

1 Data Structure

1.1 Segment Tree

```

1 struct SegT{
2     const int MAXN = 1e5+5;
3     int tr[MAXN*4], arr[MAXN], tag[MAXN*4];
4
5     int combine(int a, int b){
6         return max(a,b);
7     }
8
9     void build(int idx, int l, int r){
10        if(l==r){
11            tr[idx] = arr[l];
12        }else{
13            int m = (l+r)/2;
14            build(idx*2,l,m);
15            build(idx*2+1,m+1,r);
16            tr[idx] = combine(tr[idx*2],tr[idx
17                                *2+1]);
18        }
19
20        void push(int idx){
21            if(tag[idx]){
22                tr[idx<<1] = max(tr[idx<<1], tag[
23                    idx]);
24                tr[idx<<1|1] = max(tr[idx<<1|1],
25                    tag[idx]);
26                tag[idx<<1] = max(tag[idx<<1], tag
27                    [idx]);
28                tag[idx<<1|1] = max(tag[idx<<1|1],
29                    tag[idx]);
30                tag[idx] = 0;
31            }
32
33            void modify(int ql, int qr, int val, int
34                idx, int l, int r){
35                if(l!=r) push(idx); //當節點並非葉節點
36                時，下推標記
37                if(ql <= l && r <= qr){
38                    tr[idx] = max(tr[idx],val);
39                    tag[idx] = max(tag[idx],val);
40                    return;
41                }
42                int m = (l+r)/2;
43                if(qr > m) modify(ql, qr, val, idx
44                    *2+1, m+1, r);
45                if(ql <= m) modify(ql, qr, val, idx*2,
46                    l, m);
47                tr[idx] = combine(tr[idx<<1],tr[idx
48                    <<1|1]);
49            }
50
51            int query(int ql, int qr, int idx, int l,
52                int r){
53                if(l!=r) push(idx);
54                if(ql <= l && r <= qr){
55                    return tr[idx];
56                }
57                int m = (l+r)/2;

```

```

49         if(ql > m){
50             return query(ql, qr, idx*2+1, m+1,
51                 r);
52         }
53         if(qr <= m){
54             return query(ql, qr, idx*2, l, m);
55         }
56         return combine(query(ql, qr, idx*2, l,
57             m), query(ql, qr, idx*2+1, m+1, r
58             ));
59     }
60 }

```

1.2 Treap

```

1 struct Treap{
2     Treap *l, *r;
3     int val, size, sum;
4     Treap(int v): l(nullptr), r(nullptr), val(
5         v), size(1), sum(v){}
6     void pull();
7
8     void Treap::pull(){
9         size = 1, sum = val;
10        if(l!=nullptr) size += l->size, sum += l->
11            sum;
12        if(r!=nullptr) size += r->size, sum += r->
13            sum;
14    }
15    int sz(Treap *t){
16        return (t==nullptr ? 0 : t->size);
17    }
18    Treap *merge(Treap *a, Treap *b){
19        if(a==nullptr) return b;
20        if(b==nullptr) return a;
21        if(rand()%(a->size+b->size) < a->size){
22            a->r = merge(a->r,b);
23            a->pull();
24            return a;
25        }else{
26            b->l = merge(a,b->l);
27            b->pull();
28            return b;
29        }
30    }
31
32    void split(Treap *t, Treap *&a, Treap *&b,
33        int k){
34        if(t==nullptr){
35            a = b = nullptr;
36            return;
37        }
38        if(sz(t->l) < k){
39            a = t;
40            split(t->r,a->r,b,k-sz(t->l)-1);
41            a->pull();
42        }else{
43            b = t;
44            split(t->l,a,b->l,k);
45            b->pull();

```

2 Graphs

2.1 BCC_Edge

```

1 void dfs(int u, int a){
2     d[u] = l[u] = t++;
3     st.emplace(u);
4     for(int v : adj[u]){
5         if(v == a) continue;
6         if(!d[v]){
7             dfs(v,u);
8             //if(d[u] < l[v]) bridges.emplace_back
9                 (u,v);
10            l[u] = min(l[u],l[v]);
11        }
12        l[u] = min(l[u],d[v]);
13    }
14    if(l[u]==d[u]){
15        bccid++;
16        int tmp;
17        do{
18            tmp = st.top(); st.pop();
19            bcc[tmp] = bccid;
20        }while(tmp!=u);
21    }

