}
14     void add_edge(int u, int v, T cap){
15         g[u].push_back(edges.size());
16         edges.push_back(edge(u, v, cap));
17         g[v].push_back(edges.size());
18         edges.push_back(edge(v, u, 0));
19     }
20     bool bfs(int s, int t){
21         h.assign(n, -1);
22         h[s] = 0;
23         queue<int> que;
24         que.push(s);
25         while(!que.empty()) {
26             int u = que.front();
27             que.pop();
28             for(auto id : g[u]) {
29                 const edge& e = edges[id];
30                 int v = e.to;
31                 if(e.cap > 0 && h[v] == -1) {
32                     h[v] = h[u] + 1;
33                     if(v == t) {
34                         return 1;
35                     }
36                     que.push(v);
37                 }
38             }
39         }
40         return 0;
41     }
42     T dfs(int u, int t, T f) {
43         if(u == t) {
44             return f;
45         }
46         T r = f;
47         for(int& i = cur[u]; i < (int) g[u].size(); ++i) {
48             int id = g[u][i];
49             const edge& e = edges[id];
50             int v = e.to;
51             if(e.cap > 0 && h[v] == h[u] + 1) {
52                 T send = dfs(v, t, min(r, e.cap));
53                 edges[id].cap -= send;
54                 edges[id ^ 1].cap += send;
55                 r -= send;
56                 if(r == 0) {
57                     return f;
58                 }
59             }
60         }
61         return f - r;
62     }
63     T flow(int s, int t, T f = numeric_limits<T>::max()) {
64         T ans = 0;
65         while(f > 0 && bfs(s, t)) {
66             cur.assign(n, 0);
67             T send = dfs(s, t, f);
68             ans += send;
69             f -= send;
70         }
71         return ans;
72     }
73     vector<pair<int, int>> min_cut(int s) {
74         vector<bool> vis(n);
75         vis[s] = true;
76         queue<int> que;
77         que.push(s);
78         while(!que.empty()) {
79             int u = que.front();
80             que.pop();
81             for(auto id : g[u]) {
82                 const auto& e = edges[id];
83                 int v = e.to;
84                 if(e.cap > 0 && !vis[v]) {

```

```

84         vis[v] = true;
85         que.push(v);
86     }
87 }
88 }
89 vector<pair<int, int>> cut;
90 for(int i = 0; i < (int) edges.size(); i += 2) {
91     const auto& e = edges[i];
92     if(vis[e.from] && !vis[e.to]) {
93         cut.push_back(make_pair(e.from, e.to));
94     }
95 }
96 return cut;
97 }
98 };

```

4 Math

4.1 Numbers

- Bernoulli numbers

$$B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

$$\sum_{j=0}^m \binom{m+1}{j} B_j = 0, \text{EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

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

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

- Pentagonal number theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2} \right)$$

- Catalan numbers

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$$

$$C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$$

- Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j :s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j :s s.t. $\pi(j) \geq j$, k j :s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

4.2 ExtendGCD

```

1 // @return $x, $y s.t. $ax + by = \gcd(a, b)$
2 ll ext_gcd(ll a, ll b, ll& x, ll& y) {
3     if(b == 0) {
4         x = 1; y = 0;
5         return a;
6     }
7     ll x2, y2;
8     ll c = a % b;
9     if(c < 0) c += b;
10    ll g = ext_gcd(b, c, x2, y2);
11    x = y2;
12    y = x2 - (a / b) * y2;
13    return g;
14 }

```

4.3 InvGCD

```

1 pair<long long, long long> inv_gcd(long long a, long long b) {
2     a %= b;
3     if(a < 0) a += b;
4     if(a == 0) return {b, 0};
5     long long s = b, t = a;
6     long long m0 = 0, m1 = 1;
7     while(t) {
8         long long u = s / t;
9         s -= t * u;
10        m0 -= m1 * u;
11        swap(s, t);

```



```

12 swap(m0, m1);
13 }
14 if(m0 < 0) m0 += b / s;
15 return {s, m0};
16 }

```

4.4 Generating Functions

- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$

$$\begin{aligned}
 & - A(rx) \Rightarrow r^n a_n \\
 & - A(x) + B(x) \Rightarrow a_n + b_n \\
 & - A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i} \\
 & - A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k} \\
 & - xA(x)' \Rightarrow n a_n \\
 & - \frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i
 \end{aligned}$$

- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

$$\begin{aligned}
 & - A(x) + B(x) \Rightarrow a_n + b_n \\
 & - A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i} \\
 & - A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k} \\
 & - xA(x) \Rightarrow n a_n
 \end{aligned}$$

- Special Generating Function

$$\begin{aligned}
 & - (1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i \\
 & - \frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{n-1}{i} x^i
 \end{aligned}$$

4.5 Theorem

- Cramer's rule

$$\begin{aligned}
 ax + by &= e & x &= \frac{ed - bf}{ad - bc} \\
 cx + dy &= f & y &= \frac{af - ec}{ad - bc}
 \end{aligned}$$

- Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

- Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

- Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ spanning trees.
- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erdős–Gallai theorem

A sequence of nonnegative integers $d_1 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \dots + d_n$

is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every $1 \leq k \leq n$.

- Gale–Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Fulkerson–Chen–Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \dots \geq a_n$ is digraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Möbius inversion formula

$$\begin{aligned}
 & - f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) \\
 & - f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d)
 \end{aligned}$$

- Spherical cap

- A portion of a sphere cut off by a plane.
- r : sphere radius, a : radius of the base of the cap, h : height of the cap, θ : $\arcsin(a/r)$.
- Volume $= \pi h^2 (3r - h) / 3 = \pi h (3a^2 + h^2) / 6 = \pi r^3 (2 + \cos \theta) (1 - \cos \theta)^2 / 3$.
- Area $= 2\pi r h = \pi (a^2 + h^2) = 2\pi r^2 (1 - \cos \theta)$.

4.6 FloorSum

