

1 Convolution

1.1 FFT

```

1 typedef complex<double> cd;
2 const double PI = acos(-1);
3
4 void FFT(vector<cd> &a, bool inv){
5
6     int n = a.size();
7
8     for (int i=1, j=0 ; i<n ; i++){
9         int bit = (n>>1);
10        for ( ; j&bit ; bit>>=1){
11            j ^= bit;
12        }
13        j ^= bit;
14        if (i<j){
15            swap(a[i], a[j]);
16        }
17    }
18
19    for (int len=2 ; len<=n ; len<=1){
20        cd wlen = polar(1.0, (inv ? 2 : -2)*PI/len);
21
22        for (int i=0 ; i<n ; i+=len){
23            cd w(1);
24            for (int j=0 ; j<len/2 ; j++){
25                cd u = a[i+j];
26                cd v = a[i+j+len/2]*w;
27                a[i+j] = u+v;
28                a[i+j+len/2] = u-v;
29                w *= wlen;
30            }
31        }
32    }
33
34    if (inv){
35        for (auto &x : a){
36            x /= n;
37        }
38    }
39
40    return;
41 }
42
43 vector<cd> polyMul(vector<cd> a, vector<cd> b){
44     int sa = a.size(), sb = b.size(), n = 1;
45
46     while (n<sa+sb-1) n *= 2;
47     a.resize(n);
48     b.resize(n);
49     vector<cd> c(n);
50
51     FFT(a, 0);
52     FFT(b, 0);
53     for (int i=0 ; i<n ; i++) c[i] = a[i]*b[i];
54     FFT(c, 1);
55
56     c.resize(sa+sb-1);
57
58     return c;
59 }

```

1.2 NTT-998244353

```

1 const int MOD = (119 << 23) + 1, ROOT = 62; // = 998244353
2 // For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 <<
3 // 21 and 483 << 21 (same root). The last two are > 10^9.
4
5 void NTT(vector<int> &a) {
6     int n = a.size();
7     int L = 31-__builtin_clz(n);
8     vector<int> rt(2, 1);
9     for (int k=2, s=2 ; k<n ; k*=2, s++){
10        rt.resize(n);
11        int z[] = {1, qp(ROOT, MOD>>s)};
12        for (int i=k ; i<2*k ; i++){
13            rt[i] = rt[i/2]*z[i&1]%MOD;
14        }
15    }
16
17    vector<int> rev(n);
18    for (int i=0 ; i<n ; i++){
19        rev[i] = (rev[i/2] | (i&1)<<(L)/2);
20    }
21
22    for (int i=0 ; i<n ; i++){
23        if (i<rev[i]){
24            swap(a[i], a[rev[i]]);
25        }
26    }
27
28    for (int k=1 ; k<n ; k*=2){
29        for (int i=0 ; i<n ; i+=2*k){
30            for (int j=0 ; j<k ; j++){
31                int z = rt[j+k]*a[i+j+k]%MOD, &ai = a[i+j];
32                a[i+j+k] = ai-z+(z>ai ? MOD : 0);
33                ai += (ai+z>MOD ? z-MOD : z);
34            }
35        }
36    }
37
38    vector<int> polyMul(vector<int> &a, vector<int> &b){
39        if (a.empty() || b.empty()) return {};
40        int s = a.size()+b.size()-1, B = 32-__builtin_clz(s), n =
41            1<<B;
42        int inv = qp(n, MOD-2);
43
44        vector<int> L(a), R(b), out(n);
45        L.resize(n), R.resize(n);
46        NTT(L), NTT(R);
47        for (int i=0 ; i<n ; i++){
48            out[-i&(n-1)] = L[i]*R[i]%MOD*inv%MOD;
49        }
50        NTT(out);
51
52        out.resize(s);
53        return out;
54 }

```

1.3 FFT-mod

```

1 /*
2 修改 const int MOD = 998244353 更改要取餘的數字

```

```

3 PolyMul(a, b) 回傳多項式乘法的結果 ( c_k = \sum_{i+j=k} a_i b_j
4     mod MOD )
5
6 大約可以支援 5e5 · a_i, b_i 皆在 MOD 以下的非負整數
7
8 const int MOD = 998244353;
9 typedef complex<double> cd;
10
11 void FFT(vector<cd> &a) {
12     int n = a.size(), L = 31-__builtin_clz(n);
13     vector<complex<long double>> R(2, 1);
14     vector<cd> rt(2, 1);
15     for (int k=2 ; k<n ; k*=2){
16         R.resize(n);
17         rt.resize(n);
18         auto x = polar(1.0L, acos(-1.0L) / k);
19         for (int i=k ; i<2*k ; i++){
20             rt[i] = R[i] = (i&1 ? R[i/2]*x : R[i/2]);
21         }
22     }
23
24     vector<int> rev(n);
25     for (int i=0 ; i<n ; i++){
26         rev[i] = (rev[i/2] | (i&1)<<(L)/2);
27     }
28
29     for (int i=0 ; i<n ; i++){
30         if (i<rev[i]) swap(a[i], a[rev[i]]);
31     }
32
33     for (int k=1 ; k<n ; k*=2){
34         for (int i=0 ; i<n ; i+=2*k){
35             for (int j=0 ; j<k ; j++){
36                 auto x = (double *)&rt[j+k];
37                 auto y = (double *)&a[i+j+k];
38                 cd z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*
39                     y[0]);
40                 a[i+j+k] = a[i+j]-z;
41                 a[i+j] += z;
42             }
43         }
44     }
45
46     return;
47 }
48
49 vector<int> PolyMul(vector<int> a, vector<int> b){
50     if (a.empty() || b.empty()) return {};
51
52     vector<int> res(a.size()+b.size()-1);
53     int B = 32-__builtin_clz(res.size()), n = (1<<B), cut =
54         int(sqrt(MOD));
55     vector<cd> L(n), R(n), outs(n), outl(n);
56
57     for (int i=0 ; i<a.size() ; i++){
58         L[i] = cd((int) a[i]/cut, (int) a[i]%cut);
59     }
60
61     for (int i=0 ; i<b.size() ; i++){
62         R[i] = cd((int) b[i]/cut, (int) b[i]%cut);
63     }
64
65     FFT(L);
66     FFT(R);
67     for (int i=0 ; i<n ; i++){
68         int j = -i&(n-1);
69         outl[j] = (L[i]+conj(L[j])) * R[i]/(2.0*n);
70         outs[j] = (L[i]-conj(L[j])) * R[i]/(2.0*n)/1i;
71     }
72
73     FFT(outl);

```

```

65 FFT(out);
66 for (int i=0 ; i<res.size() ; i++){
67     int av = (int)(real(outl[i])+0.5), cv = (int)(imag(
        outs[i])+0.5);
68     int bv = (int)(imag(outl[i])+0.5) + (int)(real(outs[i]
        ])+0.5);
69     res[i] = ((av%MOD*cut+bv) % MOD*cut+cv) % MOD;
70 }
71 return res;
72 }
73 }

```

1.4 FFT-2

```

1 typedef complex<double> cd;
2
3 void FFT(vector<cd> &a) {
4     int n = a.size(), L = 31-__builtin_clz(n);
5     vector<complex<long double>> R(2, 1);
6     vector<cd> rt(2, 1);
7     for (int k=2 ; k<n ; k*=2){
8         R.resize(n);
9         rt.resize(n);
10        auto x = polar(1.0L, acos(-1.0L) / k);
11        for (int i=k ; i<2*k ; i++){
12            rt[i] = R[i] = (i&1 ? R[i/2]*x : R[i/2]);
13        }
14    }
15
16    vector<int> rev(n);
17    for (int i=0 ; i<n ; i++){
18        rev[i] = (rev[i/2] | (i&1)<<L)/2;
19    }
20    for (int i=0 ; i<n ; i++){
21        if (i<rev[i]) swap(a[i], a[rev[i]]);
22    }
23    for (int k=1 ; k<n ; k*=2){
24        for (int i=0 ; i<n ; i+=2*k){
25            for (int j=0 ; j<k ; j++){
26                auto x = (double *)&rt[j+k];
27                auto y = (double *)&a[i+j+k];
28                cd z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*
                    y[0]);
29                a[i+j+k] = a[i+j]-z;
30                a[i+j] += z;
31            }
32        }
33    }
34    return;
35 }
36
37 vector<double> PolyMul(const vector<double> a, const vector<
    double> b){
38     if (a.empty() || b.empty()) return {};
39     vector<double> res(a.size()+b.size()-1);
40     int L = 32 - __builtin_clz(res.size()), n = 1 << L;
41     vector<cd> in(n), out(n);
42
43     copy(a.begin(), a.end(), begin(in));
44     for (int i=0 ; i<b.size() ; i++){
45         in[i].imag(b[i]);
46     }
47     FFT(in);

```

```

48     for (cd& x : in) x *= x;
49     for (int i=0 ; i<n ; i++){
50         out[i] = in[-i & (n - 1)] - conj(in[i]);
51     }
52     FFT(out);
53
54     for (int i=0 ; i<res.size() ; i++){
55         res[i] = imag(out[i]) / (4 * n);
56     }
57
58     return res;
59 }

```

2 Data-Structure

2.1 GP-Hash-Table

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> order_set;
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().time_since_epoch().count();
15        return splitmix64(x + FIXED_RANDOM);
16    }
17 };
18
19 gp_hash_table<int, int, custom_hash> ss;

```

2.2 Sparse-Table

```

1 vector<vector<int>> st;
2 void build(vector<int> v){
3     int h = __lg(v.size());
4     st.resize(h+1);
5     st[0] = v;
6
7     for (int i=1 ; i<=h ; i++){
8         int gap = (1<<(i-1));
9         for (int j=0 ; j+gap<st[i-1].size() ; j++){
10            st[i].push_back(min(st[i-1][j], st[i-1][j+gap]));
11        }
12    }
13 }
14
15 // 回傳 [ll, rr) 的最小值
16 int RMQ(int ll, int rr){
17     int h = __lg(rr-ll);

```

