Contents

1 Settings

1.1 C++

```
#include <bits/stdc++.h>
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
#pragma GCC optimize("unroll-loops")
#define ll long long
#define eps 1e-7
#define all(x) ((x).begin()),((x).end())
#define usecppio ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
using namespace std;
using pii = pair<int, int>;
```

2 Data Structures

2.1 Segment Tree - Range Minimum

Range minimum, 0-based data.

```
struct Range_Minimum_Tree
{
    int n;
    vector<int> segtree;

    Range_Minimum_Tree(const vector<int> &data)
    {
        n = data.size();
        segtree.resize(4 * n);
        initialize(data, 0, n - 1, 1);
    }

    int initialize(const vector<int> &data, int l, int r, int node)
    {
        if (l == r)
            return segtree[node] = data[l];
        int mid = (l + r) / 2;
        int lmin = initialize(data, l, mid, node * 2);
        int rmin = initialize(data, mid + 1, r, node * 2 + 1);
        return segtree[node] = min(lmin, rmin);
    }
    int minq(int l, int r, int node, int nodeleft, int noderight)
    {
        if (r < nodeleft || noderight < l)
            return INT_MAX;
        if (l <= nodeleft && noderight <= r)
            return segtree[node];
        int mid = (nodeleft + noderight) / 2;
        return min(minq(l,r,node*2,nodeleft,mid),
        minq(l,r,node*2+1,mid+1,noderight));
    }
    void update(int left,int right,int node, int change_node ,int diff)
    {
        if (!(left <= change_node &&change_node <= right))
            return;
        if (left == right)
        {
            segtree[node] += diff;
        }
        if (left != right)
        {
            int mid = (left + right) / 2;
            update(left, mid, node * 2, change_node, diff);
            update(mid +1,right, node * 2 +1, change_node, diff);
            segtree[node] = min(segtree[node*2], segtree[node*2+1]);
        }
    }
    void update(int change_node, int diff)
    {
        update(0,n-1,1,change_node,diff);
    }
    void update_to(int change_node, int to)
    {
        int u = sumq(change_node, change_node);
        update(change_node, to-u);
    }
};
```

2.2 Segment Tree Lazy Propagation

Lazy Propagation (Segment constant addition), 0-based data.

```
struct SegTree
{
    int n;
```

```
vector <int> segtree;
vector <int> lazy;

SegTree()
{
    n = 0;
}

SegTree(vector <int> &data)
{
    n = data.size();
    segtree.resize(4*n);
    lazy.resize(4*n);
    init(data,1,0,n-1);
}

int init(vector <int> &data, int node, int l, int r)
{
    if (l==r)
    {
        segtree[node] = data[l];
        return segtree[node];
    }
    int mid = (l+r)/2;
    int ls = init(data,node*2,l,mid);
    int rs = init(data,node*2+1,mid+1,r);
    segtree[node] = (ls+rs);
    return segtree[node];
}

void propagation(int node, int nl, int nr)
{
    if (lazy[node]!=0)
    {
        segtree[node] += (lazy[node] * (nr-nl+1));
        if (nl != nr)
        {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }
}

void range_upd(int s, int e, int k)
{
    return range_upd(s,e,k,1,0,n-1);
}

void range_upd(int s, int e, int k, int node, int nl, int nr)
{
    propagation(node,nl,nr);

    if (nr < s || nl > e)
        return;
    if (s <= nl && nr <= e)
    {
        lazy[node] += k;
        propagation(node,nl,nr);
        return;
    }
    int mid = (nl + nr)/2;
    range_upd(s,e,k,node*2,nl,mid);
    range_upd(s,e,k,node*2+1,mid+1,nr);
    segtree[node] = segtree[node*2] + segtree[node*2+1];
    return;
}

int sum(int s, int e)
{
    return sum(s,e,1,0,n-1);
}

int sum(int s, int e, int node, int nl, int nr)
{
    propagation(node,nl,nr);
    if (nr < s || nl > e)
        return 0;
    if (s <= nl && nr <= e)
    {
        return segtree[node];
    }
    int mid = (nl+nr)/2;
```

```
        return (sum(s,e,node*2,nl,mid) + sum(s,e,node*2+1,mid+1,nr));
    }

};
```