```

1 // @param $n < 2^{32}$
2 // @param $1 \leq m < 2^{32}$
3 // @return $\sum_{i=0}^{n-1} \lfloor \frac{ai+b}{m} \rfloor$
4 ull floor_sum_unsigned(ull n, ull m, ull a, ull b) {
5     ull ans = 0;
6     while(true) {
7         if(a >= m) {
8             ans += n * (n - 1) / 2 * (a / m);
9             a %= m;
10        }
11        if(b >= m) {
12            ans += n * (b / m);
13            b %= m;
14        }
15        ull y_max = a * n + b;
16        if(y_max < m) break;
17        n = (ull)(y_max / m);
18        b = (ull)(y_max % m);
19        swap(m, a);
20    }
21    return ans;
22 }
23
24 ll floor_sum(ll n, ll m, ll a, ll b) {
25     assert(0 <= n && n < (1LL << 32));
26     assert(1 <= m && m < (1LL << 32));
27     ull ans = 0;
28     if(a < 0) {
29         ull a2 = (a % m + m) % m;
30         ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
31         a = a2;
32     }
33     if(b < 0) {
34         ull b2 = (b % m + m) % m;
35         ans -= 1ULL * n * ((b2 - b) / m);
36         b = b2;
37     }
38     return ans + floor_sum_unsigned(n, m, a, b);
39 }

```

4.7 Factorizer

```

1 template<class T>
2 vector<pair<T, int>> MergeFactors(const vector<pair<T, int>>& a
3     , const vector<pair<T, int>>& b) {
4     vector<pair<T, int>> c;
5     int i = 0, j = 0;
6     while(i < SZ(a) || j < SZ(b)) {
7         if(i < SZ(a) && j < SZ(b) && a[i].F == b[j].F) {
8             c.EB(a[i].F, a[i].S + b[j].S);
9             ++i, ++j;
10            continue;
11        }
12        if(j == SZ(b) || (i < SZ(a) && a[i].F < b[j].F)) c.PB(a[i]
13            ++);
14        else c.PB(b[j++]);
15    }
16    return c;
17 }
18
19 template<class T>
20 vector<pair<T, int>> RhoC(const T& n, const T& c) {
21     if(n <= 1) return {};
22     if(n % 2 == 0) return MergeFactors({{2, 1}}, RhoC(n / 2, c));
23     if(is_prime_constexpr(n)) return {{n, 1}};
24     T x = 2, saved = 2, p = 1, lam = 1;
25     while(true) {
26         x = (x * x % n + c) % n;
27         T g = __gcd((x - saved) % n, n);
28         if(g != 1) return MergeFactors(RhoC(g, c + 1), RhoC(n / g,
29             c + 1));
30         if(p == lam) {
31             saved = x;
32             p <<= 1;
33             lam = 0;
34         }
35         lam += 1;
36     }
37     return {};
38 }
39
40 template<class T>
41 vector<pair<T, int>> Factorize(T n) {
42     if(n <= 1) return {};
43     return RhoC(n, T(1));
44 }

```

```

40 template<class T>
41 vector<T> BuildDivisorsFromFactors(const vector<pair<T, int>>&
    factors) {
42     int total = 1;
43     for(int i = 0; i < SZ(factors); ++i) total *= factors[i].
        second + 1;
44     vector<T> divisors;
45     divisors.reserve(total);
46     divisors.PB(1);
47     for(auto [p, cnt] : factors) {
48         int sz = SZ(divisors);
49         for(int i = 0; i < sz; ++i) {
50             T cur = divisors[i];
51             for(int j = 0; j < cnt; ++j) {
52                 cur *= p;
53                 divisors.PB(cur);
54             }
55         }
56     }
57     // sort(ALL(divisors));
58     return divisors;
59 }

```

4.8 PowMod