```

18     return min(st[h][ll], st[h][rr-(1<<h)]);
19 }

```

2.3 Order-Set

```

1 // .find_by_order(k) 回傳第 k 小的值 (based-0)
2 // .order_of_key(k) 回傳有多少元素比 k 小
3
4 #include <ext/pb_ds/tree_policy.hpp>
5 using namespace __gnu_pbds;
6 typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> order_set;

```

2.4 BIT

```

1 vector<int> BIT(MAX_SIZE);
2 void update(int pos, int val){
3     for (int i=pos ; i<MAX_SIZE ; i+=i&-i){
4         BIT[i]+=val;
5     }
6 }
7
8 int query(int pos){
9     int ret=0;
10    for (int i=pos ; i>0 ; i-=i&-i){
11        ret+=BIT[i];
12    }
13    return ret;
14 }
15
16 // const int MAX_N = (1<<20)
17 // const int LOG_N = 20;
18 int k_th(int k){ // 回傳 BIT 中第 k 小的元素 (based-1)
19     int target = k-1, now = 0;
20     for (int i=LOG_N-1 ; i>=0 ; i--){
21         if (BIT[now+(1<<i)]<=k){
22             k -= BIT[now+(1<<i)];
23             now += 1<<i;
24         }
25     }
26     return now+1;
27 }

```

2.5 Add-Set-Segment-Tree

```

1 // [ll, rr), based-0
2 // 使用前記得 init(陣列大小), build(陣列名稱)
3 // add(ll, rr): 區間修改
4 // set(ll, rr): 區間賦值
5 // query(ll, rr): 區間求和 / 求最大值
6 struct segment_tree{
7     struct node{
8         int add_tag=0;
9         int set_tag=0;
10        int sum=0;

```

```

11     int ma=0;
12 };
13
14 vector<node> arr;
15
16 void init(int n){
17     arr.resize(n<<2);
18 }
19
20 node pull(node A, node B){
21     node C;
22     C.sum=A.sum+B.sum;
23     C.ma=max(A.ma, B.ma);
24     return C;
25 }
26
27 void push(int idx, int ll, int rr){
28     if (arr[idx].set_tag>0){
29         // set 優先實作
30         arr[idx].sum=(rr-ll)*arr[idx].set_tag;
31         arr[idx].ma=arr[idx].set_tag;
32
33         if (rr-ll>1){
34             arr[idx*2+1].add_tag=0;
35             arr[idx*2+1].set_tag=arr[idx].set_tag;
36             arr[idx*2+2].add_tag=0;
37             arr[idx*2+2].set_tag=arr[idx].set_tag;
38         }
39
40         arr[idx].set_tag=0;
41     }
42     if (arr[idx].add_tag>0){
43         // add 次要實作
44         arr[idx].sum+=(rr-ll)*arr[idx].add_tag;
45         arr[idx].ma+=arr[idx].add_tag;
46
47         if (rr-ll>1){
48             arr[idx*2+1].add_tag+=arr[idx].add_tag;
49             arr[idx*2+2].add_tag+=arr[idx].add_tag;
50         }
51
52         arr[idx].add_tag=0;
53     }
54 }
55
56 void build(vector<int> &v, int idx=0, int ll=0, int rr=n)
57 {
58     if (rr-ll==1){
59         arr[idx].sum=v[ll];
60         arr[idx].ma=v[ll];
61     }else{
62         int mid=(ll+rr)/2;
63         build(v, idx*2+1, ll, mid);
64         build(v, idx*2+2, mid, rr);
65         arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
66     }
67 }
68
69 void add(int ql, int qr, int val, int idx=0, int ll=0,
70         int rr=n){
71     push(idx, ll, rr);
72     if (rr<=ql || qr<=ll) return;
73     if (ql<=ll && rr<=qr){
74         arr[idx].add_tag+=val;
75         push(idx, ll, rr);
76     }
77     int mid=(ll+rr)/2;
78     add(ql, qr, val, idx*2+1, ll, mid);
79     add(ql, qr, val, idx*2+2, mid, rr);
80     arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
81 }
82
83 void set(int ql, int qr, int val, int idx=0, int ll=0,
84         int rr=n){
85     push(idx, ll, rr);
86     if (rr<=ql || qr<=ll) return;
87     if (ql<=ll && rr<=qr){
88         arr[idx].add_tag=0;
89         arr[idx].set_tag=val;
90         push(idx, ll, rr);
91         return;
92     }
93     int mid=(ll+rr)/2;
94     set(ql, qr, val, idx*2+1, ll, mid);
95     set(ql, qr, val, idx*2+2, mid, rr);
96     arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
97 }
98
99 node query(int ql, int qr, int idx=0, int ll=0, int rr=n)
100 {
101     push(idx, ll, rr);
102     if (rr<=ql || qr<=ll) return node();
103     if (ql<=ll && rr<=qr) return arr[idx];
104
105     int mid=(ll+rr)/2;
106     return pull(query(ql, qr, idx*2+1, ll, mid), query(ql,
107         qr, idx*2+2, mid, rr));
108 }
109 } ST;

```

```

74     return;
75 }
76 int mid=(ll+rr)/2;
77 add(ql, qr, val, idx*2+1, ll, mid);
78 add(ql, qr, val, idx*2+2, mid, rr);
79 arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
80 }
81
82 void set(int ql, int qr, int val, int idx=0, int ll=0,
83         int rr=n){
84     push(idx, ll, rr);
85     if (rr<=ql || qr<=ll) return;
86     if (ql<=ll && rr<=qr){
87         arr[idx].add_tag=0;
88         arr[idx].set_tag=val;
89         push(idx, ll, rr);
90         return;
91     }
92     int mid=(ll+rr)/2;
93     set(ql, qr, val, idx*2+1, ll, mid);
94     set(ql, qr, val, idx*2+2, mid, rr);
95     arr[idx]=pull(arr[idx*2+1], arr[idx*2+2]);
96 }
97
98 node query(int ql, int qr, int idx=0, int ll=0, int rr=n)
99 {
100     push(idx, ll, rr);
101     if (rr<=ql || qr<=ll) return node();
102     if (ql<=ll && rr<=qr) return arr[idx];
103
104     int mid=(ll+rr)/2;
105     return pull(query(ql, qr, idx*2+1, ll, mid), query(ql,
106         qr, idx*2+2, mid, rr));
107 }
108 } ST;

```

2.6 Treap

```

1 struct Treap{
2     Treap *l, *r;
3     int pri, val, sz;
4
5     Treap(int _val){
6         l = nullptr;
7         r = nullptr;
8         pri = rand();
9         val = _val;
10        sz = 1;
11    }
12 } *root;
13
14 int size(Treap *a){
15     return a ? a->sz : 0;
16 }
17
18 void pull(Treap *t){
19     t->sz = size(t->l)+size(t->r)+1;
20 }
21
22 Treap *merge(Treap *a, Treap *b){
23     // 如果一個為空，就回傳另一個
24     if (!a || !b) return a ? a : b;
25 }

```

```

26 if (a->pri>b->pri){
27     a->r = merge(a->r, b);
28     pull(a);
29     return a;
30 }else{
31     b->l = merge(a, b->l);
32     pull(b);
33     return b;
34 }
35
36 }
37
38 void split(Treap *t, int k, Treap *&a, Treap *&b){
39     // 如果樹為空就直接返回
40     if (!t) a = b = nullptr;
41
42     else if (size(t->l)+1<=k){ // 用 k 分割 treap
43         // 如果以左子節點為根 + 目前節點合法：
44         a = t;
45         split(t->r, k-size(t->l)-1, a->r, b);
46         pull(a);
47     }else{
48         b = t;
49         split(t->l, k, a, b->l);
50         pull(b);
51     }
52 }
53
54 ostream & operator << (ostream &os, Treap *t){
55     if (t==0) return os;
56     os << t->l;
57     os << (char)t->val;
58     os << t->r;
59     return os;
60 }
61
62 void print(Treap *t){
63     if (t->l!=0) print(t->l);
64     cout << (char)t->val;
65     if (t->r!=0) print(t->r);
66 }

```

2.7 Persistent-Segment-Tree

```

1 /*
2 全部都是 0-based
3
4 宣告
5 Persistent_Segment_Tree st(n+q);
6 st.build(v, 0);
7
8 函式：
9 update_version(pos, val, ver)：對版本 ver 的 pos 位置改成 val
10 query_version(ql, qr, ver)：對版本 ver 查詢 [ql, qr) 的區間和
11 clone_version(ver)：複製版本 ver 到最新的版本
12 */
13 struct Persistent_Segment_Tree{
14     int node_cnt = 0;
15     struct Node{
16         int lc = -1;
17         int rc = -1;

```

```

18     int val = 0;
19 };
20 vector<Node> arr;
21 vector<int> version;
22
23 Persistent_Segment_Tree(int sz){
24     arr.resize(32*sz);
25     version.push_back(node_cnt++);
26     return;
27 }
28
29 void pull(Node &c, Node a, Node b){
30     c.val = a.val+b.val;
31     return;
32 }
33
34 void build(vector<int> &v, int idx, int ll = 0, int rr =
35     n){
36     auto &now = arr[idx];
37
38     if (rr-ll==1){
39         now.val = v[ll];
40         return;
41     }
42
43     int mid = (ll+rr)/2;
44     now.lc = node_cnt++;
45     now.rc = node_cnt++;
46     build(v, now.lc, ll, mid);
47     build(v, now.rc, mid, rr);
48     pull(now, arr[now.lc], arr[now.rc]);
49     return;
50 }
51
52 void update(int pos, int val, int idx, int ll = 0, int rr =
53     n){
54     auto &now = arr[idx];
55
56     if (rr-ll==1){
57         now.val = val;
58         return;
59     }
60
61     int mid = (ll+rr)/2;
62     if (pos<mid){
63         arr[node_cnt] = arr[now.lc];
64         now.lc = node_cnt;
65         node_cnt++;
66         update(pos, val, now.lc, ll, mid);
67     }else{
68         arr[node_cnt] = arr[now.rc];
69         now.rc = node_cnt;
70         node_cnt++;
71         update(pos, val, now.rc, mid, rr);
72     }
73     pull(now, arr[now.lc], arr[now.rc]);
74     return;
75 }
76
77 void update_version(int pos, int val, int ver){
78     update(pos, val, version[ver]);
79 }
80
81 Node query(int ql, int qr, int idx, int ll = 0, int rr =
82     n){
83     auto &now = arr[idx];

```

2.8 Trie