```

2.2 BCC_Vertex

```

1 const int N = 2e5+5;
2 vector<int> adj[N], adj2[N], bccids[N];
3
4 int low[N], dfn[N], bccid[N], bcccnt, w[N],
5     cut[N], inst[N], t, tt;
6
7 void dfs(int u, int par){
8     low[u] = dfn[u] = ++t;
9     int cnt = 0;
10    st.push(u);
11    for(auto v : adj[u]){
12        if(v == par) continue;
13        if(!dfn[v]){
14            dfs(v,u); cnt++;
15            low[u] = min(low[u],low[v]);
16            if(low[v] >= dfn[u]){
17                cut[u] = 1;
18                ++bcccnt;
19                int x;
20                do{
21                    if(st.empty()) break;
22                    x = st.top(); st.pop();
23                    //if(bccid[x]) bcc[bccid[x]].erase
24                        (bcc[bccid[x]].find(w[x]));
25                    bccids[x].push_back(bcccnt);

```

```

25        bccid[x] = bcccnt;
26    }while(x!=v);
27    //if(bccid[u]) bcc[bccid[u]].erase(
28        bcc[bccid[u]].find(w[u]));
29    bccids[u].push_back(bcccnt);
30    bccid[u] = bcccnt;
31    }else if(dfn[v] < dfn[u]){
32        low[u] = min(low[u],dfn[v]);
33    }
34    }
35    if(par!=-1&&cnt < 2) cut[u] = 0;
36 }

```

2.3 dijkstra

```

1 priority_queue<pair<int,int>,vector<pair<int
2     ,int>>, greater<pair<int,int>>> pq;
3 pq.push({0,s});
4 dis[s] = 0;
5 inq[s] = 1;
6 while(!pq.empty()){
7     auto [w,u] = pq.top(); pq.pop();
8     inq[u] = 0;
9     for(auto [v,w] : adj[u]){
10        if(dis[v] > dis[u]+ w){
11            dis[v] = dis[u]+w;
12            pq.push({dis[v],v});
13            inq[v] = 1;
14        }
15    }
16 }
17 }

```

2.4 SCC_korasaju

```

1 const int N = 1e6+5;
2 vector<int> adj[N], adj2[N];
3 int vis[N], scc[N], id;
4 stack<int> st;
5
6 void dfs1(int u){
7     vis[u] = true;
8     for(auto v : adj[u]){
9         if(!vis[v]) dfs1(v);
10    }
11    st.push(u);
12 }
13
14 void dfs2(int u){
15     scc[u] = id;
16     for(auto v : adj2[u]){
17         if(!scc[v]) dfs2(v);
18    }
19 }

```

2.5 SCC_tarjan

```

1 const int N = 1e6+5;
2 vector<int> adj[N];
3 int dfn[N], low[N], inst[N], scc[N], sccid =
  0, cnt = 0;
4 stack<int> st;
5
6 void dfs(int u){
7     dfn[u] = low[u] = ++cnt;
8     st.push(u);
9     inst[u] = 1;
10    for(auto v : adj[u]){
11        if(!dfn[v]){
12            dfs(v);
13            low[u] = min(low[u], low[v]);
14        }else if(inst[v]){
15            low[u] = min(low[u], dfn[v]);
16        }
17    }
18    if(low[u]==dfn[u]){
19        int x;
20        do{
21            x = st.top();
22            st.pop();
23            scc[x] = sccid;
24            inst[x] = 0;
25        }while(x!=u);
26        sccid++;
27    }
28 }

```

3 Number Theory

3.1 FFT

```

1 typedef complex<double> cp;
2
3 const double pi = acos(-1);
4 const int NN = 131072;

```

```

5 struct FastFourierTransform{
6     /*
7     Iterative Fast Fourier Transform

```

How this works? Look at this

```

12 0th recursion 0(000) 1(001) 2(010)
              3(011) 4(100) 5(101) 6(110)
              7(111)
13 1th recursion 0(000) 2(010) 4(100)
              6(110) | 1(011) 3(011) 5(101)
              7(111)
14 2th recursion 0(000) 4(100) | 2(010)
              6(110) | 1(011) 5(101) | 3(011)
              7(111)
15 3th recursion 0(000) | 4(100) | 2(010) |
              6(110) | 1(011) | 5(101) | 3(011) |
              7(111)