```

1 constexpr long long Pow(long long x, long long n, int m) {
2     if(m == 1) return 0;
3     unsigned int _m = (unsigned int)(m);
4     unsigned long long r = 1;
5     x %= m;
6     if(x < 0) x += m;
7     unsigned long long y = x;
8     while(n) {
9         if(n & 1) r = (r * y) % _m;
10        y = (y * y) % _m;
11        n >>= 1;
12    }
13    return r;
14 }

```

4.9 CRT

```

1 // @return
2 // ${text}{remainder, modulo}$
3 // or
4 // $0, 0$ if do not exist
5 pair<long long, long long> crt(const vector<long long>& r,
    const vector<long long>& m) {
6     assert(r.size()==m.size());
7     int n = r.size();
8     // Contracts: 0 <= r0 < m0
9     long long r0 = 0, m0 = 1;
10    for(int i = 0; i < n; i++) {
11        assert(1 <= m[i]);
12        long long r1 = r[i] % m[i];
13        if(r1 < 0) r1 += m[i];
14        long long m1 = m[i];
15        if(m0 < m1) {
16            swap(r0, r1);
17            swap(m0, m1);
18        }
19        if(m0 % m1 == 0) {
20            if(r0 % m1 != r1) return {0, 0};
21            continue;
22        }
23        long long g, im;
24        tie(g, im) = inv_gcd(m0, m1);
25        long long u1 = (m1 / g);
26        if((r1 - r0) % g) return {0, 0};
27        long long x = (r1 - r0) / g * u1 * im % u1;
28        r0 += x * m0;
29        m0 *= u1;
30        if(r0 < 0) r0 += m0;
31    }
32    return {r0, m0};
33 }

```

4.10 DiscreteLog

```

1 int DiscreteLog(int s, int x, int y, int m) {
2     constexpr int K = 0;
3     hash_map<int, int> p;
4     int b = 1;
5     for(int i = 0; i < K; ++i) {
6         p[y] = i;
7         y = 1LL * y * x % m;
8         b = 1LL * b * x % m;

```

```

9     }
10    for(int i = 0; i < m + 10; i += K) {
11        s = 1LL * s * b % m;
12        if(p.find(s) != p.end()) return i + K - p[s];
13    }
14    return -1;
15 }
16 int DiscreteLog(int x, int y, int m) {
17     if(m == 1) return 0;
18     int s = 1;
19     for(int i = 0; i < 100; ++i) {
20         if(s == y) return i;
21         s = 1LL * s * x % m;
22     }
23     if(s == y) return 100;
24     int p = 100 + DiscreteLog(s, x, y, m);
25     return (pow_mod(x, p, m) != y ? -1 : p);
26 }

```

4.11 LinearSieve

```

1 vector<bool> is_prime;
2 vector<int> primes, phi, mobius;
3 void linear_sieve(int n) {
4     n += 1;
5     is_prime.resize(n);
6     fill(2 + begin(is_prime), end(is_prime), true);
7     phi.resize(n); mobius.resize(n);
8     phi[1] = mobius[1] = 1;
9     for(int i = 2; i < n; ++i) {
10        if(is_prime[i]) {
11            primes.push_back(i);
12            phi[i] = i - 1;
13            mobius[i] = -1;
14        }
15        for(auto j : primes) {
16            if(i * j >= n) break;
17            is_prime[i * j] = false;
18            if(i % j == 0) {
19                mobius[i * j] = 0;
20                phi[i * j] = phi[i] * j;
21                break;
22            } else {
23                mobius[i * j] = mobius[i] * mobius[j];
24                phi[i * j] = phi[i] * phi[j];
25            }
26        }
27    }
28 }

```

5 Misc

5.1 FastIO

```

1 inline char gc() {
2     static const int BUF_SIZE = 1 << 22;
3     static int Counts = 1 << 23;
4     static char Buffer[BUF_SIZE];
5     static char *Pointer = Buffer, *End = Buffer;
6     if(Pointer == End) {
7         if(Counts < BUF_SIZE) {
8             return EOF;
9         }
10        Counts = fread(Buffer, 1, BUF_SIZE, stdin);
11        Pointer = Buffer;
12        End = Buffer + Counts;
13    }
14    return *(Pointer++);
15 }
16
17 template<class T>
18 inline void read(T& x) {
19     static char c;
20     do {
21         c = gc();
22     } while(c < '0' && c != '-');
23     bool neg = (c == '-');
24     if(!neg) {
25         x = c - '0';
26     } else x = 0;
27     while((c = gc()) >= '0') {
28         x = (x << 3) + (x << 1) + (c & 15);
29     }
30     if(neg) {
31         x = -x;
32     }
33 }
34
35 template<class T, class... U>

```