2.3 2D Segment Tree

```
// Original ref http://www.secmem.org/blog/2019/11/15/2D-segment-tree/
// Slightly Modified
auto gif = [](int a, int b){return a+b;};
class SEG2D
{
public:
    int n;
    int m;
    vector <vector <int>> tree;
    SEG2D(int n = 0, int m = 0)
    {
        tree.resize(2*n);
        for (int i = 0; i<2*n; i++) tree[i].resize(2*m);
        this->n = n;
        this->m = m;
    }
    SEG2D(int n, int m, vector<vector<int>> &data)
    {
        tree.resize(2*n);
        for (int i = 0; i<2*n; i++) tree[i].resize(2*m);
        this->n = n;
        this->m = m;
        init(data);
    }
    void init(vector <vector <int>> & data)
    {
        n = data.size();
        m = data.front().size();
        tree = vector<vector<int>>(2*n, vector<int>(2*m, 0));
        for (int i = 0; i<n; i++)
            for (int j = 0; j<m; j++)
                tree[i+n][j+m] = data[i][j];
        for (int i = n; i<2*n; i++)
            for (int j = m-1; j>0; j--)
                tree[i][j] = gif(tree[i][j*2], tree[i][j*2+1]);
        for (int i = n-1; i>0; i--)
            for (int j = 1; j<2*m; j++)
                tree[i][j] = gif(tree[i*2][j], tree[i*2+1][j]);
    }
    void update(int x, int y, int val)
    {
        tree[x+n][y+m] = val;
        for(int i = y+m; i > 1; i /= 2)
            tree[x+n][i/2] = gif(tree[x+n][i] , tree[x+n][i^1]);
        for (int i = x+n; i>1; i/=2)
        {
            for (int j = y+m; j>=1; j/=2)
            {
                tree[i/2][j] = gif(tree[i][j] , tree[i^1][j]);
            }
        }
    }
    int query_1D(int x, int yl, int yr)
    {
        int res = 0;
        int u = yl+m, v = yr+m+1;
        for(; u<v; u/=2, v/=2)
        {
            if (u & 1)
                res = gif(res, tree[x][u++]);
            if (v & 1)
                res = gif(res, tree[x][--v]);
        }
        return res;
    }
    int query_2D(int xl, int xr, int yl, int yr)
    {
        int res = 0;
        int u = xl+n, v = xr+n+1;
        for(; u<v; u/=2, v/=2)
        {
            if (u & 1)
            {
                int k = query_1D(u++, yl, yr);
                res = gif(res, k);
            }
            if (v & 1)
            {
```

```
            int k = query_1D(--v, yl, yr);
            res = gif(res, k);
          }
        }
        return res;
    }
    void print()
    {
        for (int i = 0; i<2*n; i++)
        {
            for (int j = 0; j< 2*m; j++)
            {
                printf("%lld ",tree[i][j]);
            }
            printf("\n");
        }
    }
};
```

2.4 Fenwick Tree

```
struct Fenwick
{
    int n;
    int tree[MAXN];
    void init()
    {
        memset(tree,0,sizeof(tree));
    }
    int sum(int p)
    {
        int ret = 0;
        for (; p > 0; p -= p & -p)
            ret += tree[p];
        return ret;
    }
    void add (int p, int val)
    {
        for (; p <= n; p += p & -p)
            tree[p] += val;
    }
    void change (int p, int val)
    {
        int u = sum(p) - sum(p-1);
        add(p, val-u);
    }
};
```

2.5 Disjoint Set Union (Union - Find)

```
// Original Author : Ashishgup
struct Disjoint_Set_Union
{
    int parent[V], size[V];
    Disjoint_Set_Union(int N = V-1)
    {
        init(N);
    }
    void init(int N)
    {
        for(int i=1;i<=N;i++)
        {
            parent[i]=i;
            size[i]=1;
        }
    }
    int Find(int K)
    {
        while(K!=parent[K])
        {
            parent[K]=parent[parent[K]];
            K=parent[K];
        }
        return K;
    }
    int getSize(int K)
    {
        return size[Find(K)];
    }
    void unite(int x, int y)
    {
        int u=Find(x), v=Find(y);
        if(u==v)
            return;
        if(size[u]>size[v])
```

```
            swap(u, v);
        size[v]+=size[u];
        size[u] = 0;
        parent[u] = parent[v];
    }
} dsu;
```

