

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                int query(int ql, int qr, int idx, int l,
51                    int r){
52                    if(l==r) push(idx);
53                    if(ql <= l && r <= qr){
54                        return tr[idx];
55                    }
56                    int m = (l+r)/2;
57

```

```

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
16    int sz(Treap *t){
17        return (t==nullptr ? 0 : t->size);
18    }
19
20    Treap *merge(Treap *a, Treap *b){
21        if(a==nullptr) return b;
22        if(b==nullptr) return a;
23        if(rand()%<math>a->size+b->size</math> < a->size){
24            a->r = merge(a->r,b);
25            a->pull();
26            return a;
27        }else{
28            b->l = merge(a,b->l);
29            b->pull();
30            return b;
31        }
32    }
33
34    void split(Treap *t, Treap *&a, Treap *&b,
35        int k){
36        if(t==nullptr){
37            a = b = nullptr;
38            return;
39        }
40        if(sz(t->l) < k){
41            a = t;
42            split(t->r,a->r,b,k-sz(t->l)-1);
43            a->pull();
44        }else{
45            b = t;
46            split(t->l,a,b->l,k);
47            b->pull();
48        }
49    }

```

2 Flow

2.1 Dinic

```

1 struct Dinic {
2     const int INF = 1e18;
3     struct edge {
4         int u, v, cap, flow;
5         edge(int u, int v, int cap): u(u), v
6             (v), cap(cap), flow(0) {}
7     };
8     vector<edge> edges;
9     vector<vector<int>> adj;
10    vector<int> level, num;
11    queue<int> q;
12    int n, s, t, cnt = 0; //To maintain the
13        id of edges
14
15    void init(int nn, int ss, int tt) {
16        n = nn + 1, s = ss, t = tt;
17        adj.resize(n);
18        level.resize(n);
19        num.resize(n);
20    }
21
22    void add_edge(int u, int v, int cap) {
23        edges.push_back({u, v, cap});
24        edges.push_back({v, u, 0});
25        adj[u].push_back(cnt++);
26        adj[v].push_back(cnt++);
27    }
28
29    bool bfs() {
30        fill(level.begin(), level.end(), -1)
31        ;
32        level[s] = 0;
33        q.push(s);
34
35        while (!q.empty()) {
36            int u = q.front();
37            q.pop();
38
39            for (auto eid : adj[u]) {
40                edge e = edges[eid];
41
42                //We only pass the edge that
43                has positive capacity
44                if (e.cap - e.flow <= 0 ||
45                    level[e.v] != -1)
46                    continue;
47
48                level[e.v] = level[u] + 1;
49                q.push(e.v);
50            }
51        }
52
53        //If we cannot reach t, then there
54        is no Augmenting Path
55    }

```

```

49        return level[t] != -1;
50    }
51
52    int dfs(int u, int now) {
53        if (now == 0)
54            return 0;
55
56        if (u == t)
57            return now;
58
59        for (; num[u] < adj[u].size(); num[u]
60            ]++) {
61            edge e = edges[adj[u][num[u]]];
62
63            if (level[e.v] != level[u] + 1
64                || e.cap - e.flow <= 0)
65                continue;
66
67            int f = dfs(e.v, min(now, e.cap
68                - e.flow));
69
70            if (!f)
71                continue;
72
73            edges[adj[u][num[u]]].flow += f;
74            edges[adj[u][num[u]] ^ 1].flow
75                -= f;
76            return f;
77        }
78
79        return 0;
80
81    int get_flow() {
82        int res = 0, now;
83
84        while (true) {
85            if (!bfs())
86                break;
87
88            fill(num.begin(), num.end(), 0);
89
90            while (now = dfs(s, INF)) {
91                res += now;
92            }
93
94            return res;
95        }
96    } flow;

```

2.2 mcmf

```

1 struct MCMF {
2     static const int N = 500, INF = 1e18;
3     int n, s, t;
4     struct Edge {
5         int v, cap, f, cost;
6         Edge(int v, int cap, int f, int cost
7             ): v(v), cap(cap), f(f), cost(
8                 cost) {}
9     };
10    vector<Edge> edges;

```