```

36 inline void read(T& a, U&... b) {
37     read(a);
38     read(b...);
39 }
40
41 template<class T>
42 inline void write(T temp, char end = '\n') {
43     static short digits[20], P;
44     if(temp == 0) {
45         putchar_unlocked('0');
46         putchar_unlocked(end);
47         return;
48     }
49     if(temp < 0) {
50         putchar_unlocked('-');
51         write(-temp, end);
52         return;
53     }
54     P = -1;
55     while(temp) {
56         digits[++P] = temp % 10;
57         temp /= 10;
58     }
59     while(P >= 0) {
60         putchar_unlocked(digits[P--] + '0');
61     }
62     putchar_unlocked(end);
63     return;
64 }

```

5.2 Debug

```

1 #ifdef LOCAL
2     #define eprintf(...) { fprintf(stderr, __VA_ARGS__); fflush(
3         stderr); }
4 #else
5     #define eprintf(...) 42
6 #endif

```

5.3 Discrete

```

1 template<class T>
2 vector<int> Discrete(const vector<T>&v){
3     vector<int>ans;
4     vector<T>tmp(v);
5     sort(begin(tmp),end(tmp));
6     tmp.erase(unique(begin(tmp),end(tmp)),end(tmp));
7     for(auto i:v)ans.push_back(lower_bound(begin(tmp),end(tmp),i)
8         -tmp.begin()+1);
9     return ans;
10 }

```

5.4 DuiPai

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(){
4     string sol,bf,make;
5     cout<<"Your solution file name :";
6     cin>>sol;
7     cout<<"Brute force file name :";
8     cin>>bf;
9     cout<<"Make data file name :";
10    cin>>make;
11    system(("g++ "+sol+" -o sol").c_str());
12    system(("g++ "+bf+" -o bf").c_str());
13    system(("g++ "+make+" -o make").c_str());
14    for(int t = 0;t<10000;++t){
15        system("./make > ./1.in");
16        double st = clock();
17        system("./sol < ./1.in > ./1.ans");
18        double et = clock();
19        system("./bf < ./1.in > ./1.out");
20        if(system("diff ./1.out ./1.ans")) {
21            printf("\033[0;31mWrong Answer\033[0m on test #%d",t);
22            return 0;
23        }
24        else if(et-st>=2000){
25            printf("\033[0;32mTime Limit exceeded\033[0m on test #%d,
26                Time %.0lfms\n",t,et-st);
27            return 0;
28        }
29        else {
30            printf("\033[0;32mAccepted\033[0m on test #%d, Time
31                %.0lfms\n", t, et - st);
32        }
33    }
34 }

```

5.5 Timer

```

1 const clock_t startTime = clock();
2 inline double getCurrentTime() {
3     return (double) (clock() - startTime) / CLOCKS_PER_SEC;
4 }

```

5.6 TernarySearch

```

1 // return the maximum of $f(x)$ in $[l, r]$
2 double ternary_search(double l, double r) {
3     while(r - l > EPS) {
4         double m1 = l + (r - l) / 3;
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1), f2 = f(m2);
7         if(f1 < f2) l = m1;
8         else r = m2;
9     }
10    return f(l);
11 }
12
13 // return the maximum of $f(x)$ in $[l, r]$
14 int ternary_search(int l, int r) {
15     while(r - l > 1) {
16         int mid = (l + r) / 2;
17         if(f(m) > f(m + 1)) r = m;
18         else l = m;
19     }
20    return r;
21 }

```

6 Other

6.1 TouristIO

```

1 static struct FastInput {
2     static constexpr int BUF_SIZE = 1 << 20;
3     char buf[BUF_SIZE];
4     size_t chars_read = 0;
5     size_t buf_pos = 0;
6     FILE *in = stdin;
7     char cur = 0;
8
9     inline char get_char() {
10        if(buf_pos >= chars_read) {
11            chars_read = fread(buf, 1, BUF_SIZE, in);
12            buf_pos = 0;
13            buf[0] = (chars_read == 0 ? -1 : buf[0]);
14        }
15        return cur = buf[buf_pos++];
16        // return cur = getchar_unlocked();
17    }
18
19    inline void tie(int) {}
20
21    inline explicit operator bool() {
22        return cur != -1;
23    }
24
25    inline static bool is_blank(char c) {
26        return c <= ' ';
27    }
28
29    inline bool skip_blanks() {
30        while(is_blank(cur) && cur != -1) {
31            get_char();
32        }
33        return cur != -1;
34    }
35
36    inline FastInput& operator>>(char& c) {
37        skip_blanks();
38        c = cur;
39        return *this;
40    }
41
42    inline FastInput& operator>>(string& s) {
43        if(skip_blanks()) {
44            s.clear();
45            do {
46                s += cur;
47            } while(!is_blank(get_char()));
48        }
49        return *this;
50    }
51 }