3 Mathematics

3.1 Useful Mathematical Formula

- Catalan Number : Number of valid parantheses strings with n pairs

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

- Nim Game : Remember - XOR of all piles.
- Lucas Formula : $\binom{n}{m} \mod p = \prod \binom{n_i}{m_i} \mod p$
- Sum of divisors of n : About $n \log\log n$.
- 어떤 수열을 n개의 증가하는 수열로 덮을 수 있다 $\leftrightarrow$ longest decreasing subsequence 의 길이가 n보다 작거나 같다. (Dilworth's Theorem)

3.2 Binomial Coefficient

If faster method is needed : use modulo inverse

```
int binomial(int n, int k)
{
    for (int i = 0; i <= n; ++i)
    {
        for (int j = 0; j <= min(i, k); ++j)
        {
            if (j == 0 || j == i)
                binom[i][j] = 1;
            else
                binom[i][j] = binom[i-1][j-1] + binom[i-1][j];
        }
    }
    return binom[n][k];
}
```

3.3 Extended Euclidean Algorithm

(x, y) such that $ax + by = \gcd(a, b) = d$.

```
int Extended_Euclidean(int a, int b, int & x, int & y)
{
    if (a == 0)
    {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int d = Extended_Euclidean(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}
```

3.4 Fast Modulo Exponentiation

Calculating x^y `mod` p in $\mathcal{O}(\log y)$ time.

```
ll modpow(ll x, ll y, ll p)
{
    ll res = 1;
    x = x % p;
    while (y > 0)
    {
        if (y & 1)
            res = (res*x) % p;
        y = y>>1;
        x = (x*x) % p;
    }
    return res;
}
```

3.5 Modular Multiplicative Inverse

```
ll modinv(ll x, ll p)
{
    return modpow(x,p-2,p);
}
```

3.6 Miller-Rabin Primality Testing

Base values of a chosen so that results are tested to be correct up to 10^{14}.

```
bool MRwitness(ll n, ll s, ll d, ll a)
{
    ll x = modpow(a, d, n);
    ll y = -1;

    while (s)
    {
        y = (x * x) % n;
        if (y == 1 && x != 1 && x != n-1)
            return false;
        x = y;
        s--;
    }
    return (y==1);
}

bool Miller_Rabin(ll n)
{
    if (n<2)
        return false;
    if (n == 2 || n == 3 || n == 5 || n == 7 ||
     n == 11 || n == 13 || n == 17)
        return true;
    if (n%2 == 0 || n%3 == 0 || n%5 == 0)
        return false;
    ll d = (n-1) / 2;
    ll s = 1;
    while (d%2==0)
    {
        d /= 2;
        s++;
    }
    int candidate[7] = {2,3,5,7,11,13,17};
    bool result = true;
    for (auto i : candidate)
    {
        result = result & MRwitness(n,s,d,i);
        if (!result)
            break;
    }
    return result;
}
```

3.7 Pollard-Rho Factorization

```
ll PollardRho(ll n)
{
    srand (time(NULL));
    if (n==1)
        return n;
    if (n % 2 == 0)
        return 2;
    ll x = (rand()%(n-2))+2;
    ll y = x;
    ll c = (rand()%(n-1))+1;
    ll d = 1;
    while (d==1)
    {
        x = (modpow(x, 2, n) + c + n)%n;
        y = (modpow(y, 2, n) + c + n)%n;
        y = (modpow(y, 2, n) + c + n)%n;
        d = gcd(abs(x-y), n);
        if (d==n)
            return PollardRho(n);
    }
    return d;
}
```

3.8 Euler Totient

Calculating number of integers below n which is coprime with n.

```
ll euler_phi(ll n)
{
    ll p=2;
    ll ephi = n;
    while(p*p<=n)
    {
        if (n%p == 0)
            ephi = ephi/p * (p-1);
        while(n%p==0)
            n/=p;
        p++;
    }
    if (n!=1)
    {
```

```
        ephi /= n;
        ephi *= (n-1);
    }
    return ephi;
}
```