```

9  vector<vector<int>> adj;
10
11 void init(int n_, int s_, int t_) {
12     n = n_ + 1, s = s_, t = t_;
13     adj.resize(n);
14 }
15
16 void add_edge(int u, int v, int cap, int
17     cost) {
18     adj[u].push_back(edges.size());
19     edges.push_back({v, cap, 0, cost});
20     adj[v].push_back(edges.size());
21     edges.push_back({u, 0, 0, -cost});
22 }
23
24 int dis[N], p[N], pe[N], inque[N];
25
26 void SPFA() {
27     for (int i = 0; i <= n; i++)
28         dis[i] = INF, p[i] = -1, pe[i] =
29         -1, inque[i] = 0;

```

```

30     dis[s] = 0;
31     queue<int> q;
32     q.push(s);
33     inque[s] = 1;

```

```

34     while (!q.empty()) {
35         int u = q.front();
36         q.pop();
37         inque[u] = false;
38
39         for (auto x : adj[u]) {
40             auto e = edges[x];
41
42             if (e.cap > 0 && dis[e.v] >
43                 dis[u] + e.cost) {
44                 dis[e.v] = dis[u] + e.
45                 cost;
46                 p[e.v] = u;
47                 pe[e.v] = x;
48
49                 if (!inque[e.v])
50                     inque[e.v] = true, q
51                     .push(e.v);
52             }
53         }
54     }

```

```

55     pair<int, int> min_cost() {
56         int flow = 0, cost = 0;
57
58         while (true) {
59             SPFA();
60
61             if (dis[t] == INF)
62                 break;
63
64             int min_flow = INF;
65
66             int now = t;
67
68             while (now != s) {
69                 min_flow = min(min_flow,
70                     edges[pe[now]].cap);

```

```

69         now = p[now];
70     };
71
72     flow += min_flow, cost +=
73     min_flow * dis[t];
74
75     now = t;
76
77     while (now != s) {
78         edges[pe[now]].cap -=
79         min_flow;
80         edges[pe[now] ^ 1].cap +=
81         min_flow;
82         now = p[now];
83     }
84
85     return {flow, cost};
} flow;

```

3 Graphs

3.1 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    }
}

```

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

```

3.3 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            if(!inq[v]){
13                pq.push({dis[v],v});
14                inq[v] = 1;
15            }
16        }
17    }
}

```

3.4 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 }

```

3.5 SCC_tarjan

```

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

```

4 Number Theory

4.1 FFT

```

1 typedef complex<double> cp;
2
3 const double pi = acos(-1);
4 const int NN = 131072;
5
6 struct FastFourierTransform{
7     /*

```

Iterative Fast Fourier Transform

How this works? Look at this

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

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

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

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

All the bits are reversed => We can save
the reverse of the numbers in an
array!

```

*/
int n, rev[NN];
cp omega[NN], iomega[NN];
void init(int n_){
    n = n_;
    for(int i = 0; i < n; i++){
        //Calculate the nth roots of unity
        omega[i] = cp(cos(2*pi*i/n_), sin(2*pi*
            i/n_));
        iomega[i] = conj(omega[i]);
    }
    int k = __lg(n_);
    for(int i = 0; i < n; i++){
        int t = 0;
        for(int j = 0; j < k; j++){
            if(i & (1<<j)) t |= (1<<(k-j-1));
        }
        rev[i] = t;
    }
}

void transform(vector<cp> &a, cp* xomega){
    for(int i = 0; i < n; i++){
        if(i < rev[i]) swap(a[i], a[rev[i]]);
        for(int len = 2; len <= n; len <= 1){
            int mid = len >> 1;
            int r = n/len;
            for(int j = 0; j < n; j += len)
                for(int i = 0; i < mid; i++){
                    cp tmp = xomega[r*i] * a[j+mid+i];
                    a[j+mid+i] = a[j+i] - tmp;
                    a[j+i] = a[j+i] + tmp;
                }
        }
    }
}

void fft(vector<cp> &a){ transform(a, omega); }
void ifft(vector<cp> &a){ transform(a, iomega); for(int i = 0; i < n; i++) a[i] /= n; }
} FFT;

```

4.2 Linear_Sieve