```

```

52 template<class T>
53 inline FastInput& read_integer(T& n) {
54     // unsafe, doesn't check that characters are actually
55     // digits
56     n = 0;
57     if(skip_blanks()) {
58         int sign = +1;
59         if(cur == '-') {
60             sign = -1;
61             get_char();
62         }
63         do {
64             n += n + (n << 3) + cur - '0';
65         } while(!is_blank(get_char()));
66         n *= sign;
67     }
68     return *this;
69 }
70 template<class T>
71 inline typename enable_if<is_integral<T>::value, FastInput
72     &>::type operator>>(T& n) {
73     return read_integer(n);
74 }
75 #if!defined(_WIN32) || defined(_WIN64)
76 inline FastInput& operator>>(__int128& n) {
77     return read_integer(n);
78 }
79 #endif
80
81 template<class T>
82 inline typename enable_if<is_floating_point<T>::value,
83     FastInput&>::type operator>>(T& n) {
84     // not sure if really fast, for compatibility only
85     n = 0;
86     if(skip_blanks()) {
87         string s;
88         (*this) >> s;
89         sscanf(s.c_str(), "%lf", &n);
90     }
91     return *this;
92 }
93 } fast_input;
94
95 #define cin fast_input
96
97 static struct FastOutput {
98     static constexpr int BUF_SIZE = 1 << 20;
99     char buf[BUF_SIZE];
100     size_t buf_pos = 0;
101     static constexpr int TMP_SIZE = 1 << 20;
102     char tmp[TMP_SIZE];
103     FILE *out = stdout;
104
105     inline void put_char(char c) {
106         buf[buf_pos++] = c;
107         if(buf_pos == BUF_SIZE) {
108             fwrite(buf, 1, buf_pos, out);
109             buf_pos = 0;
110         }
111     }
112     // putchar_unlocked(c);
113 }
114 ~FastOutput() {
115     fwrite(buf, 1, buf_pos, out);
116 }
117
118 inline FastOutput& operator<<(char c) {
119     put_char(c);
120     return *this;
121 }
122
123 inline FastOutput& operator<<(const char* s) {
124     while(*s) {
125         put_char(*s++);
126     }
127     return *this;
128 }
129
130 inline FastOutput& operator<<(const string& s) {
131     for(int i = 0; i < (int) s.size(); i++) {
132         put_char(s[i]);
133     }
134     return *this;
135 }
136
137 template<class T>
138 inline char* integer_to_string(T n) {
139     // beware of TMP_SIZE
140     char* p = tmp + TMP_SIZE - 1;
141     if(n == 0) {
142         *--p = '0';
143     } else {
144         bool is_negative = false;
145         if(n < 0) {
146             is_negative = true;

```

```

146         n = -n;
147     }
148     while(n > 0) {
149         *--p = (char) ('0' + n % 10);
150         n /= 10;
151     }
152     if(is_negative) {
153         *--p = '-';
154     }
155 }
156 return p;
157 }
158
159 template<class T>
160 inline typename enable_if<is_integral<T>::value, char*>::type
161     stringify(T n) {
162     return integer_to_string(n);
163 }
164
165 #if!defined(_WIN32) || defined(_WIN64)
166 inline char* stringify(__int128 n) {
167     return integer_to_string(n);
168 }
169 #endif
170
171 template<class T>
172 inline typename enable_if<is_floating_point<T>::value, char
173     *>::type stringify(T n) {
174     sprintf(tmp, "%.17f", n);
175     return tmp;
176 }
177
178 template<class T>
179 inline FastOutput& operator<<(const T& n) {
180     auto p = stringify(n);
181     for(; *p != 0; p++) {
182         put_char(*p);
183     }
184     return *this;
185 }
186 } fast_output;
187
188 #define cout fast_output

```

7 Setup

7.1 Template

```

1 #include <bits/extc++.h>
2 #include <bits/stdc++.h>
3 #pragma gcc optimize("ofast, unroll-loops, no-stack-protector,
4     fast-math")
5 #define IOS ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
6 #define int long long
7 #define double long double
8 #define pb push_back
9 #define sz(x) (int)(x).size()
10 #define all(v) begin(v), end(v)
11 #define debug(x) cerr<<"x<<" = "<<x<<"\n"
12 #define LINE cout<<"\n-----\n"
13 #define endl '\n'
14 #define VI vector<int>
15 #define F first
16 #define S second
17 #define MP(a,b) make_pair(a,b)
18 #define rep(i,m,n) for(int i = m; i<=n; ++i)
19 #define res(i,m,n) for(int i = m; i>=n; --i)
20 #define gcd(a,b) __gcd(a,b)
21 #define lcm(a,b) a*b/gcd(a,b)
22 #define Case() int _; cin>>_; for(int Case = 1; Case<=_; ++Case)
23 #define pii pair<int,int>
24 using namespace __gnu_cxx;
25 using namespace __gnu_pbds;
26 using namespace std;
27 template <typename K, typename cmp = less<K>, typename T =
28     thin_heap_tag> using _heap = __gnu_pbds::priority_queue<K,
29     cmp, T>;
30 template <typename K, typename M = null_type> using _hash =
31     gp_hash_table<K, M>;
32 const int N = 1e6+5, L = 20, mod = 1e9+7;
33 const long long inf = 2e18+5;
34 const double eps = 1e-7, pi = acos(-1);
35 mt19937 mt(std::chrono::system_clock::now().time_since_epoch().
36     count());
37 void solve(){
38 }
39
40 signed main(){
41     IOS;
42     solve();
43 }

