1 Data Structure

1.1 Segment Tree

```cpp
struct SegT{
  const int MAXN = 1e5+5;
  int tr[MAXN*4], arr[MAXN], tag[MAXN*4];

  int combine(int a, int b){
      return max(a,b);
  }

  void build(int idx, int l, int r){
      if(l==r){
          tr[idx] = arr[l];
      }else{
          int m = (l+r)/2;
          build(idx*2,l,m);
          build(idx*2+1,m+1,r);
          tr[idx] = combine(tr[idx*2],tr[idx*2+1]);
      }
  }

  void push(int idx){
      if(tag[idx]){
          tr[idx<<1] = max(tr[idx<<1], tag[idx]);
          tr[idx<<1|1] = max(tr[idx<<1|1], tag[idx]);
          tag[idx<<1] = max(tag[idx<<1], tag[idx]);
          tag[idx<<1|1] = max(tag[idx<<1|1], tag[idx]);
          tag[idx] = 0;
      }
  }

  void modify(int ql, int qr, int val, int idx, int l, int r){
      if(l!=r) push(idx); //當節點並非葉節點時，下推標記
      if(ql <= l && r <= qr){
          tr[idx] = max(tr[idx],val);
          tag[idx] = max(tag[idx],val);
          return;
      }
      int m = (l+r)/2;
      if(qr > m) modify(ql, qr, val, idx*2+1, m+1, r);
      if(ql <= m) modify(ql, qr, val, idx*2, l, m);
      tr[idx] = combine(tr[idx<<1],tr[idx<<1|1]);
  }

  int query(int ql, int qr, int idx, int l, int r){
      if(l!=r) push(idx);
      if(ql <= l && r <= qr){
          return tr[idx];
      }
      int m = (l+r)/2;
      if(ql > m){
          return query(ql, qr, idx*2+1, m+1, r);
      }
      if(qr <= m){
          return query(ql, qr, idx*2, l, m);
      }
      return combine(query(ql, qr, idx*2, l, m), query(ql, qr, idx*2+1, m+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 = 1e5+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 = 1;
        while(n < n_) n<<=1;
        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++)
            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(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(int *a){transform(a,omega);}
    void idft(int *a){transform(a,iomega); for(int i = 0;i < n;i++) a[i] = a[i]*inv %MOD;}

    int tmp[8][N];

    void copy_(int *a, int *b, int m){
        for(int i = 0;i < m;i++)
            a[i] = b[i];
        for(int i = m;i < n;i++)
            a[i] = 0;
    }

    void copy(int *a, int *b, int m){
        for(int i = 0;i < m;i++)
            a[i] = b[i];
    }

    //B_{k+1} = B_k(2-AB_k) (mod MOD)
    void inverse(int *a, int *b, int m){
        //Uses tmp[0], tmp[1]
        if(m==1){
            b[0] = fastpow(a[0],MOD-2);
            return;
        }
        inverse(a,b,m>>1);
        init(m<<1);
        copy_(tmp[0],a,m); copy_(tmp[1],b,m>>1);
        dft(tmp[0]); dft(tmp[1]);
        for(int i = 0;i < n;i++) tmp[0][i] = tmp[1][i]*(2-tmp[0][i]*tmp[1][i]%MOD+MOD)%MOD;
        idft(tmp[0]);
        copy(b,tmp[0],m);
    }


    //Q_{k+1} = pow(2,MOD-2)(Q_k + P*pow(Q_k,MOD-2)) (mod MOD)
    void sqrt(int *a, int *b, int m){
        //Uses tmp[2], tmp[3]
        if(m==1){
            b[0] = 1;
            return;
        }
        sqrt(a,b,m>>1);
        for(int i = m;i < m<<1;i++)
            b[i] = 0;
        inverse(b,tmp[2],m);
        init(m<<1);
        for(int i = m;i < m<<1;i++)
            b[i] = tmp[2][i] = 0;
        int inv2 = fastpow(2,MOD-2);
        copy_(tmp[3],a,m);
        dft(tmp[3]); dft(tmp[2]);
        for(int i = 0;i < n;i++)
            tmp[3][i] = tmp[3][i]*tmp[2][i]%MOD;
        idft(tmp[3]);
        for(int i = 0;i < m;i++)
            b[i] = (b[i]+tmp[3][i])%MOD*inv2%MOD;
    }

    void derivative(int *a, int *b, int m){
        for(int i = 1;i < m;i++) b[i-1] = a[i]*i%MOD;
        b[m-1] = 0;
    }

    void integral(int *a, int *b, int m){
        for(int i = m-1;i;i--) b[i] = a[i-1]*fastpow(i,MOD-2)%MOD;
        b[0] = 0;
    }

    void ln(int *a, int *b, int m){
        //Uses tmp[4], tmp[5]
        inverse(a,b,m);
        derivative(a,tmp[5],m);

        init(m<<1);
        copy_(tmp[4],b,m), copy_(tmp[5],tmp[5],m);
        dft(tmp[4]), dft(tmp[5]);
        for(int i = 0;i < m<<1;i++) tmp[4][i] = tmp[4][i]*tmp[5][i]%MOD;
        idft(tmp[4]);
        integral(tmp[4],b,m);
    }

    void exp(int *a, int *b, int m){
        //Uses tmp[6], tmp[7]
        b[0] = 1;
        for(int i = 4,j = 2;j <= m;j = i, i <<=1){
            ln(b,tmp[6],j);
            tmp[6][0] = (a[0]+1-tmp[6][0]+MOD)%MOD;
            for(int k = 1; k < j;k++) tmp[6][k] = (a[k]-tmp[6][k]+MOD)%MOD;

            fill(tmp[6]+j,tmp[6]+i,0);
            dft(b), dft(tmp[6]);
            for(int k = 0; k < i;k++) b[k] = b[k]*tmp[6][k]%MOD;
            idft(b);
            fill(b+j,b+i,0);
        }
    }

} NTT;
```

Keep On The Hard Work!

Contents