```

1 const int N = 1e5+5;
2 int k, lpf[N];
3
4 vector<int> primes;
5 void init(){
6     fill(lpf, lpf+N, 1);
7     for(int i = 2; i < N; i++){
8         if(lpf[i]==1){
9             lpf[i] = i;
10            primes.push_back(i);
11        }
12        for(int x : primes){
13            if(x*i > N) break;
14            lpf[x*i]=x;
15            if(x==lpf[i]) break;
16        }
17    }
18 }

```

4.3 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];
16     void init(int n_){
17         n = 1;
18         while(n < n_) n<=1;
19         inv = fastpow(n, MOD-2);
20         int k = __lg(n);
21         int x = fastpow(G, (MOD-1)/n);
22         omega[0] = 1;
23         for(int i = 1; i < n; i++){
24             omega[i] = omega[i-1] * x % MOD;
25             iomega[n-1] = fastpow(omega[n-1], MOD-2);
26             for(int i = n-2; i >= 0; i--){
27                 iomega[i] = iomega[i+1] * x % MOD;
28             }
29             for(int i = 0; i < n; i++){
30                 int t = 0;
31                 for(int j = 0; j < k; j++){
32                     if(i&(1<<j)) t |= (1<<(k-j-1));
33                 }
34                 rev[i] = t;
35             }
36         }
37     }
38     void transform(int *a, int *xomega){
39         for(int i = 0; i < n; i++){
40             for(int i = 0; i < n; i++){
41                 if(i < rev[i]) swap(a[i], a[rev[i]]);
42                 for(int len = 2; len <= n; len <= 1){
43                     int mid = len >> 1;
44                     int r = n/len;
45                     for(int j = 0; j < n; j += len)
46                         for(int i = 0; i < mid; i++){
47                             cp tmp = xomega[r*i] * a[j+mid+i];
48                             a[j+mid+i] = a[j+i] - tmp;
49                             a[j+i] = a[j+i] + tmp;
50                         }
51                 }
52             }
53         }
54     }
55 } FFT;

```

```

37     if(i < rev[i]) swap(a[i], a[rev[i]]);
38     for(int len = 2; len <= n; len <= 1){
39         int mid = len >> 1;
40         int r = n/len;
41         for(int j = 0; j < n; j += len){
42             for(int i = 0; i < mid; i++){
43                 int tmp = xomega[r*i] * a[j+mid+i] % MOD;
44                 a[j+mid+i] = (a[j+i] - tmp + MOD) % MOD;
45                 a[j+i] = (a[j+i] + tmp) % MOD;
46             }
47         }
48     }
49     void dft(int *a){transform(a, omega);}
50     void idft(int *a){transform(a, iomega);}
51     for(int i = 0; i < n; i++) a[i] = a[i] * inv % MOD;
52 }
53
54 int tmp[8][N];
55
56 void copy_(int *a, int *b, int m){
57     for(int i = 0; i < m; i++)
58         a[i] = b[i];
59     for(int i = m; i < n; i++)
60         a[i] = 0;
61 }
62
63 void copy(int *a, int *b, int m){
64     for(int i = 0; i < m; i++)
65         a[i] = b[i];
66 }
67
68 //B_{k+1} = B_k(2-AB_k) (mod MOD)
69 void inverse(int *a, int *b, int m){
70     //Uses tmp[0], tmp[1]
71     if(m==1){
72         b[0] = fastpow(a[0], MOD-2);
73         return;
74     }
75     inverse(a, b, m>>1);
76     init(m<<1);
77     copy_(tmp[0], a, m); copy_(tmp[1], b, m>>1);
78     dft(tmp[0]); dft(tmp[1]);
79     for(int i = 0; i < n; i++) tmp[0][i] =
80         tmp[1][i]*(2-tmp[0][i]*tmp[1][i]
81         %MOD+MOD)%MOD;
82     idft(tmp[0]);
83     copy(b, tmp[0], m);
84 }
85
86 //Q_{k+1} = pow(2, MOD-2)(Q_k + P*pow(Q_k
87     , MOD-2)) (mod MOD)
88 void sqrt(int *a, int *b, int m){
89     //Uses tmp[2], tmp[3]
90     if(m==1){
91         b[0] = 1;
92         return;
93     }
94     sqrt(a, b, m>>1);
95     for(int i = m; i < m<<1; i++){
96         b[i] = 0;
97         inverse(b, tmp[2], m);
98         init(m<<1);
99         for(int i = m; i < m<<1; i++){
100             b[i] = tmp[2][i] = 0;
101             int inv2 = fastpow(2, MOD-2);
102             copy_(tmp[3], a, m);
103             dft(tmp[3]); dft(tmp[2]);
104             for(int i = 0; i < n; i++){
105                 tmp[3][i] = tmp[3][i]*tmp[2][i] % MOD;
106             }
107             idft(tmp[3]);
108             for(int i = 0; i < m; i++){
109                 b[i] = (b[i]+tmp[3][i])%MOD*inv2 %MOD;
110             }
111         }
112     }
113     void derivative(int *a, int *b, int m){
114         for(int i = 1; i < m; i++) b[i-1] = a[i] * i % MOD;
115         b[m-1] = 0;
116     }
117
118     void integral(int *a, int *b, int m){
119         for(int i = m-1; i >= 0; i--) b[i] = a[i]
120             -1]*fastpow(i, MOD-2)%MOD;
121         b[0] = 0;
122     }
123
124     void ln(int *a, int *b, int m){
125         //Uses tmp[4], tmp[5]
126         inverse(a, b, m);
127         derivative(a, tmp[5], m);
128         init(m<<1);
129         copy_(tmp[4], b, m); copy_(tmp[5], tmp[5], m);
130         dft(tmp[4]); dft(tmp[5]);
131         for(int i = 0; i < m<<1; i++) tmp[4][i]
132             = tmp[4][i]*tmp[5][i]%MOD;
133         idft(tmp[4]);
134         integral(tmp[4], b, m);
135     }
136
137     void exp(int *a, int *b, int m){
138         //Uses tmp[6], tmp[7]
139         b[0] = 1;
140         for(int i = 4; j = 2; j <= m; j = i, i
141             <<=1){
142             ln(b, tmp[6], j);
143             tmp[6][0] = (a[0]+1-tmp[6][0]+MOD)%MOD;
144             for(int k = 0; k < j; k++) tmp[6][k] = (a[k]-tmp[6][k]+MOD)%MOD;
145             fill(tmp[6]+j, tmp[6]+i, 0);
146             dft(b); dft(tmp[6]);
147             for(int k = 0; k < i; k++) b[k] = b[k]*tmp[6][k]%MOD;
148             idft(b);
149             fill(b+j, b+i, 0);
150         }
151     }
152 } NTT;