```

1 struct Trie{
2     struct Data{
3         int nxt[2]={0, 0};
4     };
5
6     int sz=0;
7     vector<Data> arr;
8
9     void init(int n){
10         arr.resize(n);
11     }
12
13     void insert(int n){
14         int now=0;
15         for (int i=N ; i>=0 ; i--){
16             int v=(n>>i)&1;
17             if (!arr[now].nxt[v]){
18                 arr[now].nxt[v]=++sz;
19             }
20             now=arr[now].nxt[v];
21         }
22     }
23
24     int query(int n){
25         int now=0, ret=0;
26         for (int i=N ; i>=0 ; i--){
27             int v=(n>>i)&1;
28             if (arr[now].nxt[1-v]){
29                 ret+=(1<<i);
30                 now=arr[now].nxt[1-v];
31             }else if (arr[now].nxt[v]){
32                 now=arr[now].nxt[v];
33             }else{
34                 return ret;
35             }
36         }
37         return ret;
38     }

```

```

81
82     if (ql<=ll && rr<=qr) return now;
83     if (rr<=ql || qr<=ll) return Node();
84
85     int mid = (ll+rr)/2;
86
87     Node ret;
88     pull(ret, query(ql, qr, now.lc, ll, mid), query(ql,
89         qr, now.rc, mid, rr));
90     return ret;
91 }
92
93 Node query_version(int ql, int qr, int ver){
94     return query(ql, qr, version[ver]);
95 }
96
97 void clone_version(int ver){
98     version.push_back(node_cnt);
99     arr[node_cnt] = arr[version[ver]];
100     node_cnt++;
101 }

```

2.9 LC-Segment-Tree

```

1 /*
2 全部都是 0-based
3
4 宣告
5 LC_Segment_Tree st(n);
6
7 函式：
8 update(val)：將一個 pair <a, b> 代表插入一條 y=ax+b 的直線
9 query(x)：查詢所有直線在位置 x 的最小值
10 */
11 const int MAX_V = 1e6+10; // 值域最大值
12
13 struct LC_Segment_Tree{
14     struct Node{ // y = ax+b
15         int a = 0;
16         int b = INF;
17
18         int y(int x){
19             return a*x+b;
20         }
21     };
22     vector<Node> arr;
23
24     LC_Segment_Tree(int n = 0){
25         arr.resize(4*n);
26     }
27
28     void update(Node val, int idx = 0, int ll = 0, int rr =
29         MAX_V){
30         if (rr-ll==1){
31             if (val.y(ll)<arr[idx].y(ll)){
32                 arr[idx] = val;
33             }
34             return;
35         }
36
37         int mid = (ll+rr)/2;
38         if (arr[idx].a > val.a) swap(arr[idx], val); // 原本
39             的斜率要比較小
40         if (arr[idx].y(mid) < val.y(mid)){ // 交點在左邊
41             update(val, idx*2+1, ll, mid);
42         }else{ // 交點在右邊
43             swap(arr[idx], val); // 在左子樹中，新線比舊線還
44                 要好
45             update(val, idx*2+2, mid, rr);
46         }
47         return;
48     }
49
50     int query(int x, int idx = 0, int ll = 0, int rr = MAX_V)
51     {
52         if (rr-ll==1){
53             return arr[idx].y(ll);
54         }
55
56         int mid = (ll+rr)/2;

```

```

53     if (x<mid){
54         return min(arr[idx].y(x), query(x, idx*2+1, ll,
55             mid));
56     }else{
57         return min(arr[idx].y(x), query(x, idx*2+2, mid,
58             rr));
59     }
};

```

2.10 Persistent-Disjoint-Set

```

1 struct Persistent_Disjoint_Set{
2     Persistent_Segment_Tree arr, sz;
3
4     void init(int n){
5         arr.init(n);
6         vector<int> v1;
7         for (int i=0 ; i<n ; i++){
8             v1.push_back(i);
9         }
10        arr.build(v1, 0);
11
12        sz.init(n);
13        vector<int> v2;
14        for (int i=0 ; i<n ; i++){
15            v2.push_back(1);
16        }
17        sz.build(v2, 0);
18    }
19
20    int find(int a){
21        int res = arr.query_version(a, a+1, arr.version.size()
22            (-1).val;
23        if (res==a) return a;
24        return find(res);
25    }
26
27    bool unite(int a, int b){
28        a = find(a);
29        b = find(b);
30
31        if (a!=b){
32
33            int sz1 = sz.query_version(a, a+1, arr.version.
34                size()-1).val;
35            int sz2 = sz.query_version(b, b+1, arr.version.
36                size()-1).val;
37
38            if (sz1<sz2){
39                arr.update_version(a, b, arr.version.size()
40                    -1);
41                sz.update_version(b, sz1+sz2, arr.version.
42                    size()-1);
43            }else{
44                arr.update_version(b, a, arr.version.size()
45                    -1);
46                sz.update_version(a, sz1+sz2, arr.version.
47                    size()-1);
48            }
49            return true;
50        }
51        return false;
52    }
};

```

3 Dynamic-Programming

3.1 SOS-DP

```

1 // 總時間複雜度為  $O(n \cdot 2^n)$ 
2 // 計算  $dp[i] = i$  所有 bit mask 子集的和
3 for (int i=0 ; i<n ; i++){
4     for (int mask=0 ; mask<(1<<n) ; mask++){
5         if ((mask>>i)&1){
6             dp[mask] += dp[mask^(1<<i)];
7         }
8     }
9 }

```

3.2 Digit-DP

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 long long l, r;
5 long long dp[20][10][2][2]; // dp[pos][pre][limit] = 後 pos
6 // 位 · pos 前一位是 pre · (是/否) 有上界 · (是/否) 有前綴零
7 // 的答案數量
8
9 long long memorize_search(string &s, int pos, int pre, bool
10 limit, bool lead){
11
12     // 已經被找過了 · 直接回傳值
13     if (dp[pos][pre][limit][lead]!=-1) return dp[pos][pre][
14         limit][lead];
15
16     // 已經搜尋完畢 · 紀錄答案並回傳
17     if (pos==(int)s.size()){
18         return dp[pos][pre][limit][lead] = 1;
19     }
20
21     // 枚舉目前的位數數字是多少
22     long long ans = 0;
23     for (int now=0 ; now<=(limit ? s[pos]-'0' : 9) ; now++){
24         if (now==pre){
25
26             // 1~9 絕對不能連續出現
27             if (pre!=0) continue;
28
29             // 如果已經不在前綴零的範圍內 · 0 不能連續出現
30             if (lead==false) continue;
31
32             ans += memorize_search(s, pos+1, now, limit&(now==(s[
33                 pos]-'0')), lead&(now==0));
34         }
35     }
36     // 已經搜尋完畢 · 紀錄答案並回傳
37 }

```

```

33     return dp[pos][pre][limit][lead] = ans;
34 }
35
36 // 回傳 [0, n] 有多少數字符合條件
37 long long find_answer(long long n){
38     memset(dp, -1, sizeof(dp));
39     string tmp = to_string(n);
40
41     return memorize_search(tmp, 0, 0, true, true);
42 }
43
44 int main(){
45
46     // input
47     cin >> l >> r;
48
49     // output - 計算 [l, r] 有多少數字任意兩個位數都不相同
50     cout << find_answer(r)-find_answer(l-1) << "\n";
51
52     return 0;
53 }

```

3.3 整數拆分

$dp[i][x]$ = 要將整數 x 拆成 i 堆的「組合數」

$dp[i+1][x+1] += dp[i][x]$ (創造新的一堆)
 $dp[i][x+i] += dp[i][x]$ (把每一堆都增加1)

4 Geometry

4.1 Point-Struct

```

1 // 記得確定 point 是要整數點還是浮點數
2 const double EPS = 1e-6;
3
4 struct point{
5     double x, y;
6
7     // 純量乘、除法
8     point operator * (int a){return {a*x, a*y};};
9     point operator / (int a){return {a/x, a/y};};
10
11     // 向量加、減法
12     point operator + (point a){return {x+a.x, y+a.y};};
13     point operator - (point a){return {x-a.x, y-a.y};};
14
15     // 內積、外積
16     double operator * (point a){return x*a.x+y*a.y};};
17     double operator ^ (point a){return x*a.y-y*a.x};};
18
19     // 極角排序 (順時鐘)
20     bool operator < (const point &a) const {return (x*a.y<a.x
21         *y);};
22
23     // 長度
24     double len(){return sqrt(x*x+y*y);};
25 }

```

```

25 // 判斷向量正負：1=正數，0=0，-1=負數
26 int sign(double a){
27     if (abs(a)<EPS) return 0;
28     else return (a>0 ? 1 : -1);
29 }
30
31 // 判斷 ab 到 ac 的方向：1=逆時鐘，0=重疊，-1=順時鐘
32 int ori(point a, point b, point c){
33     return sign((b-a)^(c-a));
34 }
35

```

4.2 Line-Intersection

```

1 bool same_seg(point a, point b, point c){
2     return sign((b-a)^(c-a))==0;
3 }
4
5 // c 是否在 ab 裡面
6 bool banana(point a, point b, point c){
7     if (!same_seg(a, b, c)) return 0;
8     return sign((a-c)*(b-c))<=0;
9 }
10
11 // 判斷 ab 是否跟 cd 相交
12 bool seg_cross(point a, point b, point c, point d){
13     int s1=ori(a, b, c);
14     int s2=ori(a, b, d);
15     int s3=ori(c, d, a);
16     int s4=ori(c, d, b);
17     if (banana(a, b, c) || banana(a, b, d) || banana(c, d, a)
18         || banana(c, d, b)) return 1;
19     return (s1*s2<0) && (s3*s4<0);
20 }