```

8 String

8.1 DynamicKMP

```

1 template<int ALPHABET, int (*)(char)>
2 class DynamicKMP {
3 public:
4     DynamicKMP() {}
5
6     DynamicKMP(const string& s) {
7         reserve(s.size());
8         for(const char& c : s) {
9             push(c);
10        }
11    }
12
13    void push(char c) {
14        int v = f(c);
15        dp.emplace_back();
16        dp.back()[v] = (int) dp.size();
17        if(p.empty()) {
18            p.push_back(0);
19            return;
20        }
21        int i = (int) p.size();
22        for(int j = 0; j < ALPHABET; ++j) {
23            if(j == v) {
24                p.push_back(dp[p[i - 1]][j]);
25            } else {
26                dp.back()[j] = dp[p[i - 1]][j];
27            }
28        }
29    }
30
31    void pop() {
32        p.pop_back();
33        dp.pop_back();
34    }
35
36    int query() const {
37        return p.back();
38    }
39
40    vector<int> query_all() const {
41        return p;
42    }
43
44    void reserve(int sz) {
45        p.reserve(sz);
46        dp.reserve(sz);
47    }
48
49 private:
50     vector<int> p;
51     vector<array<int, ALPHABET>> dp;
52 };

```

8.2 RollingHash

```

1 template<int HASH_COUNT, bool PRECOMPUTE_POWERS = false>
2 class Hash {
3 public:
4     static constexpr int MAX_HASH_PAIRS = 10;
5
6     // {mul, mod}
7     static constexpr const pair<int, int> HASH_PAIRS[] =
8     {
9         {827167801, 999999937}, {998244353, 999999929},
10        {146672737, 922722049}, {204924373, 952311013},
11        {585761567, 955873937}, {484547929, 901981687},
12        {856009481, 987877511}, {852853249, 996724213},
13        {937381759, 994523539}, {116508269, 993179543}};
14
15    Hash() : Hash("") {}
16
17    Hash(const string& s) : n(s.size()) {
18        static_assert(HASH_COUNT > 0 && HASH_COUNT <=
19            MAX_HASH_PAIRS);
20        for(int i = 0; i < HASH_COUNT; ++i) {
21            const auto& p = HASH_PAIRS[i];
22            pref[i].resize(n);
23            pref[i][0] = s[0];
24            for(int j = 1; j < n; ++j) {
25                pref[i][j] = (1LL * pref[i][j - 1] * p.first + s[j]) %
26                    p.second;
27            }
28        }
29        if(PRECOMPUTE_POWERS) {
30            build_powers(n);
31        }
32    }
33
34    void build_powers(int n) {
35        for(int i = 0; i < HASH_COUNT; ++i) {
36            const auto& p = HASH_PAIRS[i];
37            for(int j = 1; j < n; ++j) {
38                pow[i][j] = (1LL * pow[i][j - 1] * p.first + p.second) %
39                    p.second;
40            }
41        }
42    }
43
44    int substr(int l, int r) const {
45        int res = 0;
46        for(int i = 0; i < HASH_COUNT; ++i) {
47            const auto& p = HASH_PAIRS[i];
48            res = (1LL * res * p.first + pref[i][r] - pref[i][l] * pow[i][r - l + 1]) %
49                p.second;
50        }
51        return res;
52    }
53
54    int get_power(int a, int b) const {
55        int res = 1;
56        while(b) {
57            if(b & 1) res = (1LL * res * p.first + p.second) % p.second;
58            b >>= 1;
59        }
60        return res;
61    }
62
63    template<int A, bool B> vector<int> Hash<A, B>::POW[Hash::
64        MAX_HASH_PAIRS];
65
66 private:
67     int n;
68     vector<array<int, HASH_COUNT>> pref;
69     vector<array<int, HASH_COUNT>> pow;
70 };