```

5 String

5.1 AC_Automaton

```

1 const int N = 1e3+5, MOD = 1e9+9;
2
3 int dp[N][N][15];
4 map<char,int> m;
5 int arr[N][4], fail[N], cnt[N], pos;
6
7 void insert(string &s){
8     int now = 0;
9     for(auto c : s){
10         int &to = arr[now][m[c]];
11         now = to ? to : to = ++pos;
12     }
13     cnt[now] = s.size();
14 }
15
16 void build(){
17     queue<int> q;
18     for(int i = 0; i < 4; i++){
19         if(arr[0][i]) q.push(arr[0][i]);
20     }
21     while(!q.empty()){
22         int now = q.front(); q.pop();
23         for(int i = 0; i < 4; i++){
24             int &to = arr[now][i], tmp;
25             if(to == 0) to = arr[fail[now]][i];
26             else{
27                 q.push(to), tmp = fail[now];
28                 while(tmp && !arr[tmp][i]) cnt[tmp] =
29                     max(cnt[tmp], cnt[arr[tmp][i]]),
30                     tmp = fail[tmp];
31                 fail[to] = arr[tmp][i];
32                 cnt[to] = max(cnt[to], cnt[arr[tmp][i]]);
33             }
34         }
35     }
36 }
37
38 void match(string &s){
39     int now = 0;
40     for(int i = 0; i < s.size(); i++){
41         while(now && !arr[now][s[i]-'a']) now = fail[now];
42         if(arr[now][s[i]-'a']) now = arr[now][s[i]-'a'];
43         if(cnt[now] > 0){
44             //FIND!
45         }
46     }
47 }