3.9 Fast Fourier Transform

Compute $A(x) * B(x)$ in $O(n \log n)$ time.
Convolution 빠르게 구하기

$$c_j = \sum_{i=0}^{j} a_i b_{j-i}$$

```
/* FFT Library : Originally Written by Myungwoo *
 * https://blog.myungwoo.kr/54                  *
 * Nonrecursive, Bit-Flipping Trick             *
 * Several Modifications                        */
#define sz(v) ((int)(v).size())
typedef complex<double> base;
typedef vector <int> vi;
typedef vector <base> vb;
const double PI = acos(-1);

void fft(vb &a, bool invert)
{
    int n = sz(a);
    for (int i = 1, j=0; i<n; i++)
    {
        int bit = n>>1;
        for (; j>=bit; bit>>=1)
        {
            j -= bit;
        }
        j += bit;
        if (i < j)
            swap(a[i],a[j]);
    }
    vector<base> root(n/2);
    double ang = 2*acos(-1)/n*(invert?-1:1);
    for(int i=0;i<n/2;i++)root[i]=base(cos(ang*i), sin(ang*i));
    for (int len = 2; len <= n; len <<= 1)
    {
        int step = n / len;
        for (int i = 0; i<n; i+= len)
        {
            for (int j = 0; j<len/2; j++)
            {
                base u = a[i+j], v = a[i+j+len/2]*root[step*j];
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
            }
        }
    }
    if (invert)
    {
        for (int i = 0; i<n; i++)
            a[i] /= n;
    }
}

/* FFT polynomial Multiplication with higher precision */
void multiply(const vi &a, const vi &b, vi &res)
{
    vector <base> fa_big, fb_big;
    vector <base> fa_small, fb_small;
    int cut_val = sqrt(P);
    int n = 1;
    while(n < max(sz(a),sz(b)))
        n <<= 1;
    n <<= 1;
    fa_big.resize(n);
    fa_small.resize(n);
    fb_big.resize(n);
    fb_small.resize(n);
    for (int i = 0; i<sz(a); i++)
    {
        fa_big[i] = a[i]/cut_val;
        fa_small[i] = a[i]%cut_val;
    }
    for (int i = 0; i<sz(b); i++)
    {
        fb_big[i] = b[i]/cut_val;
        fb_small[i] = b[i]%cut_val;
    }
    fft(fa_big,0), fft(fb_big,0);
```

```
    fft(fa_small, 0), fft(fb_small, 0);
    vector <base> fa_big_2(all(fa_big));
    vector <base> fa_small_2(all(fa_small));
    for (int i = 0; i<n; i++)
    {
        fa_big_2[i] *= fb_big[i];
        fa_big[i] *= fb_small[i];
        fa_small[i] *= fb_small[i];
        fa_small_2[i] *= fb_big[i];
    }
    fft(fa_small,1);
    fft(fa_small_2, 1);
    fft(fa_big, 1);
    fft(fa_big_2, 1);
    res.resize(n);
    for (int i = 0; i<n; i++)
    {
        int ss = (int64_t)round(fa_small[i].real());
        int sb = (int64_t)round(fa_small_2[i].real());
        int bs = (int64_t)round(fa_big[i].real());
        int bb = (int64_t)round(fa_big_2[i].real());
        res[i] = ss;
        res[i] += (sb+bs)*cut_val;
        res[i] += bb*cut_val*cut_val;
    }
}


/* FFT polynomial Multiplication with less precision */
void multiply(const vi &a, const vi &b, vi &res)
{
    vector <base> fa(all(a)), fb(all(b));
    int n = 1;
    while(n < max(sz(a),sz(b)))
        n <<= 1;
    n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,0), fft(fb,0);
    for (int i = 0; i<n; i++)
        fa[i] *= fb[i];
    fft(fa,1);
    res.resize(n);
    for (int i = 0; i<n; i++)
        res[i] = int64_t(fa[i].real()+(fa[i].real()>0?0.5:-0.5));
}
```