```

4.3 Pick's-Theorem

給定頂點坐標均是整點的簡單多邊形，面積 = 內部格點數 + 邊上格點數/2 - 1

4.4 Convex-Hull

```

1 vector<point> convex_hull(vector<point> points){
2
3     sort(points.begin(), points.end());
4     vector<point> hull;
5
6     for (int _=0 ; _<2 ; _++){
7         int sz=hull.size();
8
9         for (int i=0 ; i<points.size() ; i++){
10             while (hull.size()>sz+2 && ori(hull[hull.size()-2], hull[hull.size()-1], points[i])<0){
11                 hull.pop_back();
12             }
13             hull.push_back(points[i]);
14         }
15     }

```

```

16         hull.pop_back();
17         reverse(points.begin(), points.end());
18     }
19     return hull;
20 }
21

```

5 Graph

5.1 Find-Bridge

```

1 vector<int> dep(MAX_N), low(MAX_N);
2 vector<pair<int, int>> bridge;
3 bitset<MAX_N> vis;
4
5 void dfs(int now, int pre){
6     vis[now] = 1;
7     low[now] = dep[now] = (now==1 ? 0 : dep[pre]+1);
8
9     for (auto x : G[now]){
10         if (x==pre){
11             continue;
12         }else if (vis[x]==0){
13             // 沒有走過的節點
14             dfs(x, now);
15             low[now] = min(low[now], low[x]);
16         }else if (vis[x]==1){
17             low[now] = min(low[now], dep[x]);
18         }
19     }
20
21     if (now!=1 && low[now]==dep[now]){
22         bridge.push_back({now, pre});
23     }
24     return;
25 }

```

5.2 Find-AP

```

1 vector<int> dep(MAX_N), low(MAX_N), AP;
2 bitset<MAX_N> vis;
3
4 void dfs(int now, int pre){
5     int cnt = 0;
6     bool ap = 0;
7     vis[now] = 1;
8     low[now] = dep[now] = (now==1 ? 0 : dep[pre]+1);
9
10    for (auto x : G[now]){
11        if (x==pre){
12            continue;
13        }else if (vis[x]==0){
14            cnt++;
15            dfs(x, now);
16            low[now] = min(low[now], low[x]);
17            if (low[x]>=dep[now]) ap=1;
18        }else{
19            low[now] = min(low[now], dep[x]);

```

```

20    }
21 }
22
23 if ((now==pre && cnt>=2) || (now!=pre && ap)){
24     AP.push_back(now);
25 }
26 }

```

5.3 SCC 與縮點

```

1 /*
2 給定一個有向圖，迴傳傳縮點後的圖、SCC 的資訊
3 所有點都以 based-0 編號
4
5 函式：
6 SCC_compress G(n): 宣告一個有 n 個點的圖
7 .add_edge(u, v): 加上一條邊 u -> v
8 .compress: O(n Log n) 計算 G3、SCC、SCC_id 的資訊
9 */
10 struct SCC_compress{
11     int n = 0, m = 0;
12     vector<vector<int>> G, inv_G, result;
13     vector<pair<int, int>> edges;
14     vector<bool> vis;
15     vector<int> order;
16
17     vector<vector<int>> SCC; // SCC[i] = 某個 SCC 中的所有點
18     vector<int> SCC_id; // SCC_id[i] = 第 i 個點在第幾個 SCC
19
20     SCC_compress(int _n){ // 點的數量
21         n = _n;
22         G.resize(n);
23         inv_G.resize(n);
24         result.resize(n);
25         vis.resize(n);
26         SCC_id.resize(n);
27     }
28
29     void add_edge(int u, int v){
30         G[u].push_back(v);
31         inv_G[v].push_back(u);
32         edges.push_back({u, v});
33         m++;
34     }
35
36     void dfs1(vector<vector<int>> &G, int now){
37         vis[now] = 1;
38         for (auto x : G[now]){
39             if (vis[x]==0){
40                 dfs1(G, x);
41             }
42         }
43         order.push_back(now);
44         return;
45     }
46
47     void dfs2(vector<vector<int>> &G, int now){
48         SCC_id[now] = SCC.size()-1;
49         SCC.back().push_back(now);
50         vis[now] = 1;

```



```

51     for (auto x : G[now]){
52         if (vis[x]==0){
53             dfs2(G, x);
54         }
55     }
56     return;
57 }
58
59 void compress(){
60     // 找反圖順序
61     fill(vis.begin(), vis.end(), 0);
62     for (int i=0; i<n; i++){
63         if (vis[i]==0){
64             dfs1(G, i);
65         }
66     }
67
68     // 找到 SCC
69     fill(vis.begin(), vis.end(), 0);
70     reverse(order.begin(), order.end());
71     for (int i=0; i<n; i++){
72         if (vis[order[i]]==0){
73             SCC.push_back(vector<int>());
74             dfs2(inv_G, order[i]);
75         }
76     }
77
78     // 縮點做 DAG
79     for (int i=0; i<m; i++){
80         if (SCC_id[edges[i].first]!=SCC_id[edges[i].second]){
81             result[SCC_id[edges[i].first]].push_back(SCC_id[edges[i].second]);
82         }
83     }
84     for (int i=0; i<SCC.size(); i++){
85         sort(result[i].begin(), result[i].end());
86         result[i].resize(unique(result[i].begin(), result[i].end())-result[i].begin());
87     }
88 }
89 };
90

```

5.4 MCMF

```

1 // frostray 會用，所以他超強
2 struct Flow {
3     struct Edge {
4         int u, rc, k, rv;
5     };
6     vector<vector<Edge>> G;
7     vector<int> par, par_eid;
8     Flow(int n) : G(n+1), par(n+1), par_eid(n+1) {}
9
10    // v->u | 流量 : c, cost: k
11    void add(int v, int u, int c, int k) {
12        G[v].push_back({u, c, k, (int)G[v].size()});
13        G[u].push_back({v, 0, -k, (int)G[v].size() - 1});
14    }
15    int spfa(int s, int t) {
16        fill(par.begin(), par.end(), -1);

```

```

17        vector<int> dis(par.size(), LONG_LONG_MAX);
18        vector<bool> in_q(par.size(), false);
19        queue<int> Q;
20        dis[s] = 0; in_q[s] = true;
21        Q.push(s);
22        while (!Q.empty()) {
23            int v = Q.front(); Q.pop();
24            in_q[v] = false;
25            for (int i = 0; i < (int)G[v].size(); i++) {
26                auto [u, rc, k, rv] = G[v][i];
27                if (rc > 0 && dis[v] + k < dis[u]) {
28                    dis[u] = dis[v] + k;
29                    par[u] = v;
30                    par_eid[u] = i;
31                    if (!in_q[u]) Q.push(u);
32                    in_q[u] = true;
33                }
34            }
35            return dis[t];
36        }
37    }
38
39    // <最大流 · 最小費用>
40    pair<int, int> flow(int s, int t) {
41        int fl = 0, cost = 0, d;
42        while ((d = spfa(s, t)) < LONG_LONG_MAX) {
43            int cur = LONG_LONG_MAX;
44            for (int v = t; v != s; v = par[v])
45                cur = min(cur, G[par[v]][par_eid[v]].rc);
46            fl += cur;
47            cost += d * cur;
48            for (int v = t; v != s; v = par[v]) {
49                G[par[v]][par_eid[v]].rc -= cur;
50                G[v][G[par[v]][par_eid[v]].rv].rc += cur;
51            }
52        }
53        return {fl, cost};
54    }
55 };

```

5.5 HLD

```

1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4
5 const int N = 100005;
6 vector<int> G[N];
7 struct HLD {
8     vector<int> pa, sz, depth, mxson, topf, id;
9     int n, idcnt = 0;
10    HLD(int _n) : n(_n), pa(_n+1), sz(_n+1), depth(_n+1), mxson(_n+1), topf(_n+1), id(_n+1) {}
11    void dfs1(int v = 1, int p = -1) {
12        pa[v] = p; sz[v] = 1; mxson[v] = 0;
13        depth[v] = (p == -1 ? 0 : depth[p] + 1);
14        for (int u : G[v]) {
15            if (u == p) continue;
16            dfs1(u, v);
17            sz[v] += sz[u];
18            if (sz[u] > sz[mxson[v]]) mxson[v] = u;
19        }
20    }

```

```

21    void dfs2(int v = 1, int top = 1) {
22        id[v] = ++idcnt;
23        topf[v] = top;
24        if (mxson[v]) dfs2(mxson[v], top);
25        for (int u : G[v]) {
26            if (u == mxson[v] || u == pa[v]) continue;
27            dfs2(u, u);
28        }
29    }
30    // query 為區間資料結構
31    int path_query(int a, int b) {
32        int res = 0;
33        while (topf[a] != topf[b]) { // 若不在同一條鍊上
34            if (depth[topf[a]] < depth[topf[b]]) swap(a, b);
35            res = max(res, 0ll); // query : l = id[topf[a]], r = id[a]
36            a = pa[topf[a]];
37        }
38        // 此時已在同一條鍊上
39        if (depth[a] < depth[b]) swap(a, b);
40        res = max(res, 0ll); // query : l = id[b], r = id[a]
41        return res;
42    }
43 };

```

5.6 Tree-Isomorphism

```

1 #include <bits/stdc++.h>
2 #pragma GCC optimize("O3,unroll-loops")
3 #define fastio ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
4 #define dbg(x) cerr << #x << " = " << x << endl
5 #define int long long
6 using namespace std;
7
8 // declare
9 const int MAX_SIZE = 2e5+5;
10 const int INF = 9e18;
11 const int MOD = 1e9+7;
12 const double EPS = 1e-6;
13 typedef vector<vector<int>> Graph;
14 typedef map<vector<int>, int> Hash;
15
16 int n, a, b;
17 int id1, id2;
18 pair<int, int> c1, c2;
19 vector<int> sz1(MAX_SIZE), sz2(MAX_SIZE);
20 vector<int> we1(MAX_SIZE), we2(MAX_SIZE);
21 Graph g1(MAX_SIZE), g2(MAX_SIZE);
22 Hash m1, m2;
23 int testcase=0;
24
25 void centroid(Graph &g, vector<int> &s, vector<int> &w, pair<int, int> &rec, int now, int pre){
26     s[now]=1;
27     w[now]=0;
28     for (auto x : g[now]){
29         if (x!=pre){
30             centroid(g, s, w, rec, x, now);
31             s[now]+=s[x];
32             w[now]=max(w[now], s[x]);
33         }