```

```

17 All the bits are reversed => We can save
   the reverse of the numbers in an
   array!
18
19 /*
20 int n, rev[NN];
21 cp omega[NN], iomega[NN];
22 void init(int n_){
23     n = n_;
24     for(int i = 0; i < n; i++){
25         //Calculate the nth roots of unity
26         omega[i] = cp(cos(2*pi*i/n_), sin(2*pi*
27             i/n_));
28         iomega[i] = conj(omega[i]);
29     }
30     int k = __lg(n);
31     for(int i = 0; i < n; i++){
32         int t = 0;
33         for(int j = 0; j < k; j++){
34             if(i & (1<<j)) t |= (1<<(k-j-1));
35         }
36         rev[i] = t;
37     }
38 }
39 void transform(vector<cp> &a, cp* xomega){
40     for(int i = 0; i < n; i++){
41         if(i < rev[i]) swap(a[i], a[rev[i]]);
42     }
43     for(int len = 2; len <= n; len <= 1){
44         int mid = len >> 1;
45         int r = n/len;
46         for(int j = 0; j < n; j += len){
47             for(int i = 0; i < mid; i++){
48                 cp tmp = xomega[r*i] * a[j+mid+i];
49                 a[j+mid+i] = a[j+i] - tmp;
50                 a[j+i] = a[j+i] + tmp;
51             }
52         }
53     }
54 void fft(vector<cp> &a){ transform(a, omega
55 ); }
56 void ifft(vector<cp> &a){ transform(a,
57 iomega); for(int i = 0; i < n; i++) a[i]
58 /= n; }
59 } FFT;

```

3.2 NTT

```

1 const int N = 1e5+5, MOD = 998244353, G = 3;
2
3 int fastpow(int n, int p){
4     int res = 1;
5     while(p){
6         if(p&1) res = res * n % MOD;
7         n = n * n % MOD;
8         p >>= 1;
9     }
10    return res;
11 }
12
13 struct NTT{
14     int n, inv, rev[N];
15     int omega[N], iomega[N];

```

```

73 void init(int n_){
74     n = 1;
75     while(n < n_) n<<=1;
76     inv = fastpow(n, MOD-2);
77     int k = __lg(n);
78     int x = fastpow(G, (MOD-1)/n);
79     omega[0] = 1;
80     for(int i = 1; i < n; i++){
81         omega[i] = omega[i-1] * x % MOD;
82         iomega[n-1] = fastpow(omega[n-1], MOD
83             -2);
84     }
85     for(int i = n-2; i >= 0; i--){
86         iomega[i] = iomega[i+1] * x %
87             MOD;
88     }
89     for(int i = 0; i < n; i++){
90         int t = 0;
91         for(int j = 0; j < k; j++){
92             if(i&(1<<j)) t |= (1<<(k-j-1));
93         }
94         rev[i] = t;
95     }
96 }
97 void transform(int *a, int *xomega){
98     for(int i = 0; i < n; i++){
99         if(i < rev[i]) swap(a[i], a[rev[i]
100             ]);
101     }
102     for(int len = 2; len <= n; len <= 1){
103         int mid = len >> 1;
104         int r = n/len;
105         for(int j = 0; j < n; j += len){
106             for(int i = 0; i < mid; i++){
107                 int tmp = xomega[r*i] *
108                     a[j+mid+i] % MOD;
109                 a[j+mid+i] = (a[j+i] -
110                     tmp + MOD) % MOD;
111                 a[j+i] = (a[j+i] + tmp) %
112                     MOD;
113             }
114         }
115     }
116 }
117 void dft(int *a){transform(a, omega);}
118 void idft(int *a){transform(a, iomega);}
119 for(int i = 0; i < n; i++) a[i] = a[i]
120     *inv % MOD; }
121
122 int tmp[8][N];
123
124 void copy_(int *a, int *b, int m){
125     for(int i = 0; i < m; i++){
126         a[i] = b[i];
127     }
128 }
129 void copy(int *a, int *b, int m){
130     for(int i = 0; i < m; i++){
131         a[i] = b[i];
132     }
133 }
134 //B_{k+1} = B_k(2-AB_k) (mod MOD)
135 void inverse(int *a, int *b, int m){
136     //Uses tmp[0], tmp[1]
137     if(m==1){
138         b[0] = fastpow(a[0], MOD-2);
139         return;

