Team notebook

Nicolas Alba

July 24, 2024

Contents

1 BitTricks

```
y = x & (x-1) // Turn off rightmost 1bit
y = x & (-x)  // Isolate rightmost 1bit
y = x | (x-1) // Right propagate rightmost 1bit(fill in 1s)
y = x | (x+1) // Turn on rightmost 0bit
y = ~x & (x+1) // Isolate rightmost 0bit
// If x is of long type, use __builtin_popcountl(x)
// If x is of long long type, use __builtin_popcountll(x)
// 1. Counts the number of ones(set bits) in an integer.
__builtin_popcount(x)
// 2. Checks the Parity of a number. Returns true(1) if the
// number has odd number of set bits, else it returns
// false(0) for even number of set bits.
 __builtin_parity(x)
// 3. Counts the leading number of zeros of the integer.
__builtin_clz(x)
// 4. Counts the trailing number of zeros of the integer.
__builtin_ctz(x)
// 5. Returns 1 + the index of the least significant 1-bit.
__builtin_ffs(x) // If x == 0, returns 0.
// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for(int s=0;s=s-m&m;) // Increasing order
```

2 Catalan

```
/*Catalan, counts the number of ways of:
( A ) B, where |A|+|B| = N, for N+1
*/

const int MOD = 1e9 + 7;
ll mul(ll x, ll y) { return (x*y)%MOD; }
ll pot(ll x, ll y) {
    if(y==0) return 1;
    ll ans = pot(x,y/2);
    ans = mul(ans,ans);
```

```
    if (y&1)ans=mul(ans,x);
    return ans;
}
ll inv(ll x) { return pot(x, MOD-2); }

// mxN it the double of the max input 'n'
const int mxN = 2e6 + 10;
vl fact(mxN,1);
void init() {
    for (int i =1;i<=mxN;i++) {
        fact[i] = mul(fact[i-1],i);
    }
}

ll catalan(ll n) {
    if (n<0) return 0;
    ll up = fact[2*n];
    ll down = mul(fact[n],fact[n+1]);
    return mul(up,inv(down));
}
```

3 CatalanConvolution

```
/*
Return Catalan Convolution.

Convolution for k=3
((( A ) B ) C ) D

Where A + B + C + D = N, for N + 1
*/

const int MOD = 1e9 + 7;
ll mul(ll x, ll y) { return (x*y)%MOD; }
ll pot(ll x, ll y) {
    if(y==0) return 1;
    ll ans = pot(x,y/2);
    ans = mul(ans,ans);
    if (y&1)ans=mul(ans,x);
    return ans;
}
ll inv(ll x) { return pot(x, MOD-2); }

// mxN it the double of the max input N, plus max K
const int mxN = 2e6 + 1e6 + 10;
vl fact(mxN,1);
ll cnk(ll n, ll k) {
    if (k < 0 || n < k) return 0;
    ll nOverK = mul(fact[n],inv(fact[k]));
    return mul(nOverK,inv(fact[n-k]));
}

void init() {
    for (int i =1;i<=mxN;i++) {
        fact[i] = mul(fact[i-1],i);
    }
}

// for parethesis example
// number of n+k pairs having k open parethesis at beginning

// (cnk(2n+k,n)*(k+1))/(n+k+1)
ll catalanCov(ll n, ll k) {
    ll up = mul(cnk(2*n+k,n),(k+1)%MOD);
    ll down = (n+k+1)%MOD;
    return mul(up,inv(down));
}

/*
6
(()

ans: 2
*/
// size, and prefix
ll countParenthesisWithPrefix(ll n, string &p) {
    if (n&1) return 0;
    ll k = 0;
    for (auto c : p) {
        if (c=='(') k++;
        else k--;
        if (k<0) return 0;
    }
    n=(n-(ll)p.size()-k)/2;
    return catalanCov(n,k);
}
```