3.10 Berlekamp-Massey Algorithm

초항 $3k$ 개로 $k*k$ 행렬 전이를 갖는 문제를 선형점화식으로 변환 + Kitamasa method 를 이용한 $k^2 \log n$ 시간 계산. 초항만 몇개 구하고 `guess_nth_term` 쓰면 ok. 점화식을 알때는 `get_nth`로 키타마사만 쓰기.
Todo : FFT를 적용한 $k \log k \log n$ 키타마사 구현하기.

```
/*
    BerlekampMassey Algorithm Implementation :
    No touch unless absolutely necessary
    Implementation : Koosaga
*/
const int mod = 1000000007;
using lint = long long;
lint ipow(lint x, lint p){
    lint ret = 1, piv = x;
    while(p){
        if(p & 1) ret = ret * piv % mod;
        piv = piv * piv % mod;
        p >>= 1;
    }
    return ret;
}
vector<int> berlekamp_massey(vector<int> x){
    vector<int> ls, cur;
    int lf, ld;
    for(int i=0; i<x.size(); i++){
        lint t = 0;
        for(int j=0; j<cur.size(); j++){
            t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
        }
        if((t - x[i]) % mod == 0) continue;
        if(cur.empty()){
            cur.resize(i+1);
            lf = i;
            ld = (t - x[i]) % mod;
            continue;
        }
        lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
```

```
        vector<int> c(i-lf-1);
        c.push_back(k);
        for(auto &j : ls) c.push_back(-j * k % mod);
        if(c.size() < cur.size()) c.resize(cur.size());
        for(int j=0; j<cur.size(); j++){
            c[j] = (c[j] + cur[j]) % mod;
        }
        if(i-lf+(int)ls.size()>=(int)cur.size()){
            tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
        }
        cur = c;
    }
    for(auto &i : cur) i = (i % mod + mod) % mod;
    return cur;
}
int get_nth(vector<int> rec, vector<int> dp, lint n){
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if(m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mul = [&rec](vector<int> v, vector<int> w){
        int m = v.size();
        vector<int> t(2 * m);
        for(int j=0; j<m; j++){
            for(int k=0; k<m; k++){
                t[j+k] += 1ll * v[j] * w[k] % mod;
                if(t[j+k] >= mod) t[j+k] -= mod;
            }
        }
        for(int j=2*m-1; j>=m; j--){
            for(int k=1; k<=m; k++){
                t[j-k] += 1ll * t[j] * rec[k-1] % mod;
                if(t[j-k] >= mod) t[j-k] -= mod;
            }
        }
        t.resize(m);
        return t;
    };
    while(n){
        if(n & 1) s = mul(s, t);
        t = mul(t, t);
        n >>= 1;
    }
    lint ret = 0;
    for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
    return ret % mod;
}
int guess_nth_term(vector<int> x, lint n){
    if(n < x.size()) return x[n];
    vector<int> v = berlekamp_massey(x);
    if(v.empty()) return 0;
    return get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no duplicate please..
vector<int> get_min_poly(int n, vector<elem> M)
{
    // smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
    vector<int> rnd1, rnd2;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){
        return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for(int i=0; i<n; i++){
        rnd1.push_back(randint(1, mod - 1));
        rnd2.push_back(randint(1, mod - 1));
    }
    vector<int> gobs;
    for(int i=0; i<2*n+2; i++){
        int tmp = 0;
        for(int j=0; j<n; j++){
            tmp += 1ll * rnd2[j] * rnd1[j] % mod;
            if(tmp >= mod) tmp -= mod;
        }
        gobs.push_back(tmp);
        vector<int> nxt(n);
        for(auto &i : M){
            nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
            if(nxt[i.x] >= mod) nxt[i.x] -= mod;
        }
        rnd1 = nxt;
    }
    auto sol = berlekamp_massey(gobs);
```

```
    reverse(sol.begin(), sol.end());
    return sol;
}
lint det(int n, vector<elem> M){
    vector<int> rnd;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){
        return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
    for(auto &i : M){
        i.v = 1ll * i.v * rnd[i.y] % mod;
    }
    auto sol = get_min_poly(n, M)[0];
    if(n % 2 == 0) sol = mod - sol;
    for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
    return sol;
}
```