```

```

34 }
35
36 w[now]=max(w[now], n-s[now]);
37 if (w[now]<=n/2){
38     if (rec.first==0) rec.first=now;
39     else rec.second=now;
40 }
41 }
42
43 int dfs(Graph &g, Hash &m, int &id, int now, int pre){
44     vector<int> v;
45     for (auto x : g[now]){
46         if (x!=pre){
47             int add=dfs(g, m, id, x, now);
48             v.push_back(add);
49         }
50     }
51     sort(v.begin(), v.end());
52
53     if (m.find(v)!=m.end()){
54         return m[v];
55     }else{
56         m[v]=++id;
57         return id;
58     }
59 }
60
61
62 void solve1(){
63
64     // init
65     id1=0;
66     id2=0;
67     c1={0, 0};
68     c2={0, 0};
69     fill(sz1.begin(), sz1.begin()+n+1, 0);
70     fill(sz2.begin(), sz2.begin()+n+1, 0);
71     fill(we1.begin(), we1.begin()+n+1, 0);
72     fill(we2.begin(), we2.begin()+n+1, 0);
73     for (int i=1 ; i<=n ; i++){
74         g1[i].clear();
75         g2[i].clear();
76     }
77     m1.clear();
78     m2.clear();
79
80     // input
81     cin >> n;
82     for (int i=0 ; i<n-1 ; i++){
83         cin >> a >> b;
84         g1[a].push_back(b);
85         g1[b].push_back(a);
86     }
87     for (int i=0 ; i<n-1 ; i++){
88         cin >> a >> b;
89         g2[a].push_back(b);
90         g2[b].push_back(a);
91     }
92
93     // get tree centroid
94     centroid(g1, sz1, we1, c1, 1, 0);
95     centroid(g2, sz2, we2, c2, 1, 0);
96
97     // process
98     int res1=0, res2=0, res3=0;
99     if (c2.second!=0){

```

```

100     res1=dfs(g1, m1, id1, c1.first, 0);
101     m2=m1;
102     id2=id1;
103     res2=dfs(g2, m1, id1, c2.first, 0);
104     res3=dfs(g2, m2, id2, c2.second, 0);
105 }else if (c1.second!=0){
106     res1=dfs(g2, m1, id1, c2.first, 0);
107     m2=m1;
108     id2=id1;
109     res2=dfs(g1, m1, id1, c1.first, 0);
110     res3=dfs(g1, m2, id2, c1.second, 0);
111 }else{
112     res1=dfs(g1, m1, id1, c1.first, 0);
113     res2=dfs(g2, m1, id1, c2.first, 0);
114 }
115
116 // output
117 cout << (res1==res2 || res1==res3 ? "YES" : "NO") << endl
118 ;
119
120 return;
121 }
122
123 signed main(void){
124     fastio;
125
126     int t=1;
127     cin >> t;
128     while (t--){
129         solve1();
130     }
131     return 0;

```

5.7 Bridge BCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 200005;
5 vector<int> G[N];
6 int low[N], depth[N];
7 bool vis[N];
8 vector<vector<int>> bcc;
9 stack<int> stk;
10
11 void dfs(int v, int p) {
12     stk.push(v);
13     vis[v] = true;
14     low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
15     for (int u : G[v]) {
16         if (u == p) continue;
17         if (!vis[u]) {
18             // (v, u) 是樹邊
19             dfs(u, v);
20             low[v] = min(low[v], low[u]);
21         } else {
22             // (v, u) 是回邊
23             low[v] = min(low[v], depth[u]);
24         }
25     }
26     // v 在不依靠父邊的情況下永遠沒辦法走到它的祖先

```

```

27     if (low[v] == depth[v]) {
28         bcc.emplace_back();
29         while (stk.top() != v) {
30             bcc.back().push_back(stk.top());
31             stk.pop();
32         }
33         bcc.back().push_back(stk.top());
34         stk.pop();
35     }
36 }
37
38 #include <bits/stdc++.h>
39 using namespace std;
40
41 const int N = 200005;
42 vector<int> G[N];
43 int low[N], depth[N];
44 bool vis[N];
45 vector<vector<int>> bcc;
46 stack<int> stk;
47
48 void dfs(int v, int p) {
49     stk.push(v);
50     vis[v] = true;
51     low[v] = depth[v] = (p == -1 ? 1 : depth[p] + 1);
52     for (int u : G[v]) {
53         if (u == p) continue;
54         if (!vis[u]) {
55             // (v, u) 是樹邊
56             dfs(u, v);
57             low[v] = min(low[v], low[u]);
58             // u 無法在不經過父邊的情況下走到 v 的祖先
59             if (low[u] >= depth[v]) {
60                 bcc.emplace_back();
61                 while (stk.top() != u) {
62                     bcc.back().push_back(stk.top());
63                     stk.pop();
64                 }
65                 bcc.back().push_back(stk.top());
66                 stk.pop();
67                 bcc.back().push_back(v);
68             } else {
69                 // (v, u) 是回邊
70                 low[v] = min(low[v], depth[u]);
71             }
72         }
73     }
74 }

```

5.8 Cut BCC

5.9 圓方樹

```

1 #include <bits/stdc++.h>
2 #define lp(i,a,b) for(int i=(a);i<(b);i++)
3 #define pii pair<int,int>
4 #define pb push_back
5 #define ins insert
6 #define ff first

```



```

7 #define ss second
8 #define opa(x) cerr << #x << " = " << x << ", ";
9 #define op(x) cerr << #x << " = " << x << endl;
10 #define ops(x) cerr << x;
11 #define etr cerr << endl;
12 #define spc cerr << ' ';
13 #define BAE(x) (x).begin(), (x).end()
14 #define STL(x) cerr << #x << " : "; for(auto &qwe:x) cerr <<
    qwe << ' '; cerr << endl;
15 #define deb1 cerr << "deb1" << endl;
16 #define deb2 cerr << "deb2" << endl;
17 #define deb3 cerr << "deb3" << endl;
18 #define deb4 cerr << "deb4" << endl;
19 #define deb5 cerr << "deb5" << endl;
20 #define bye exit(0);
21 using namespace std;
22
23 const int mxn = (int)(2e5) + 10;
24 const int mxlg = 17;
25 int last_special_node = (int)(1e5) + 1;
26 vector<int> E[mxn], F[mxn];
27
28 struct edg{
29     int fr, to;
30     edg(int _fr, int _to){
31         fr = _fr;
32         to = _to;
33     }
34 };
35 ostream& operator<<(ostream& os, edg x){os << x.fr << "--" <<
    x.to;}
36 vector<edg> EV;
37
38 void tarjan(int v, int par, stack<int>& S){
39     static vector<int> dfn(mxn), low(mxn);
40     static vector<bool> to_add(mxn);
41     static int nowT = 0;
42
43     int childs = 0;
44     nowT += 1;
45     dfn[v] = low[v] = nowT;
46     for(auto &ne:E[v]){
47         int i = EV[ne].to;
48         if(i == par) continue;
49         if(!dfn[i]){
50             S.push(ne);
51             tarjan(i, v, S);
52             childs += 1;
53             low[v] = min(low[v], low[i]);
54
55             if(par >= 0 && low[i] >= dfn[v]){
56                 vector<int> bcc;
57                 int tmp;
58                 do{
59                     tmp = S.top(); S.pop();
60                     if(!to_add[EV[tmp].fr]){
61                         to_add[EV[tmp].fr] = true;
62                         bcc.pb(EV[tmp].fr);
63                     }
64                     if(!to_add[EV[tmp].to]){
65                         to_add[EV[tmp].to] = true;
66                         bcc.pb(EV[tmp].to);
67                     }
68                 }while(tmp != ne);
69                 for(auto &j:bcc){
70                     to_add[j] = false;
71
72                     F[last_special_node].pb(j);
73                     F[j].pb(last_special_node);
74                 }
75                 last_special_node += 1;
76             }
77         }
78         else{
79             low[v] = min(low[v], dfn[i]);
80             if(dfn[i] < dfn[v]){ // edge i--v will be visited
81                 // twice at here, but we only need one.
82                 S.push(ne);
83             }
84         }
85     }
86
87     int dep[mxn], jmp[mxn][mxlg];
88     void dfs_lca(int v, int par, int depth){
89         dep[v] = depth;
90         for(auto &i:F[v]){
91             if(i == par) continue;
92             jmp[i][0] = v;
93             dfs_lca(i, v, depth + 1);
94         }
95     }
96
97     inline void build_lca(){
98         jmp[1][0] = 1;
99         dfs_lca(1, -1, 1);
100         lp(j, 1, mxlg){
101             lp(i, 1, mxn){
102                 jmp[i][j] = jmp[jmp[i][j-1]][j-1];
103             }
104         }
105     }
106
107     inline int lca(int x, int y){
108         if(dep[x] < dep[y]){ swap(x, y); }
109
110         int diff = dep[x] - dep[y];
111         lp(j, 0, mxlg){
112             if((diff >> j) & 1){
113                 x = jmp[x][j];
114             }
115         }
116         if(x == y) return x;
117
118         for(int j = mxlg - 1; j >= 0; j--){
119             if(jmp[x][j] != jmp[y][j]){
120                 x = jmp[x][j];
121                 y = jmp[y][j];
122             }
123         }
124         return jmp[x][0];
125     }
126
127     inline bool can_reach(int fr, int to){
128         if(dep[to] > dep[fr]) return false;
129
130         int diff = dep[fr] - dep[to];
131         lp(j, 0, mxlg){
132             if((diff >> j) & 1){
133                 fr = jmp[fr][j];
134             }
135         }
136         return fr == to;
137     }
138 }
139
140 int main(){
141     ios::sync_with_stdio(false); cin.tie(0);
142     // freopen("test_input.txt", "r", stdin);
143     int n, m, q; cin >> n >> m >> q;
144     lp(i, 0, m){
145         int u, v; cin >> u >> v;
146         E[u].pb(EV.size());
147         EV.pb(edg(u, v));
148         E[v].pb(EV.size());
149         EV.pb(edg(v, u));
150     }
151     E[0].pb(EV.size());
152     EV.pb(edg(0, 1));
153     stack<int> S;
154     tarjan(0, -1, S);
155     build_lca();
156
157     lp(queries, 0, q){
158         int fr, to, relay; cin >> fr >> to >> relay;
159         if(fr == relay || to == relay){
160             cout << "NO\n";
161             continue;
162         }
163         if((can_reach(fr, relay) || can_reach(to, relay)) &&
164             dep[relay] >= dep[lca(fr, to)]){
165             cout << "NO\n";
166             continue;
167         }
168         cout << "YES\n";
169     }
170 }

```

5.10 Dijkstra