```

```

140     }
141     inverse(a, b, m >> 1);
142     init(m << 1);
143     copy_(tmp[0], a, m); copy_(tmp[1], b, m
144         >> 1);
145     dft(tmp[0]); dft(tmp[1]);
146     for(int i = 0; i < n; i++){
147         tmp[0][i] = (tmp[0][i] * tmp[1][i]
148             % MOD + MOD) % MOD;
149     }
150     idft(tmp[0]);
151     copy(b, tmp[0], m);
152 }
153
154 //Q_{k+1} = pow(2, MOD-2)(Q_k + P*pow(Q_k
155     , MOD-2)) (mod MOD)
156 void sqrt(int *a, int *b, int m){
157     //Uses tmp[2], tmp[3]
158     if(m==1){
159         b[0] = 1;
160         return;
161     }
162     sqrt(a, b, m >> 1);
163     for(int i = m; i < m << 1; i++){
164         b[i] = 0;
165     }
166     inverse(b, tmp[2], m);
167     init(m << 1);
168     for(int i = m; i < m << 1; i++){
169         b[i] = tmp[2][i] = 0;
170     }
171     int inv2 = fastpow(2, MOD-2);
172     copy_(tmp[3], a, m);
173     dft(tmp[3]); dft(tmp[2]);
174     for(int i = 0; i < n; i++){
175         tmp[3][i] = tmp[3][i] * tmp[2][i] %
176             MOD;
177     }
178     idft(tmp[3]);
179     for(int i = 0; i < m; i++){
180         b[i] = (b[i] + tmp[3][i]) % MOD * inv2
181             % MOD;
182     }
183 }
184
185 void derivative(int *a, int *b, int m){
186     for(int i = 1; i < m; i++){
187         b[i-1] = a[i]
188             *i % MOD;
189     }
190     b[m-1] = 0;
191 }
192
193 void integral(int *a, int *b, int m){
194     for(int i = m-1; i >= 0; i--){
195         b[i] = a[i]
196             *fastpow(i, MOD-2) % MOD;
197     }
198     b[0] = 0;
199 }
200
201 void ln(int *a, int *b, int m){
202     //Uses tmp[4], tmp[5]
203     inverse(a, b, m);
204     derivative(a, tmp[5], m);
205     init(m << 1);
206     copy_(tmp[4], b, m); copy_(tmp[5], tmp
207         [5], m);
208     dft(tmp[4]); dft(tmp[5]);
209     for(int i = 0; i < m << 1; i++){
210         tmp[4][i] = tmp[4][i] * tmp[5][i] %
211             MOD;
212     }
213     idft(tmp[4]);
214     integral(tmp[4], b, m);

```

```
129     }
130
131     void exp(int *a, int *b, int m){
132         //Uses tmp[6], tmp[7]
133         b[0] = 1;
134         for(int i = 4, j = 2; j <= m; j = i, i
            <<=1){
135             ln(b, tmp[6], j);
136             tmp[6][0] = (a[0]+1-tmp[6][0]+
                MOD)%MOD;
137             for(int k = 1; k < j; k++) tmp
                [6][k] = (a[k]-tmp[6][k]+MOD
                )%MOD;
138
139             fill(tmp[6]+j, tmp[6]+i, 0);
140             dft(b), dft(tmp[6]);
141             for(int k = 0; k < i; k++) b[k] =
                b[k]*tmp[6][k]%MOD;
142             idft(b);
143             fill(b+j, b+i, 0);
144         }
145     }
146
147 } NTT;
```

KEEP ON THE HARD WORK!

Contents	
1 Data Structure	1
1.1 Segment Tree	1
1.2 Treap	1
2 Graphs	1
2.1 BCC_Edge	1
2.2 BCC_Vertex	1
2.3 dijkstra	1
2.4 SCC_korasaju	1
2.5 SCC_tarjan	2
3 Number Theory	2
3.1 FFT	2
3.2 NTT	2