4 Geometry

4.1 CCW

```
//Is 3 points Counterclockwise? 1 : -1
//0 : on same line
int CCW(Point a, Point b, Point c)
{
    int op = a.x*b.y + b.x*c.y + c.x*a.y;
    op -= (a.y*b.x + b.y*c.x + c.y*a.x);
    if (op > 0)
        return 1;
    else if (op == 0)
        return 0;
    else
        return -1;
}
```

4.2 Point in polygon

Returns boolean, if point is in the polygon (represented as vector of points).

```
// point in polygon test
inline double is_left(Point p0, Point p1, Point p2)
{
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}
bool is_in_polygon(Point p, vector<Point>& poly)
{
    int wn = 0;
    for (int i = 0; i < poly.size(); ++i)
    {
        int ni = (i + 1 == poly.size()) ? 0 : i + 1;
        if (poly[i].y <= p.y)
        {
            if (poly[ni].y > p.y)
                if (is_left(poly[i], poly[ni], p) > 0)
                    ++wn;
        }
        else
        {
            if (poly[ni].y <= p.y)
                if (is_left(poly[i], poly[ni], p) < 0)
                    --wn;
        }
    }
    return wn != 0;
}
```

4.3 Length of Segment Union

Length of segment union, from vector of {start, end}.

```
//Src : e-maxx
int length_union(const vector<pair<int, int>> &a)
{
    int n = a.size();
    vector<pair<int, bool>> x(n*2);
    for (int i = 0; i < n; i++)
    {
        x[i*2] = {a[i].first, false};
        x[i*2+1] = {a[i].second, true};
    }

    sort(x.begin(), x.end());

    int result = 0;
    int c = 0;
    for (int i = 0; i < n * 2; i++)
```

```
    {
        if (i > 0 && x[i].first > x[i-1].first && c > 0)
            result += x[i].first - x[i-1].first;
        if (x[i].second)
            c--;
        else
            c++;
    }
    return result;
}
```

4.4 Closest Pair Problem

Requires : Points must be sorted with x-axis.
Runs in $\mathcal{O}(n \log^2 n)$

```
int dist (Point &p, Point &q)
{
    return (p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y);
}

bool compare(Point &p, Point &q)
{
    return (p.x < q.x);
}

bool ycompare(Point &p, Point &q)
{
    return (p.y<q.y);
}
Point pts[101010];

int closest_pair(Point p[], int n)
{
    if (n==2)
        return dist(p[0], p[1]);
    if (n==3)
    {
        return min(dist(p[0],p[1]),
        min(dist(p[1],p[2]),dist(p[0],p[2])));
    }
    Point mid[n];
    int line = (p[n/2 - 1].x + p[n/2].x) / 2;
    int d = min(closest_pair(p, n/2), closest_pair(p + n/2, n - n/2));
    int pp = 0;
    for (int i = 0; i < n; i++)
    {
        int t = line - p[i].x;
        if (t*t < d)
        {
            mid[pp] = p[i];
            pp++;
        }
    }
    sort(mid,mid+pp,ycompare);
    for (int i = 0; i < pp - 1; i++)
        for (int j = i + 1; j < pp && mid[j].y - mid[i].y < d; j++)
            d = min(d, dist(mid[i], mid[j]));
    return d;
}
```