4 ClosestPairOfPoints

```
// It seems O(n log n), not sure but it worked for 50000

#define x first
#define y second
long long dist2(pair<int, int> a, pair<int, int> b) {
	return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y) * (a.y
	    - b.y);
}
pair<int, int> closest_pair(vector<pair<int, int>> a) {
	int n = a.size();
	assert(n >= 2);
	vector<pair<pair<int, int>, int>> p(n);
	for (int i = 0; i < n; i++) p[i] = {a[i], i};
	sort(p.begin(), p.end());
	int l = 0, r = 2;
	long long ans = dist2(p[0].x, p[1].x);
	pair<int, int> ret = {p[0].y, p[1].y};
	while (r < n) {
		while (l < r && 1LL * (p[r].x.x - p[l].x.x) * (p[r].x.x -
		    p[l].x.x) >= ans) l++;
		for (int i = l; i < r; i++) {
			long long nw = dist2(p[i].x, p[r].x);
			if (nw < ans) {
				ans = nw;
				ret = {p[i].y, p[r].y};
			}
		}
		r++;
	}
	return ret;
}

// Tested: https://vjudge.net/solution/52922194/ccPUXODAMWTzpzCEvXbV
void test_case() {
    ll n;
    cin >> n;
    vector<pair<int,int>> points(n);
    for (int i = 0;i<n;i++) cin >> points[i].x >> points[i].y;
    auto ans = closest_pair(points);
    cout << fixed << setprecision(6);
    if (ans.F > ans.S) swap(ans.F,ans.S);
    ld dist = sqrtl(dist2(points[ans.F],points[ans.S]));
    cout << ans.F << " " << ans.S << " " << dist << endl;
}
```

5 ConvexHullTrick

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m, and
     query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */

 // For minimum you can multiply by -1 'k' and 'm' when adding, and the
     answer when querying.
 // Tested in https://atcoder.jp/contests/dp/submissions/55836691
#pragma once

struct Line {
	mutable ll k, m, p;
	bool operator<(const Line& o) const { return k < o.k; }
	bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
	// (for doubles, use inf = 1/.0, div(a,b) = a/b)
	static const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		return a / b - ((a ^ b) < 0 && a % b); }
	bool isect(iterator x, iterator y) {
		if (y == end()) return x->p = inf, 0;
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}
	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
```

```
            if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
            while ((y = x) != begin() && (--x)->p >= y->p)
                isect(x, erase(y));
        }
        ll query(ll x) {
            assert(!empty());
            auto l = *lower_bound(x);
            return l.k * x + l.m;
        }
};
```

6 DP-Mask-Over-Submasks

```
// DP of submask over submasks
// O(3^n)

// j&(-j); get a '1' bit of j
// for (int j=i;j;j = (j-1)&i){...} j is submask of 'i' the mask

ll dp[1<<18]; // answer of mask
ll cst[1<<18]; // cost of use a submask
ll a[18][18]; // elements
ll pos[1<<18]; // trick to get fast the pos
void test_case() {
    ll n;
    cin >> n;
    for (int i =0;i<n;i++) {
        for (int j =0;j<n;j++) {
            cin >> a[i][j];
        }
    }
    for (int i =0;i<n;i++) {
        pos[1<<i] = i;
    }
    for (int i = 0;i<(1<<n);i++) {
        ll j = i;
        vl idxs;
        while (j) {
            ll k = j&(-j);
            idxs.pb(pos[k]);
            j^=k;
        }
        for (int j = 0;j<idxs.size();j++) {
            for (int k = j+1;k<idxs.size();k++) {
                cst[i] += a[idxs[j]][idxs[k]];
            }
        }
        dp[i] = cst[i];
        for (int j=i;j;j = (j-1)&i) {
            dp[i] = max(dp[i],cst[j]+dp[i^j]);
        }
    }
    cout << dp[(1<<n)-1] << "\n";
}
```

7 Optimized-Polard-Rho

```
#define fore(i, b, e) for(int i = b; i < e; i++)
ll gcd(ll a, ll b){return a?gcd(b%a,a):b;}
ll mulmod(ll a, ll b, ll m) {
  ll r=a*b-(ll)((long double)a*b/m+.5)*m;
  return r<0?r+m:r;
}
ll expmod(ll b, ll e, ll m){
  if(!e)return 1;
  ll q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return e&1?mulmod(b,q,m):q;
}
bool is_prime_prob(ll n, int a){
  if(n==a)return true;
  ll s=0,d=n-1;
  while(d%2==0)s++,d/=2;
  ll x=expmod(a,d,n);
  if((x==1)||(x+1==n))return true;
  fore(_,0,s-1){
    x=mulmod(x,x,n);
    if(x==1)return false;
    if(x+1==n)return true;
  }
  return false;
}
bool rabin(ll n){ // true iff n is prime
  if(n==1)return false;
  int ar[]={2,3,5,7,11,13,17,19,23};
```