```

1 // 可以在  $O(E \log E)$  的時間複雜度解決在無負權有向圖單點源最短
2 路
3
4 const int INF = 2e18; // 要確保 INF 開的足夠大
5
6 vector<vector<pair<int, int>>> G(n); // G[i] = <節點, 權重>
7 vector<int> dis(n, INF);
8 priority_queue<pair<int, int>, vector<pair<int, int>>,
9     greater<pair<int, int>>> pq;
10 dis[s] = 0;
11 pq.push({0, s});
12
13 while (pq.size()){
14     int now_dis = pq.top().first;
15     int now_node = pq.top().second;
16     pq.pop();
17
18     if (now_dis > dis[now_node]) continue;
19
20     for (auto x : G[now_node]){
21         if (now_dis + x.second < dis[x.first]){
22             dis[x.first] = now_dis + x.second;
23             pq.push({x.first, dis[x.first]});
24         }
25     }
26 }

```

6 Math

6.1 Burnside's-Lemma

$$\sum_{k=1}^n \frac{c(k)}{n}$$

- n : 有多少種置換方式 (例如 : 旋轉方式)
- $c(k)$: 所有可能中 , 經過 k 次旋轉後 , 仍不會和別人相同的方式的數量

6.2 線性篩

```
1 const int MAX_N = 5e5;
2
3 // lpf[i] = i 的最小質因數
4 vector<int> prime, lpf(MAX_N);
5
6 void prime_init(){
7     for (int i=2 ; i<MAX_N ; i++){
8         if (lpf[i]==0){
9             lpf[i]=i;
10            prime.push_back(i);
11        }
12
13        for (int j : prime){
14            if (i*j>=MAX_N) break;
15            lpf[i*j]=j;
16            if (lpf[i]==j) break;
17        }
18    }
19 }
```

6.3 Lucas's-Theorem

```
1 // 對於很大的  $C^n_m$  對質數  $p$  取模 , 只要  $p$  不大就可以用。
2 int Lucas(int n, int m, int p){
3     if (m==0) return 1;
4     return (C(n%p, m%p, p)*Lucas(n/p, m/p, p)%p);
5 }
```

6.4 Miller-Rabin

```
1 // O(1)
2 typedef Uint unsigned long long
3 Uint modmul(Uint a, Uint b, Uint m) {
4     int ret = a*b - m*(Uint)(1.L/m*a*b);
5     return ret + m*(ret < 0) - m*(ret>=(int)m);
6 }
7
8 int qp(int b, int p, int m){
9     int ret = 1;
10    for ( ; p ; p>>=1){
11        if (p&1){
```

```
12        ret = modmul(ret, b, m);
13    }
14    b = modmul(b, b, m);
15 }
16 return ret;
17 }
18
19 vector<int> llsprp = {2, 325, 9375, 28178, 450775, 9780504,
20 1795265022};
21 bool isprime(int n, vector<int> sprp = llsprp){
22     if (n==2) return 1;
23     if (n<2 || n%2==0) return 0;
24
25     int t = 0;
26     int u = n-1;
27     for ( ; u%2==0 ; t++) u>>=1;
28
29     for (int i=0 ; i<sprp.size() ; i++){
30         int a = sprp[i]%n;
31         if (a==0 || a==1 || a==n-1) continue;
32         int x = qp(a, u, n);
33         if (x==1 || x==n-1) continue;
34         for (int j=0 ; j<t ; j++){
35             x = modmul(x, x, n);
36             if (x==1) return 0;
37             if (x==n-1) break;
38         }
39         if (x==n-1) continue;
40         return 0;
41     }
42     return 1;
43 }
44 }
```

6.5 Matrix

```
1 struct Matrix{
2     int n, m;
3     vector<vector<int>> arr;
4
5     Matrix(int _n, int _m){
6         n = _n;
7         m = _m;
8         arr.resize(n, vector<int>(m));
9     }
10
11     Matrix operator * (const Matrix B){
12         Matrix ret(n, B.m);
13
14         for (int i=0 ; i<n ; i++){
15             for (int j=0 ; j<B.m ; j++){
16                 for (int k=0 ; k<m ; k++){
17                     ret.arr[i][j] += arr[i][k]*B.arr[k][j];
18                     ret.arr[i][j] %= MOD;
19                 }
20             }
21         }
22         return ret;
23     }
24 }
25 ;;
```

6.6 Pollard-Rho

```
1 mt19937 seed(chrono::steady_clock::now().time_since_epoch().
2 count());
3 int rnd(int l, int r){
4     return uniform_int_distribution<int>(l, r)(seed);
5 }
6 // O(n^{1/4}) 回傳 1 或自己的因數 , 記得先判斷 n 是不是質數
7 (用 Miller-Rabin)
8 int Pollard_Rho(int n){
9     int s = 0, t = 0;
10    int c = rnd(1, n-1);
11
12    int step = 0, goal = 1;
13    int val = 1;
14
15    for (goal=1 ; ; goal<=1, s=t, val=1){
16        for (step=1 ; step<=goal ; step++){
17            t = ((__int128)t*t+c)%n;
18            val = ((__int128)val*abs(t-s)%n;
19
20            if ((step % 127) == 0){
21                int d = __gcd(val, n);
22                if (d>1) return d;
23            }
24        }
25
26        int d = __gcd(val, n);
27        if (d>1) return d;
28    }
29 }
```

6.7 最大質因數

```
1 void max_fac(int n, int &ret){
2     if (n<=ret || n<2) return;
3     if (isprime(n)){
4         ret = max(ret, n);
5         return;
6     }
7
8     int p = Pollard_Rho(n);
9     max_fac(p, ret), max_fac(n/p, ret);
10 }
```

6.8 中國剩餘定理 (m 互質)

```
1 vector<int> a, m;
2
3 int extgcd(int a, int b, int &x, int &y){
4     if (b==0){
5         x=1, y=0;
6         return a;
7     }
8
9     int ret=extgcd(b, a%b, y, x);
10    y-=a/b*x;
11    return ret;
```

```

12 }
13
14 // n = 有幾個式子 · 求解  $x \equiv a_i \pmod{m_i}$ 
15 int CRT(int n, vector<int> &a, vector<int> &m){
16     int p=1, ans=0;
17
18     vector<int> M(n), inv_M(n);
19
20     for (int i=0 ; i<n ; i++){
21         for (int i=0 ; i<n ; i++){
22             M[i]=p/m[i];
23             extgcd(M[i], m[i], inv_M[i], tmp);
24             ans+=a[i]*inv_M[i]*M[i];
25             ans%=p;
26         }
27
28         return (ans%p+p)%p;
29 }

```

6.9 中國剩餘定理 (m 不互質)

```

1 int extgcd(int a, int b, int &x, int &y){
2     if (b==0){
3         x=1, y=0;
4         return a;
5     }
6
7     int ret=extgcd(b, a%b, y, x);
8     y-=a/b*x;
9     return ret;
10 }
11
12 // 對於方程組的式子兩兩求解
13 // {是否有解, {a, m}}
14 pair<bool, pair<int, int>> CRT(int a1, int m1, int a2, int m2
15 ){
16     int g=__gcd(m1, m2);
17     if ((a2-a1)%g!=0) return {0, {-1, -1}};
18
19     int x, y;
20     extgcd(m1, m2, x, y);
21
22     x=(a2-a1)*x/g; // 兩者不能相反
23     a1=x*m1+a1;
24     m1=m1*m2/g;
25     a1=(a1%m1+m1)%m1;
26     return {1, {a1, m1}};
27 }

```

6.10 歐拉公式

```

1 // phi(n) = 小於 n 並與 n 互質的正整數數量。
2 //  $O(\sqrt{n})$  · 回傳 phi(n)
3 int phi(int n){
4     int ret = n;
5
6     for (int i=2 ; i*i<=n ; i++){
7         if (n%i==0){
8             while (n%i==0) n /= i;

```

```

9             ret = ret*(i-1)/i;
10         }
11     }
12     if (n>1) ret = ret*(n-1)/n;
13     return ret;
14 }
15
16 //  $O(n \log n)$  · 回傳 1~n 的 phi 值
17 vector<int> phi_1_to_n(int n){
18     vector<int> phi(n+1);
19     phi[0]=0;
20     phi[1]=1;
21
22     for (int i=2 ; i<=n ; i++){
23         phi[i]=i-1;
24     }
25
26     for (int i=2 ; i<=n ; i++){
27         for (int j=2*i ; j<=n ; j+=i){ // 枚舉所有倍數
28             phi[j]-=phi[i];
29         }
30     }
31
32     return phi;
33 }
34 }

```

6.11 卡特蘭數

任意括號序列： $C_n = \frac{1}{n+1} \binom{2n}{n}$

6.12 歐拉定理

若 a, m 互質 · 則：

$$a^n \bmod m = a^{n \bmod \varphi(m)} \bmod m$$

若 a, m 可能是任何數 · 則：

$$a^{\varphi(m) + [n \bmod \varphi(m)]} \bmod m$$

6.13 Fraction

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /// Fraction template starts ///
5 #define fraction_template_bonus_check
6 const long long ll_overflow_warning_value = (long long)(3e9);
7
8 long long gcd(long long a, long long b){
9     if(a == 0) return 0;
10    if(b == 0) return a;
11    if(a < b) return gcd(b,a);
12    return gcd(b, a%b);
13 }
14 struct frac{