```

```

26 void add_char(char c) {
27     for(int i = 0; i < HASH_COUNT; ++i) {
28         const auto& p = HASH_PAIRS[i];
29         pref[i].push_back((1LL * (n == 0 ? 0 : pref[i].back()) *
30             p.first + c) % p.second);
31     }
32     n += 1;
33     if(PRECOMPUTE_POWERS) {
34         build_powers(n);
35     }
36 }
37
38 // Return hash values for [l, r)
39 array<int, HASH_COUNT> substr(int l, int r) {
40     array<int, HASH_COUNT> res{};
41     for(int i = 0; i < HASH_COUNT; ++i) {
42         res[i] = substr(i, l, r);
43     }
44     return res;
45 }
46
47 array<int, HASH_COUNT> merge(const vector<pair<int, int>>&
48     seg) {
49     array<int, HASH_COUNT> res{};
50     for(int i = 0; i < HASH_COUNT; ++i) {
51         const auto& p = HASH_PAIRS[i];
52         for(auto [l, r] : seg) {
53             res[i] = (1LL * res[i] * get_power(i, r - l + 1) + substr(i,
54                 l, r)) % p.second;
55         }
56     }
57     return res;
58 }
59
60 // build powers up to x^k
61 void build_powers(int k) {
62     for(int i = 0; i < HASH_COUNT; ++i) {
63         const auto& p = HASH_PAIRS[i];
64         int sz = (int) POW[i].size();
65         if(sz > k) {
66             continue;
67         }
68         if(sz == 0) {
69             POW[i].push_back(1);
70             sz = 1;
71         }
72         while(sz <= k) {
73             POW[i].push_back(1LL * POW[i].back() * p.first % p.
74                 second);
75             sz += 1;
76         }
77     }
78 }
79
80 inline int size() const {
81     return n;
82 }
83
84 private:
85     int n;
86     static vector<int> POW[MAX_HASH_PAIRS];
87     array<vector<int>, HASH_COUNT> pref;
88
89     int substr(int k, int l, int r) {
90         assert(0 <= k && k < HASH_COUNT);
91         assert(0 <= l && l <= r && r <= n);
92         const auto& p = HASH_PAIRS[k];
93         if(l == r) {
94             return 0;
95         }
96         int res = pref[k][r - 1];
97         if(l > 0) {
98             res -= 1LL * pref[k][l - 1] * get_power(k, r - l + 1) % p.
99                 second;
100         }
101         if(res < 0) {
102             res += p.second;
103         }
104         return res;
105     }
106
107     int get_power(int a, int b) {
108         if(PRECOMPUTE_POWERS) {
109             build_powers(b);
110             return POW[a][b];
111         }
112         const auto& p = HASH_PAIRS[a];
113         return power(p.first, b, p.second);
114     }
115 };
116
117 template<int A, bool B> vector<int> Hash<A, B>::POW[Hash::
118     MAX_HASH_PAIRS];

```

8.3 SuffixArray

```

1 struct suffix_array{
2     int n;
3     vector<int>SA, Rank, LCP;
4     void counting_sort(vector<int>&v, auto getkey){
5         int n = 0;
6         for(auto i:v)n = max(n, getkey(i)+1);
7         vector<int>bucket(n), ans(v.size());
8         for(auto i:v)++bucket[getkey(i)];
9         partial_sum(begin(bucket), end(bucket), begin(bucket));
10        for(auto ite = v.rbegin(); ite!=v.rend(); ++ite)ans[--bucket[
            getkey(*ite)]] = move(*ite);
11        v.swap(ans);
12        return;
13    }
14    suffix_array(string s):n(s.size()){
15        SA.resize(n), Rank.resize(n), LCP.resize(n);
16        for(int i = 0; i<n; ++i)SA[i] = i;
17        sort(SA.begin(), SA.end(), [&](int a, int b){
18            return s[a]<s[b];
19        });
20        for(int i = 0; i<n; ++i){
21            Rank[SA[i]] = (i?Rank[SA[i-1]]+(s[SA[i]]!=s[SA[i-1]]):SA
                [0]);
22        }
23        for(int k = 0; (1<<k)<=n; ++k){
24            vector<int>idx;
25            for(int i = n-(1<<k); i<n; ++i)idx.push_back(i);
26            for(auto i:SA)if(i>=(1<<k))idx.push_back(i-(1<<k));
27            counting_sort(idx, [&](int a){return Rank[a];});
28            SA.swap(idx);
29            vector<int>new_rank(n);
30            new_rank[SA[0]] = 0;
31            for(int i = 1; i<n; ++i){
32                auto cmp = [&](int a, int b){
33                    return Rank[a]!=Rank[b] or a+(1<<k)>=n or Rank[a+(1<<
                        k)]!=Rank[b+(1<<k)];
34                };
35                new_rank[SA[i]] = new_rank[SA[i-1]]+cmp(SA[i-1], SA[i]);
36            }
37            Rank.swap(new_rank);
38        }
39        for(int i = 0, k = 0; i<n; ++i){
40            if(Rank[i]==0)continue;
41            if(k)--k;
42            while(i+k<n and SA[Rank[i]-1]+k<n and s[i+k]==s[SA[Rank[i]
                ]-1+k])++k;
43            LCP[Rank[i]] = k;
44        }
45    }
46 };