```
  fore(i,0,9)if(!is_prime_prob(n,ar[i]))return false;
  return true;
}
// optimized version: replace rho and fact with the following:
const int MAXP=1e6+1; // sieve size
int sv[MAXP]; // sieve
ll add(ll a, ll b, ll m){return (a+=b)<m?a:a-m;}
ll rho(ll n){
  static ll s[MAXP];
  while(1){
    ll x=rand()%n,y=x,c=rand()%n;
    ll *px=s,*py=s,v=0,p=1;
    while(1){
      *py++=y=add(mulmod(y,y,n),c,n);
      *py++=y=add(mulmod(y,y,n),c,n);
      if((x=*px++)==y)break;
      ll t=p;
      p=mulmod(p,abs(y-x),n);
      if(!p)return gcd(t,n);
      if(++v==26){
        if((p=gcd(p,n))>1&&p<n)return p;
        v=0;
      }
    }
    if(v&&(p=gcd(p,n))>1&&p<n)return p;
  }
}
void init_sv(){
  fore(i,2,MAXP)if(!sv[i])for(ll j=i;j<MAXP;j+=i)sv[j]=i;
}
void fact(ll n, map<ll,int>& f){ // call init_sv first!!!
  for(auto&& p:f){
    while(n%p.F==0){
      p.S++; n/=p.F;
    }
  }
  if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
  else if(rabin(n))f[n]++;
  else {ll q=rho(n);fact(q,f);fact(n/q,f);}
}
```

8 PointInConvexPolygon

```
ll IN = 0;
ll ON = 1;
ll OUT = 2;
vector<string> ANS = {"IN", "ON", "OUT"};

#define pt pair<ll,ll>
#define x first
#define y second

pt sub(pt a, pt b) { return {a.x - b.x, a.y - b.y}; }
ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x = 180 -> sin = 0
ll orient(pt a, pt b, pt c) { return cross(sub(b,a),sub(c,a)); }//
    clockwise = -

// poly is in clock wise order
ll insidePoly(vector<pt> &poly, pt query) {
    ll n = poly.size();
    ll left = 1;
    ll right = n - 2;
    ll ans = -1;
    if (!(orient(poly[0], poly[1], query) <= 0
        && orient(poly[0], poly[n-1], query) >= 0)) {
        return OUT;
    }
    while (left <= right) {
        ll mid = (left + right) / 2;
        if (orient(poly[0], poly[mid], query) <= 0) {
            left = mid + 1;
            ans = mid;
        } else {
            right = mid - 1;
        }
    }
    left = ans;
    right = ans + 1;
    if (orient(poly[left], query, poly[right]) < 0) {
        return OUT;
    }
    if (orient(poly[left], poly[right], query) == 0
       || (left == 1 && orient(poly[0], poly[left], query) == 0)
       || (right == n-1 && orient(poly[0], poly[right], query) == 0)) {
        return ON;
    }
    return IN;
```

```
}
```

9 PolygonDiameter

```
// Given a set of points, it returns
// the diameter (the biggest distance between 2 points)
// tested: https://open.kattis.com/submissions/13937489
const double eps = 1e-9;
int sign(double x) { return (x > eps) - (x < -eps); }

struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT operator - (const PT &a) const { return PT(x - a.x, y - a.y); }
    bool operator < (PT a) const { return sign(a.x - x) == 0 ? y < a.y :
        x < a.x; }
    bool operator == (PT a) const { return sign(a.x - x) == 0 && sign(a.y
        - y) == 0; }

};
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b - a, c -
    a)); }

double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j]) >= 0) {
               ans = max(ans, dist2(p[i], p[j]));
               j = (j + 1) % n;
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
    }
    return sqrt(ans);
}
vector<PT> convex_hull(vector<PT> &p) {
        if (p.size() <= 1) return p;
        vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() - 2], up.back(),
            p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() - 2], dn.back(),
            p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}

void test_case() {
    ll n;
    cin >>n;
    vector<PT> p(n);
    for (int i = 0;i<n;i++) cin >> p[i].x >> p[i].y;
    p = convex_hull(p);
    cout << fixed<<setprecision(10) << diameter(p) << "\n";
}
```

10 fast-hadamard-transform

```
// like polynomial multiplication, but XORing exponents
```

```
// instead of adding them (also ANDing, ORing)
const int MAXN=1<<18;