```

5.2 KMP

```

1 const int N = 1e6 + 5;
2 int f[N];
3
4 void build_failure(string s) {

```

```

5     int p = f[0] = -1;
6
7     for (int i = 1; i < s.size(); i++) {
8         while (p != -1 && s[p + 1] != s[i])
9             p = f[p];
10
11         if (s[p + 1] == s[i])
12             p++;
13
14         f[i] = p;
15     }
16 }
17
18 int KMP(string s, string t) {
19     int cnt = 0;
20     int p = -1;
21
22     for (int i = 0; i < t.size(); i++) {
23         while (p != -1 && s[p + 1] != t[i])
24             p = f[p];
25
26         if (s[p + 1] == t[i])
27             p++;
28
29         if (p + 1 == s.size()) {
30             cnt++;
31         }
32     }
33
34     return cnt;
35 }

```

5.3 Suffix_Array

```

1 void count_sort(int pos[], int n, vector<int>
2     > &rank){
3     int cnt[n] = {};
4     for(int i = 0; i < n; i++) cnt[rank[pos[i]]]++;
5     for(int i = 1; i < n; i++) cnt[i] += cnt[i-1];
6     int ans[n];
7     for(int i = n-1; ~i; i--){
8         ans[cnt[rank[pos[i]]]-1] = pos[i];
9         --cnt[rank[pos[i]]];
10    }
11    for(int i = 0; i < n; i++) pos[i] = ans[i];
12 }
13 // 只需要在求完Suffix Array之後順便求LCP就好了
14
15 void get_suffix(string &s, int pos[], vector
16     <int> &lcp){
17     s += '$';
18     int n = s.size();
19     vector<int> rank(n);
20     //k = 0
21     iota(pos, pos+n, 0);
22     sort(pos, pos+n, [&](int a, int b){ return s
23         [a] < s[b]; });
24     for(int i = 0; i < n; i++){
25         if(i==0){
26             rank[pos[i]] = 0;

```

```

23     }else{
24         rank[pos[i]] = rank[pos[i-1]] + (s[pos
25             [i]]!=s[pos[i-1]]);
26     }
27     //k > 0
28     vector<int> new_rank(n);
29     for(int k = 0; (1<<k) < n; k++){
30         for(int i = 0; i < n; i++){
31             pos[i] = (pos[i] - (1<<k)%n + n) % n;
32             count_sort(pos, n, rank);
33             new_rank[pos[0]] = 0;
34             for(int i = 1; i < n; i++){
35                 pair<int,int> prev = {rank[pos[i-1]],
36                     rank[(pos[i-1]+(1<<k))%n]};
37                 pair<int,int> now = {rank[pos[i]],
38                     rank[(pos[i]+(1<<k))%n]};
39                 new_rank[pos[i]] = new_rank[pos[i-1]]
40                     + (prev!=now);
41             }
42             rank = new_rank;
43         }
44     }
45     //求LCP Array
46     int k = 0;
47     for(int i = 0; i < n; i++){
48         int pi = rank[i];
49         int j = pos[pi-1];
50         while(i+k < n && j+k < n && s[i+k]==s[j+k])
51             k++;
52         lcp[pi] = k;
53         k = max(0, k-1);
54     }
55 }

```

5.4 ZValue

```

1 vector<int> z(const string& s)
2 {
3     int n = s.size();
4     vector<int> z(n);
5     int l = 0, r = 0;
6     for (int i = 0; i < n; i++) {
7         z[i] = max(min(z[i-l], r-i), 0);
8         while (i+z[i] < n && s[z[i]]==s[i+z[i]])
9             l = i, r = i+z[i], z[i]++;
10    }
11    return z;
12 }

```

6 Sublime-Text

6.1 SublimeBuild

```

1 //This is the build file script for sublime-
2 text, rename it as XXX.sublime_build

```

```

3 {
4     "cmd": ["g++", "-std=c++17", "$file", "-o",
5         "${file_path}/${file_base_name}",
6         "file_regex": "\\^(\\.\\.\\.)*:([0-9]+):?([0-9]+)
7         ??:? (.*?)$",
8         "working_dir": "${file_path}",
9         "selector": "source.c, source.c++, source.
10         cxx, source.cpp",
11         "variants":
12         [
13             {
14                 "name": "Run",
15                 "cmd": ["bash", "-c", "g++ -std=c++17 '$file
16                     ' -I . -o '${file_path}/${
17                     file_base_name}' && gnome-terminal --
18                     bash -c '\\ulimit -s unlimited; '${
19                     file_path}/${file_base_name}'; read;exit
20                     ; exec bash\""]
21             }
22         ]
23     }
24 }

```