```

8.4 Trie

```

1 template<int ALPHABET = 26, char MIN_CHAR = 'a'>
2 class trie {
3 public:
4     struct Node {
5         int go[ALPHABET];
6         Node() {
7             memset(go, -1, sizeof(go));
8         }
9     };
10
11    trie() {
12        newNode();
13    }
14
15    inline int next(int p, int v) {
16        return nodes[p].go[v] != -1 ? nodes[p].go[v] : nodes[p].go[
            v] = newNode();
17    }
18
19    inline void insert(const vector<int>& a, int p = 0) {
20        for(int v : a) {
21            p = next(p, v);
22        }
23    }
24
25    inline void clear() {
26        nodes.clear();
27        newNode();
28    }
29
30    inline int longest_common_prefix(const vector<int>& a, int p
        = 0) const {
31        int ans = 0;
32        for(int v : a) {
33            if(nodes[p].go[v] != -1) {
34                ans += 1;
35                p = nodes[p].go[v];
36            } else {

```


ACM ICPC Team

Reference -

LeeJiaHuaPlayMinecraft

Contents

1	Data-Structure	1
1.1	Treap	1
1.2	Segtree	1
1.3	DsuUndo	1
1.4	DSU	2
1.5	Fenwick	2
1.6	Persistent DSU	2
1.7	TimingSegtree	2
1.8	AreaOfRectangles	2
1.9	SparseTable	3
1.10	DynamicSegtree	3
1.11	ZkwSegtree	3
1.12	MoAlgo	4
1.13	Hash	4
1.14	RedBlackTree	4
2	Geometry	4
2.1	Theorem	4
2.2	PointInPolygon	4
2.3	PointInConvex	4
2.4	MaximumDistance	5
2.5	PolarAngleSort	5
2.6	MinimumDistance	5
2.7	ConvexHull	5
2.8	Template	5
3	Graph	6
3.1	HLD	6
3.2	Bridges	6
3.3	MCMF	6
3.4	LCA	7
3.5	CentroidDecomposition	7
3.6	BCC AP	7
3.7	SCC	7
3.8	Dinic	8
4	Math	8
4.1	Numbers	8
4.2	ExtendGCD	8
4.3	InvGCD	8
4.4	GeneratingFunctions	9
4.5	Theorem	9
4.6	FloorSum	9
4.7	Factorizer	9
4.8	PowMod	10
4.9	CRT	10
4.10	DiscreteLog	10
4.11	LinearSieve	10
5	Misc	10
5.1	FastIO	10
5.2	Debug	11
5.3	Discrete	11
5.4	DuiPai	11
5.5	Timer	11
5.6	TenarySearch	11
6	Other	11
6.1	TouristIO	11
7	Setup	12
7.1	Template	12
8	String	13
8.1	DynamicKMP	13
8.2	RollingHash	13
8.3	SuffixArray	14
8.4	Trie	14

ACM ICPC Judge Test - LeeJiaHuaPlayMinecraft

C++ Resource Test

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 namespace system_test {
5
6     const size_t KB = 1024;
7     const size_t MB = KB * 1024;
8     const size_t GB = MB * 1024;
9
10    size_t block_size, bound;
11    void stack_size_dfs(size_t depth = 1) {
12        if (depth >= bound)
13            return;
14        int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
15        memset(ptr, 'a', block_size);
16        cout << depth << endl;
17        stack_size_dfs(depth + 1);
18    }
19
20    void stack_size_and_runtime_error(size_t block_size, size_t
        bound = 1024) {
21        system_test::block_size = block_size;
22        system_test::bound = bound;
23        stack_size_dfs();
24    }
25
26    double speed(int iter_num) {
27        const int block_size = 1024;
28        volatile int A[block_size];
29        auto begin = chrono::high_resolution_clock::now();
30        while (iter_num--)
31            for (int j = 0; j < block_size; ++j)
32                A[j] += j;
33        auto end = chrono::high_resolution_clock::now();
34        chrono::duration<double> diff = end - begin;
35        return diff.count();
```

```
36    }
37
38    void runtime_error_1() {
39        // Segmentation fault
40        int *ptr = nullptr;
41        *(ptr + 7122) = 7122;
42    }
43
44    void runtime_error_2() {
45        // Segmentation fault
46        int *ptr = (int *)memset;
47        *ptr = 7122;
48    }
49
50    void runtime_error_3() {
51        // munmap_chunk(): invalid pointer
52        int *ptr = (int *)memset;
53        delete ptr;
54    }
55
56    void runtime_error_4() {
57        // free(): invalid pointer
58        int *ptr = new int[7122];
59        ptr += 1;
60        delete[] ptr;
61    }
62
63    void runtime_error_5() {
64        // maybe illegal instruction
65        int a = 7122, b = 0;
66        cout << (a / b) << endl;
67    }
68
69    void runtime_error_6() {
70        // floating point exception
71        volatile int a = 7122, b = 0;
72        cout << (a / b) << endl;
73    }
74
75    void runtime_error_7() {
76        // call to abort.
77        assert(false);
78    }
79
80    } // namespace system_test
81
82    #include <sys/resource.h>
83    void print_stack_limit() { // only work in Linux
84        struct rlimit l;
85        getrlimit(RLIMIT_STACK, &l);
86        cout << "stack_size = " << l.rlim_cur << " byte" << endl;
87    }
```