

1 Data Structure

1.1 Segment Tree

```

1 struct SegT{
2     int d[4*N];
3     int lazy_tag[4*N];
4     int combine(int a, int b){
5         return a+b;
6     }
7     void build(int a[], int ind = 1, int l =
8         0, int r = N-1){
9         if(l==r){
10             d[ind]=a[l];
11         }else{
12             int mid = (l+r)/2;
13             build(a,ind<<1,l,mid);
14             build(a,ind<<1|1,mid+1,r);
15             d[ind] = combine(d[ind<<1],d[ind
16                 <<1|1]);
17         }
18     }
19     void modify(int pos, int val, int ind = 1,
20         int l = 0, int r = N-1){
21         if(l==r){
22             d[ind] = val;
23         }else{
24             int mid = (l+r)/2;
25             if(pos<=mid) modify(pos,val,ind<<1,l,
26                 mid);
27             else modify(pos,val,ind<<1|1,mid+1,r);
28             d[ind] = combine(d[ind<<1],d[ind
29                 <<1|1]);
30         }
31     }
32     void range_modify(int ml, int mr, int val,
33         int ind = 1, int l = 0, int r = N-1){
34         if(ml>r||mr<l) return;
35         if(ml<=l&&mr>=r){
36             lazy_tag[ind] += val;
37             d[ind] += (r-l+1)*val;
38             return;
39         }
40         int mid = (l+r)/2;
41         range_modify(ml,mr,val,ind<<1,l,mid);
42         range_modify(ml,mr,val,ind<<1|1,mid+1,r)
43         ;
44         d[ind] = combine(d[ind<<1],d[ind<<1|1]);
45     }
46     void apply(int ind, int val, int l, int r)
47     {
48         if(ind>=0&&ind<4*N){
49             d[ind] += (r-l+1)*val;
50             lazy_tag[ind] += val;
51         }
52     }
53     int query(int ql, int qr, int ind = 1, int
54         l = 0, int r = N-1){
55         if(ql>r||qr<l) return 0;
56         if(ql<=l&&qr>=r) return d[ind];
57         int mid = (l+r)/2;
58         if(lazy_tag[ind]){
59             apply(ind<<1, lazy_tag[ind], l, mid);
60             apply(ind<<1|1, lazy_tag[ind], mid+1,

```

```

51         apply(ind<<1|1, lazy_tag[ind], mid+1,
52             r);
53         d[ind] = combine(d[ind<<1],d[ind
54             <<1|1]);
55         lazy_tag[ind] = 0;
56     }
57     return combine(query(ql,qr,ind<<1,l,mid)
58         ,query(ql,qr,ind<<1|1,mid+1,r));
59 }

```

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     }else{
25         b->l = merge(a,b->l);
26         b->pull();
27     }
28     return b;
29 }
30 void split(Treap *t, Treap *&a, Treap *&b,
31     int k){
32     if(t==nullptr){
33         a = b = nullptr;
34         return;
35     }
36     if(sz(t->l) < k){
37         a = t;
38         split(t->r,a->r,b,k-sz(t->l)-1);
39         a->pull();
40     }else{
41         b = t;
42         split(t->l,a,b->l,k);
43         b->pull();
44     }
45 }
46 }

```

2 Graphs

2.1 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 [ww,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 }

```

3 Number Theory

3.1 FFT

```

1 typedef complex<double> cp;
2
3 const double pi = acos(-1);
4 const int NN = 131072;
5
6 struct FastFourierTransform{
7     /*
8      * Iterative Fast Fourier Transform
9      *
10     How this works? Look at this
11
12     0th recursion 0(000) 1(001) 2(010)
13                   3(011) 4(100) 5(101) 6(110)
14                   7(111)
15
16     1th recursion 0(000) 2(010) 4(100)
17                   6(110) | 1(011) 3(011) 5(101)
18                   7(111)
19
20     2th recursion 0(000) 4(100) | 2(010)
21                   6(110) | 1(011) 5(101) | 3(011)
22                   7(111)
23
24     3th recursion 0(000) | 4(100) | 2(010) |
25                   6(110) | 1(011) | 5(101) | 3(011)
26                   7(111)
27
28     All the bits are reversed => We can save
29     the reverse of the numbers in an
30     array!
31
32 */
33     int n, rev[NN];
34     cp omega[NN], iomega[NN];
35     void init(int n){

```

```

22     n = n_;
23     for(int i = 0; i < n; i++){
24         //Calculate the nth roots of unity
25         omega[i] = cp(cos(2*pi*i/n_), sin(2*pi*
26             i/n_));
27         iomega[i] = conj(omega[i]);
28     }
29     int k = __lg(n_);
30     for(int i = 0; i < n; i++){
31         int t = 0;
32         for(int j = 0; j < k; j++){
33             if(i & (1<<j)) t |= (1<<(k-j-1));
34         }
35         rev[i] = t;
36     }
37 }
38 void transform(vector<cp> &a, cp* xomega){
39     for(int i = 0; i < n; i++){
40         if(i < rev[i]) swap(a[i],a[rev[i]]);
41     }
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 void fft(vector<cp> &a){ transform(a,omega
54     ); }
55 void ifft(vector<cp> &a){ transform(a,
56     iomega); for(int i = 0; i < n; i++) a[i]
57     /= n; }
58 } FFT;

```

3.2 NTT

```

1 const int N = 5e5+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 struct NTT{
13     int n, inv, rev[N];
14     int omega[N], iomega[N];
15     void init(int n){
16         n = n_;
17         inv = fastpow(n,MOD-2);
18         int k = __lg(n);
19         int x = fastpow(G, (MOD-1)/n);
20         omega[0] = 1;
21         for(int i = 1; i < n; i++){

```

```
23     omega[i] = omega[i-1] * x % MOD;
24     iomega[n-1] = fastpow(omega[n-1],MOD-2);
25     for(int i = n-2; i >= 0; i--){
26         iomega[i] = iomega[i+1] * x % MOD;
27     }
28     for(int i = 0; i < n; i++){
29         int t = 0;
30         for(int j = 0; j < k; j++){
31             if(i&(1<<j)) t |= (1<<k-j-1);
32             rev[i] = t;
33         }
34     }
35 void transform(vector<int> &a, int *xomega)
36 {
37     for(int i = 0; i < n; i++){
38         if(i < rev[i]) swap(a[i],a[rev[i]]);
39     }
40     for(int len = 2; len <= n; len <<= 1){
41         int mid = len>>1;
42         int r = n/len;
43         for(int j = 0; j < n; j += len){
44             for(int i = 0; i < mid; i++){
45                 int tmp = xomega[r*i] * a[j+mid+i]
46                     % MOD;
47                 a[j+mid+i] = (a[j+i] - tmp + MOD)
48                     % MOD;
49                 a[j+i] = (a[j+i]+tmp)%MOD;
50             }
51         }
52     }
53 }
54 void dft(vector<int> &a){transform(a,omega);}
55 void idft(vector<int> &a){transform(a,
56     iomega); for(int i = 0; i < n; i++) a[i]
57     = a[i]*inv %MOD;}
58 } NTT;
```

KEEP ON THE HARD WORK!

Contents

1 Data Structure

1

2 Graphs

3 Number Theory

1

1.1	Segment Tree	1
1.2	Treap	1
2.1	dijkstra	1

3.1	FFT	1
3.2	NTT	1