1 Data Structure

1.1 Segment Tree

```cpp
struct SegT{
  int d[4*N];
  int lazy_tag[4*N];
  int combine(int a, int b){
    return a+b;
  }
  void build(int a[], int ind = 1, int l = 0, int r = N-1){
    if(l==r){
      d[ind]=a[l];
    }else{
      int mid = (l+r)/2;
      build(a,ind<<1,l,mid);
      build(a,ind<<1|1,mid+1,r);
      d[ind] = combine(d[ind<<1],d[ind<<1|1]);
    }
  }
  void modify(int pos, int val, int ind = 1, int l = 0, int r = N-1){
    if(l==r){
      d[ind] = val;
    }else{
      int mid = (l+r)/2;
      if(pos<=mid) modify(pos,val,ind<<1,l,mid);
      else modify(pos,val,ind<<1|1,mid+1,r);
      d[ind] = combine(d[ind<<1],d[ind<<1|1]);
    }
  }
  void range_modify(int ml, int mr, int val, int ind = 1, int l = 0, int r = N-1){
    if(ml>r||mr<l) return;
    if(ml<=l&&mr>=r){
      lazy_tag[ind] += val;
      d[ind] += (r-l+1)*val;
      return;
    }
    int mid = (l+r)/2;
    range_modify(ml,mr,val,ind<<1,l,mid);
    range_modify(ml,mr,val,ind<<1|1,mid+1,r);
    d[ind] = combine(d[ind<<1],d[ind<<1|1]);
  }
  void apply(int ind, int val, int l, int r){
    if(ind>=0&&ind<4*N){
      d[ind] += (r-l+1)*val;
      lazy_tag[ind] += val;
    }
  }
  int query(int ql, int qr, int ind = 1, int l = 0, int r = N-1){
    if(ql>r||qr<l) return 0;
    if(ql<=l&&qr>=r) return d[ind];
    int mid = (l+r)/2;
    if(lazy_tag[ind]){
      apply(ind<<1, lazy_tag[ind], l, mid);
      apply(ind<<1|1, lazy_tag[ind], mid+1, r);
      d[ind] = combine(d[ind<<1],d[ind<<1|1]);
      lazy_tag[ind] = 0;
    }
    return combine(query(ql,qr,ind<<1,l,mid),query(ql,qr,ind<<1|1,mid+1,r));
  }
};
```

1.2 Treap

```cpp
struct Treap{
  Treap *l, *r;
  int val, size, sum;
  Treap(int v): l(nullptr), r(nullptr), val(v), size(1), sum(v){}
  void pull();
};

void Treap::pull(){
  size = 1, sum = val;
  if(l!=nullptr) size += l->size, sum += l->sum;
  if(r!=nullptr) size += r->size, sum += r->sum;
}

int sz(Treap *t){
  return (t==nullptr ? 0 : t->size);
}

Treap *merge(Treap *a, Treap *b){
  if(a==nullptr) return b;
  if(b==nullptr) return a;
  if(rand()%(a->size+b->size) <a->size){
    a->r = merge(a->r,b);
    a->pull();
    return a;
  }else{
    b->l = merge(a,b->l);
    b->pull();
    return b;
  }
}

void split(Treap *t, Treap *&a, Treap *&b, int k){
  if(t==nullptr){
    a = b = nullptr;
    return;
  }
  if(sz(t->l) < k){
    a = t;
    split(t->r,a->r,b,k-sz(t->l)-1);
    a->pull();
  }else{
    b = t;
    split(t->l,a,b->l,k);
    b->pull();
  }
}
```

2 Graphs

2.1 dijkstra

```cpp
priority_queue<pair<int,int>,vector<pair<int,int>>, greater<pair<int,int>>> pq;
pq.push({0,s});
dis[s] = 0;
inq[s] = 1;
while(!pq.empty()){
  auto [ww,u] = pq.top(); pq.pop();
  inq[u] = 0;
  for(auto [v,w] : adj[u]){
    if(dis[v] > dis[u]+ w){
      dis[v] = dis[u]+w;
      if(!inq[v]){
        pq.push({dis[v],v});
        inq[v] = 1;
      }
    }
  }
}
```

3 Number Theory

3.1 FFT

```cpp
typedef complex<double> cp;

const double pi = acos(-1);
const int NN = 131072;

struct FastFourierTransform{
  /*
    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;
```

3.2 NTT

```cpp
const int N = 5e5+5, MOD = 998244353, G = 3;

int fastpow(int n, int p){
  int res = 1;
  while(p){
    if(p&1) res = res * n % MOD;
    n = n * n % MOD;
    p >>= 1;
  }
  return res;
}

struct NTT{
  int n, inv, rev[N];
  int omega[N], iomega[N];
  void init(int n_){
    n = n_;
    inv = fastpow(n,MOD-2);
    int k = __lg(n);
    int x = fastpow(G, (MOD-1)/n);
    omega[0] = 1;
    for(int i = 1;i < n;i++)
```

```cpp
      omega[i] = omega[i-1] * x % MOD;
    iomega[n-1] = fastpow(omega[n-1],MOD-2);
    for(int i = n-2; i >= 0; i--)
      iomega[i] = iomega[i+1] * x % MOD;
    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<int> &a, int *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++){
          int tmp = xomega[r*i] * a[j+mid+i] % MOD;
          a[j+mid+i] = (a[j+i] - tmp + MOD) % MOD;
          a[j+i] = (a[j+i]+tmp)%MOD;
        }
      }
    }
  }
  void dft(vector<int> &a){transform(a,omega);}
  void idft(vector<int> &a){transform(a, iomega); for(int i = 0;i < n;i++) a[i] = a[i]*inv %MOD;}
} NTT;
```

Keep On The Hard Work!

Contents