#define fore(i,l,r) for(int i=int(l);i<int(r);++i)
#define SZ(x) ((int)(x).size())

ll c1[MAXN+9],c2[MAXN+9];//MAXN must be power of 2!
void fht(ll* p, int n, bool inv){
    for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)fore(j,0,l){
        ll u=p[i+j],v=p[i+l+j];
        // if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
        // else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
        //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
        //else p[i+j]=-u+v,p[i+l+j]=u;
        if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
        else p[i+j]=v,p[i+l+j]=u-v;
    }
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
    int n=1<<(32-__builtin_clz(max(SZ(p1),SZ(p2))-1));
    fore(i,0,n)c1[i]=0,c2[i]=0;
    fore(i,0,SZ(p1))c1[i]=p1[i];
    fore(i,0,SZ(p2))c2[i]=p2[i];
    fht(c1,n,false);fht(c2,n,false);
    fore(i,0,n)c1[i]*=c2[i];
    fht(c1,n,true);
    return vector<ll>(c1,c1+n);
}

// maxime the OR of a pair of given nums and count
// how many pairs can get that maximum OR
// tested: https://csacademy.com/contest/archive/task/maxor
void test_case() {
    ll n; cin >> n;
    vl a(MAXN),b(MAXN);
    for (int i =0;i<n;i++) {
        ll x;
        cin >> x;
        a[x]++;
        b[x]++;
    }
    vl c = multiply(a,b);
    pair<ll,ll> best = {0, c[0]};
    for (int i = 0;i<MAXN;i++) {
        if (c[i]) best = {i,(c[i]-a[i])/2};
    }
    cout <<best.F << " " << best.S << endl;
}
```

11 general$_{l}azy_{t}ree$

```
struct Node {
    ll mn;
    ll size = 1;

    Node(ll mn):mn(mn) {
    }
};

struct Func {
    ll a = 0;
};

Node e() { // op(x, e()) = x
    Node a(INT64_MAX);
    return a;
};

Func id() { // mapping(x, id()) = x
    Func l = {0};
    return l;
}

Node op(Node &a, Node &b) { // associative property
    Node c = e();
    c.size = a.size + b.size;
    c.mn = min(a.mn, b.mn);
    return c;
}

Node mapping(Node node, Func &lazy) {
    node.mn += lazy.a;
    return node;
}
```

```
Func composicion(Func &prev, Func &actual) {
    prev.a = prev.a + actual.a;
    return prev;
}

struct lazytree {
    int n;
    vector<Node> nodes;
    vector<Func> lazy;

    void init(int nn) {
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        ll m = size *2;
        nodes.assign(m, e());
        lazy.assign(m, id());
    }

    void push(int i, int sl, int sr) {
        nodes[i] = mapping(nodes[i], lazy[i]);
        if (sl != sr) {
            lazy[i * 2 + 1] = composicion(lazy[i*2+1],lazy[i]);
            lazy[i * 2 + 2] = composicion(lazy[i*2+2],lazy[i]);
        }
        lazy[i] = id();
    }

    void apply(int i, int sl, int sr, int l, int r, Func f) {
        push(i, sl, sr);
        if (l <= sl && sr <= r) {
            lazy[i] = f;
            push(i,sl,sr);
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
            apply(i * 2 + 1, sl, mid, l, r, f);
            apply(i * 2 + 2, mid + 1, sr, l, r, f);
            nodes[i] = op(nodes[i*2+1],nodes[i*2+2]);
        }
    }

    void apply(int l, int r, Func f) {
        assert(l <= r);
        assert(r < n);
        apply(0, 0, n - 1, l, r, f);
    }

    void update(int i, Node node) {
        assert(i < n);
        update(0, 0, n-1, i, node);
    }

    void update(int i, int sl, int sr, int pos, Node node) {
        if (sl <= pos && pos <= sr) {
            push(i,sl,sr);
            if (sl == sr) {
                nodes[i] = node;
            } else {
                int mid = (sl + sr) >> 1;
                update(i * 2 + 1, sl, mid, pos, node);
                update(i * 2 + 2, mid + 1, sr, pos, node);
                nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
            }
        }
    }

    Node query(int i, int sl, int sr, int l, int r) {
        push(i,sl,sr);
        if (l <= sl && sr <= r) {
            return nodes[i];
        } else if (sr < l || r < sl) {
            return e();
        } else {
            int mid = (sl + sr) >> 1;
            auto a = query(i * 2 + 1, sl, mid, l, r);
            auto b = query(i * 2 + 2, mid + 1, sr, l, r);
            return op(a,b);
        }
    }

    Node query(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return query(0, 0, n - 1, l, r);
    }
};
```