```

```

15     long long a, b;
16     frac(long long _a = 0, long long _b = 1){
17         a = _a; b = _b;
18         if(b == 0){
19             cerr << "Error: division by zero\n";
20             cerr << "Called : Constructor(" << _a << ", " <<
                _b << ")\n";
21             return;
22         }
23         if(a == 0){b = 1; return;}
24         if(b < 0){a = -a; b = -b;}
25         long long gcd_ab = gcd(std::abs(a), b);
26         if(gcd_ab != 1){a /= gcd_ab; b /= gcd_ab;}
27
28         #ifdef fraction_template_bonus_check
29         if(std::abs(a) > ll_overflow_warning_value || b >
            ll_overflow_warning_value){
30             cerr << "Overflow warning : " << a << "/" << b <<
                "\n";
31         }
32         #endif // fraction_template_bonus_check
33     }
34     frac operator+(frac const &B){
35         return frac(a*(B.b)+(B.a)*b, b*(B.b));
36     }
37     frac operator-(frac const &B){
38         return frac(a*(B.b)-(B.a)*b, b*(B.b));
39     }
40     frac operator*(frac const &B){
41         return frac(a*(B.a), b*(B.b));
42     }
43     frac operator/(frac const &B){
44         return frac(a*(B.b), b*(B.a));
45     }
46
47     frac operator+=(frac const &B){
48         *this = frac(a*(B.b)+(B.a)*b, b*(B.b));
49     }
50     frac operator-=(frac const &B){
51         *this = frac(a*(B.b)-(B.a)*b, b*(B.b));
52     }
53     frac operator*=(frac const &B){
54         *this = frac(a*(B.a), b*(B.b));
55     }
56     frac operator/=(frac const &B){
57         *this = frac(a*(B.b), b*(B.a));
58     }
59
60     frac abs(){
61         a = std::abs(a);
62         return *this;
63     }
64
65     bool operator<(frac const &B){
66         return a*B.b < B.a*b;}
67     bool operator<=(frac const &B){
68         return a*B.b <= B.a*b;}
69     bool operator>(frac const &B){
70         return a*B.b > B.a*b;}
71     bool operator>=(frac const &B){
72         return a*B.b >= B.a*b;}
73     bool operator==(frac const &B){
74         return a * B.b == B.a * b;}
75     bool operator!=(frac const &B){
76         return a * B.b != B.a * b;}
77 };
78 ostream& operator<<(ostream &os, const frac& A){
79     os << A.a << "/" << A.b;
80     return os;
81 }
82
83 /// Fraction template ends ///
84
85 void test(frac A, frac B){
86     cout << "A = " << A << endl;
87     cout << "B = " << B << endl;

```

```

78 cout << endl;
79 cout << "A + B = " << A + B << endl;
80 cout << "A - B = " << A - B << endl;
81 cout << "A * B = " << A * B << endl;
82 cout << "A / B = " << A / B << endl;
83 cout << endl;
84 cout << "(A < B) = " << (A < B) << endl;
85 cout << "(A <= B) = " << (A <= B) << endl;
86 cout << "(A > B) = " << (A > B) << endl;
87 cout << "(A >= B) = " << (A >= B) << endl;
88 cout << "(A == B) = " << (A == B) << endl;
89 cout << "(A != B) = " << (A != B) << endl;
90 cout << "-----\n";
91 return;
92 }
93
94 int main(){
95     frac tmp1(-7, 2);
96     frac tmp2(5, 3);
97     test(tmp1, tmp2);
98
99     frac tmp3(-7);
100    frac tmp4(0);
101    test(tmp3, tmp4);
102    return 0;
103 }

```

6.14 錯排公式

錯排公式：(n 個人中，每個人皆不再原來位置的組合數)

$$dp_i = \begin{cases} 1 & i = 0 \\ 0 & i = 1 \\ (i-1)(dp_{i-1} + dp_{i-2}) & \text{otherwise} \end{cases}$$

6.15 Quick-Pow

```

1 int qp(int b, int p, int m = MOD){
2     int ret = 1;
3     for (; p; p>>=1){
4         if (p&1) ret = ret*b%m;
5         b = b*b%m;
6     }
7     return ret;
8 }

```

6.16 二元一次方程式

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases} = \begin{cases} x = \frac{ed-bf}{ad-bc} \\ y = \frac{af-ec}{ad-bc} \end{cases}$$

若 $x = \frac{0}{0}$ 且 $y = \frac{0}{0}$ ，則代表無限多組解。若 $x = \frac{*}{0}$ 且 $y = \frac{*}{0}$ ，則代表無解。

6.17 Josephus

```

1 // 有 n 個人，第偶數個報數的人被刪掉，問第 k 個被踢掉的是誰
2 int solve(int n, int k){
3     if (n==1) return 1;
4     if (k<=(n+1)/2){
5         if (2*k>n) return 2*k%n;
6         else return 2*k;
7     }else{
8         int res=solve(n/2, k-(n+1)/2);
9         if (n&1) return 2*res+1;
10        else return 2*res-1;
11    }
12 }

```

7 Misc

7.1 Xor-Basis

```

1 vector<int> basis;
2 void add_vector(int x){
3     for (auto v : basis){
4         x=min(x, x^v);
5     }
6     if (x) basis.push_back(x);
7 }
8
9 // 給一數字集合 S，求能不能 XOR 出 x
10 bool check(int x){
11     for (auto v : basis){
12         x=min(x, x^v);
13     }
14     return x;
15 }
16
17 // 給一數字集合 S，求能 XOR 出多少數字
18 // 答案等於 2^{basis 的大小}
19
20 // 給一數字集合 S，求 XOR 出最大的數字
21 int get_max(){
22     int ans=0;
23     for (auto v : basis){
24         ans=max(ans, ans^v);
25     }
26     return ans;
27 }

```

7.2 Default-Code

```

1 #include <bits/stdc++.h>
2 #define int long long
3 #define fastio ios::sync_with_stdio(0), cin.tie(0);
4 using namespace std;
5
6 #ifdef LOCAL
7 void debug(){cerr << "\n";}

```

```

8 template<class T, class ... U>
9 void debug(T a, U ... b){cerr << a << " ", debug(b...);}
10 template<class T> void pary(T l, T r){
11     while (l!=r) cerr << *l << " ", l++;
12     cerr << "\n";
13 }
14 #else
15 #pragma GCC optimize("O3,unroll-loops")
16 #pragma GCC target("avx,avx2,sse,sse2,sse3,sse4,popcnt")
17 #define debug(...) void()
18 #define pary(...) void()
19 #endif
20
21 const int MAX_N = 5e5+10;
22 const int INF = 2e18;
23
24 int n, tmp;
25 vector<int> v;
26
27 void solve1(){
28
29     return;
30 }
31
32 signed main(){
33
34     fastio;
35
36     int t = 1;
37     while (t--){
38         solve1();
39     }
40
41     return 0;
42 }

```

7.3 Fast-Input

```

1 // fast IO
2 inline char readchar(){
3     static char buffer[BUFSIZ], *now = buffer + BUFSIZ, *
4     end = buffer + BUFSIZ;
5     if (now == end)
6     {
7         if (end < buffer + BUFSIZ)
8             return EOF;
9         end = (buffer + fread(buffer, 1, BUFSIZ, stdin));
10        now = buffer;
11    }
12    return *now++;
13 }
14 inline int nextint(){
15     int x = 0, c = readchar(), neg = false;
16     while (('0' > c || c > '9') && c != '-' && c != EOF) c =
17         readchar();
18     if (c == '-') neg = true, c = readchar();
19     while ('0' <= c && c <= '9') x = (x<<3) + (x<<1) + (c^'0')
20         , c = readchar();
21     if (neg) x = -x;
22     return x; // returns 0 if EOF
23 }

```

7.4 Radix-Sort

```

1 // 值域限制: 0 ~ 1073741823(2^30-1)
2 inline void radix_sort(vector<int> &a, int n){
3     static int cnt[32768] = {0};
4     vector<int> tmpa(n);
5     for(int i = 0; i < n; ++i)
6         ++cnt[a[i] & 32767];
7     for(int i = 1; i < 32768; ++i)
8         cnt[i] += cnt[i-1];
9     static int temp;
10    for(int i = n-1; i >= 0; --i){
11        temp = a[i] & 32767;
12        --cnt[temp];
13        tmpa[cnt[temp]] = a[i];
14    }
15
16    static int cnt2[32768] = {0};
17    for(int i = 0; i < n; ++i)
18        ++cnt2[(tmpa[i]>>15)];
19    for(int i = 1; i < 32768; ++i)
20        cnt2[i] += cnt2[i-1];
21
22    for(int i = n-1; i >= 0; --i){
23        temp = (tmpa[i]>>15);
24        --cnt2[temp];
25        a[cnt2[temp]] = tmpa[i];
26    }
27    return;
28 }

```

7.5 Set-Pq-Sort

```

1 // priority_queue
2 struct cmp{
3     bool operator () (Data a, Data b){
4         return a.x<b.x;
5     }
6 };
7 priority_queue<Data, vector<Data>, cmp> pq;
8
9 // set
10 struct Data{
11     int x;
12
13     bool operator < (const Data &b){
14         return x<b.x;
15     }
16 };

```

7.6 2-SAT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct TWO_SAT {
5     int n, N;
6     vector<vector<int>> G, rev_G;
7     deque<bool> used;

```

```

8     vector<int> order, comp;
9     deque<bool> assignment;
10    void init(int _n) {
11        n = _n;
12        N = _n * 2;
13        G.resize(N + 5);
14        rev_G.resize(N + 5);
15    }
16    void dfs1(int v) {
17        used[v] = true;
18        for (int u : G[v]) {
19            if (!used[u])
20                dfs1(u);
21        }
22        order.push_back(v);
23    }
24    void dfs2(int v, int c1) {
25        comp[v] = c1;
26        for (int u : rev_G[v]) {
27            if (comp[u] == -1)
28                dfs2(u, c1);
29        }
30    }
31    bool solve() {
32        order.clear();
33        used.assign(N, false);
34        for (int i = 0; i < N; ++i) {
35            if (!used[i])
36                dfs1(i);
37        }
38        comp.assign(N, -1);
39        for (int i = 0, j = 0; i < N; ++i) {
40            int v = order[N - i - 1];
41            if (comp[v] == -1)
42                dfs2(v, j++);
43        }
44        assignment.assign(n, false);
45        for (int i = 0; i < N; i += 2) {
46            if (comp[i] == comp[i + 1])
47                return false;
48            assignment[i / 2] = (comp[i] > comp[i + 1]);
49        }
50        return true;
51    }
52    void add_disjunction(int a, bool na, int b, bool nb) { //
53        // A or B
54        // na means whether a is negative or not
55        // nb means whether b is negative or not
56        a = 2 * a ^ na;
57        b = 2 * b ^ nb;
58        int neg_a = a ^ 1;
59        int neg_b = b ^ 1;
60        G[neg_a].push_back(b);
61        G[neg_b].push_back(a);
62        rev_G[b].push_back(neg_a);
63        rev_G[a].push_back(neg_b);
64        return;
65    }
66    void get_result(vector<int>& res) {
67        res.clear();
68        for (int i = 0; i < n; ++i)
69            res.push_back(assignment[i]);
70    }
71 };
72 /* CSES Giant Pizza
73 3 5

```

```

73 + 1 + 2
74 - 1 + 3
75
76 - + + + -
77 */
78 int main() {
79     int n, m;
80     cin >> n >> m;
81     TWO_SAT E;
82     E.init(m);
83
84     char c1, c2;
85     int inp1, inp2;
86     for (int i = 0; i < n; i++) {
87         cin >> c1 >> inp1;
88         cin >> c2 >> inp2;
89         E.add_disjunction(inp1 - 1, c1 == '-', inp2 - 1, c2
90             == '-');
91     }
92     bool able = E.solve();
93     if (able) {
94         vector<int> ans;
95         E.get_result(ans);
96         for (int i : ans)
97             cout << (i == true ? '+' : '-') << ' ';
98         cout << '\n';
99     } else {
100        cout << "IMPOSSIBLE\n";
101    }
102
103    return 0;
104 }

```

7.7 Enumerate-Subset

```

1 // 時間複雜度 O(3^n)
2 // 枚舉每個 mask 的子集
3 for (int mask=0; mask<(1<<n); mask++){
4     for (int s=mask; s>=0; s=(s-1)&m){
5         // s 是 mask 的子集
6         if (s==0) break;
7     }
8 }

```

8 String

8.1 Hash

```

1 mt19937 rnd(chrono::steady_clock::now().time_since_epoch().
2     count());
3 int A = rnd(), B = 1000000007;
4
5 vector<int> myPow, myPre;
6 void hash_init(string s){
7     myPow.resize(s.size());
8     myPre.resize(s.size());

```

```

8   for (int i=0 ; i<s.size() ; i++){
9       if (i==0){
10          myPow[i] = 1;
11          myPre[i] = s[i];
12      }else{
13          myPow[i] = myPow[i-1]*A%B;
14          myPre[i] = (myPre[i-1]*A+s[i])%B;
15      }
16  }
17  }
18  return;
19 }
20 }
21 }
22 int hash_value(int l, int r){ // 取得 s[l..r] 的數值
23     if (l==0) return myPre[r];
24     return ((myPre[r]-myPre[l-1]*myPow[r-l+1])%B+B)%B;
25 }

```

8.2 Z-Function

```

1 vector<int> z_value;
2 void z_function(string s){
3     z_value.resize(s.size());
4     int ll = 0, rr = 0;
5
6     for (int i=1 ; i<s.size() ; i++){
7         int j = 0;
8
9         if (i<rr) j = min(z_value[i-ll], rr-i);
10        while (s[j]==s[i+j]) j++;
11        z_value[i] = j;
12
13        if (i+j>rr){
14            ll = i;
15            rr = i+j;
16        }
17    }
18
19    z_value[0] = s.size();
20    return;
21 }

```

8.3 Suffix-Array

```

1  /*
2  s = temmie
3  pos = 6 5 1 4 3 2 0
4  rnk = 6 2 5 4 3 1 0
5
6  pos:
7  (空字串) -> e -> emmie -> ie -> mie -> mmie -> temmie
8
9  rnk[i] = i 在 pos 的哪個位置 (第幾小的陣列)
10 */
11
12 vector<int> pos, rnk;
13
14 void Radix_Sort(vector<array<int, 3>> &v){

```

```

15     int n = v.size();
16
17     for (int p=1 ; p>=0 ; p--){
18         vector<int> cnt(n);
19         for (auto x : v){
20             cnt[x[p]]++;
21         }
22
23         vector<array<int, 3>> tmp(n);
24         vector<int> ptr(n); // ptr[i] = 目前 second 是 i 的元
25                                素要放在哪個位置
26         ptr[0] = 0;
27         for (int i=1 ; i<n ; i++){
28             ptr[i] = ptr[i-1]+cnt[i-1];
29         }
30         for (auto x : v){
31             int i = x[p];
32             tmp[ptr[i]] = x;
33             ptr[i]++;
34         }
35         v = tmp;
36     }
37     return;
38 }
39 void Build_SA(string s){
40     s += '$';
41     int n = s.size();
42     rnk.resize(n);
43     pos.resize(n);
44
45     vector<array<int, 2>> tmp(n);
46     for (int i=0 ; i<n ; i++) tmp[i] = {s[i], i};
47     sort(tmp.begin(), tmp.end());
48     for (int i=0 ; i<n ; i++) pos[i] = tmp[i][1];
49     rnk[pos[0]] = 0;
50     for (int i=1 ; i<n ; i++){
51         if (tmp[i][0]==tmp[i-1][0]) rnk[pos[i]] = rnk[pos[i-1]];
52         else rnk[pos[i]] = rnk[pos[i-1]]+1;
53     }
54
55     for (int k=0 ; (1<<k)<n ; k++){
56         vector<array<int, 3>> tmp(n);
57         for (int i=0 ; i<n ; i++) tmp[i] = {rnk[i], rnk[(i
58             +(1<<k))%n], i};
59         Radix_Sort(tmp);
60         for (int i=0 ; i<n ; i++) pos[i] = tmp[i][2];
61         rnk[pos[0]] = 0;
62         for (int i=1 ; i<n ; i++){
63             if (tmp[i][0]==tmp[i-1][0] && tmp[i][1]==tmp[i-1][1]) rnk[pos[i]] = rnk[pos[i-1]];
64             else rnk[pos[i]] = rnk[pos[i-1]]+1;
65         }
66     }

```

8.4 Longest-Common-Prefix-Array

```

1  /*
2
3  rnk:

```

```

4  (空字串) -> e -> emmie -> ie -> mie -> mmie -> temmie
5
6  lcp[i] = 第 i 小的後綴跟 i-1 小的後綴的 Lcp (based-1)
7  [(無意義, -1), 0, 1, 0, 0, 1, 0]
8  */
9  vector<int> pos, rnk;
10 vector<int> lcp;
11
12 void Build_LCP(string s){
13     int n = s.size();
14     s += '$';
15     lcp.resize(s.size());
16     lcp[0] = -1;
17
18     int k = 0;
19     for (int i=0 ; i<n ; i++){
20         int pi = rnk[i];
21         int j = pos[pi-1];
22
23         while (i+k<n && j+k<n && s[i+k]==s[j+k]) k++;
24
25         lcp[pi] = k;
26         k = max((int)0, k-1);
27     }
28 }

```

8.5 Manacher

```

1 string Manacher(string str) {
2     string tmp = "$#";
3     for (char i : str) {
4         tmp += i;
5         tmp += '#';
6     }
7
8     vector<int> p(tmp.size(), 0);
9     int mx = 0, id = 0, len = 0, center = 0;
10    for (int i=1 ; i<(int)tmp.size() ; i++) {
11        p[i] = mx > i ? min(p[id*2-i], mx-i) : 1;
12
13        while (tmp[i+p[i]] == tmp[i-p[i]]) p[i]++;
14        if (mx<i+p[i]) mx = i+p[i], id = i;
15        if (len<p[i]) len = p[i], center = i;
16    }
17    return str.substr((center-len)/2, len-1);
18 }

```


Contents

1	Convolution	1	4	Geometry	5	6.8	中國剩餘定理 (m 互質)	10
1.1	FFT	1	4.1	Point-Struct	5	6.9	中國剩餘定理 (m 不互質)	11
1.2	NTT-998244353	1	4.2	Line-Intersection	6	6.10	歐拉公式	11
1.3	FFT-mod	1	4.3	Pick's-Theorem	6	6.11	卡特蘭數	11
1.4	FFT-2	2	4.4	Convex-Hull	6	6.12	歐拉定理	11
2	Data-Structure	2	5	Graph	6	6.13	Fraction	11
2.1	GP-Hash-Table	2	5.1	Find-Bridge	6	6.14	錯排公式	12
2.2	Sparse-Table	2	5.2	Find-AP	6	6.15	Quick-Pow	12
2.3	Order-Set	2	5.3	SCC 與縮點	6	6.16	二元一次方程式	12
2.4	BIT	2	5.4	MCMF	7	6.17	Josephus	12
2.5	Add-Set-Segment-Tree	2	5.5	HLD	7	7	Misc	12
2.6	Treap	3	5.6	Tree-Isomorphism	7	7.1	Xor-Basis	12
2.7	Persistent-Segment-Tree	3	5.7	Bridge BCC	8	7.2	Default-Code	12
2.8	Trie	4	5.8	Cut BCC	8	7.3	Fast-Input	12
2.9	LC-Segment-Tree	4	5.9	圓方樹	8	7.4	Radix-Sort	13
2.10	Persistent-Disjoint-Set	5	5.10	Dijkstra	9	7.5	Set-Pq-Sort	13
3	Dynamic-Programming	5	6	Math	10	7.6	2-SAT	13
3.1	SOS-DP	5	6.1	Burnside's-Lemma	10	7.7	Enumerate-Subset	13
3.2	Digit-DP	5	6.2	線性篩	10	8	String	13
3.3	整數拆分	5	6.3	Lucas's-Theorem	10	8.1	Hash	13
			6.4	Miller-Rabin	10	8.2	Z-Function	14
			6.5	Matrix	10	8.3	Suffix-Array	14
			6.6	Pollard-Rho	10	8.4	Longest-Common-Prefix-Array	14
			6.7	最大質因數	10	8.5	Manacher	14