4.5 Graham Scan + Rotating Calipers

```
struct point
{
    int x, y, dx, dy;
    point operator+(const point &other)
    {
        return {x+other.x, y+other.y};
    }
    point operator-(const point &other)
    {
        return {x-other.x, y-other.y};
    }
    point operator*(const int C)
    {
        return {x*C, y*C, dx, dy};
    }
    int normsq()
    {
        return x*x+y*y;
    }
};

int dist(point &a, point &b)
{
```

```
    return (a-b).normsq();
}
class convex_hull
{
public:
    int n;
    static int ccw (point a, point b, point c)
    {
        int v = (b.x - a.x) * (c.y - a.y) - (b.y-a.y)*(c.x-a.x);
        if (v > 0) return 1;
        if (!v) return 0;
        return -1;
    }
    static bool compy (point &a, point &b)
    {
        if (a.y == b.y) return a.x < b.x;
        return a.y < b.y;
    }
    vector <point> pt;
    vector <point> convex;
    void graham_scan()
    {
        convex.clear();
        sort(all(pt), compy);
        point down = pt[0];
        sort(all(pt), [&down](auto a, auto b)
        {
            int u = ccw(down, a, b);
            if (u!=0) return u>0;
            else return dist(down, a) < dist(down, b);
        });
        convex.push_back(pt[0]);
        convex.push_back(pt[1]);
        for (int i = 2; i<n; i++)
        {
            while(convex.size() > 1)
            {
                point tp = convex.back();
                convex.pop_back();
                point tptp = convex.back();
                if (ccw(tptp,tp,pt[i]) > 0)
                {
                    convex.push_back(tp);
                    break;
                }
            }
            convex.push_back(pt[i]);
        }
    }
    void rotating_calipers()
    {
        int msz = 0;
        int t = 0;
        point o = {0, 0};
        int nc = convex.size();
        point a = convex[0], b = convex[0];
        for(int i=0; i<convex.size(); i++)
        {
            while(t+1 < convex.size() &&
            ccw(o, convex[i+1]-convex[i], convex[t+1]-convex[t])>=0)
            {
                if (msz < dist(convex[i], convex[t]))
                {
                    msz = dist(convex[i], convex[t]);
                    a = convex[i], b = convex[t];
                }
                t++;
            }
            if (msz < dist(convex[i], convex[t]))
            {
                msz = dist(convex[i], convex[t]);
                a = convex[i], b = convex[t];
            }
        }
        printf("%lld %lld %lld %lld\n",a.x,a.y,b.x,b.y);
    }
};
```

4.6 Intersection of Line Segment

```
//jason9319.tistory.com/358. modified
int isIntersect(Point a, Point b, Point c, Point d)
{
    int ab = ccw(a, b, c)*ccw(a, b, d);
```

```
    int cd = ccw(c, d, a)*ccw(c, d, b);
    if (ab == 0 && cd == 0)
    {
        if (a > b)swap(a, b);
        if (c > d)swap(c, d);
        return (c <= b&&a <= d);
    }
    return (ab <= 0 && cd <= 0);
}
```

5 Graphs

5.1 Topological Sorting

Topological sorting with dfs

```
vector <int> graph[V];
bool visited[V];
vector <int> sorted;

void dfs(int root)
{
    visited[root] = 1;
    for (auto it:graph[root])
    {
        if (!visited[it])
            dfs(it);
    }
    sorted.push_back(root);
}

int main()
{
    int n, m;
    scanf("%d%d",&n,&m);
    for (int i = 0; i<m; i++)
    {
        int small,big;
        scanf("%d%d",&small,&big);
        graph[small].push_back(big);
    }
    for (int i = 1; i<=n; i++)
        if (!visited[i])
            dfs(i);
    reverse(sorted.begin(),sorted.end()); // must reverse!
}
```

5.2 Lowest Common Ancestor

LCA Algorithm by sparse table.
minlen : (x, y) 사이를 잇는 간선 중 최소 길이 간선.
maxlen : (x, y) 사이를 잇는 간선 중 최대 길이 간선.

```
int n, k;
bool visited[101010];
int par[101010][21], maxedge[101010][21], minedge[101010][21];
int d[101010];
vector <pii> graph[101010]; // {destination, weight}
void dfs(int here,int depth) // run dfs(root,0)
{
    visited[here] = true;
    d[here] = depth;
    for (auto there : graph[here])
    {
        if (visited[there.first])
            continue;
        dfs(there.first, depth + 1);
        par[there.first][0] = here;
        maxedge[there.first][0] = there.second;
        minedge[there.first][0] = there.second;
    }
}

void precomputation()
{
    for (int i = 1; i<21; i++)
    {
        for (int j = 1; j<=n; j++)
        {
            par[j][i] = par[par[j][i-1]][i-1];
            maxedge[j][i] = max(maxedge[j][i - 1],
                maxedge[par[j][i - 1]][i - 1]);
            minedge[j][i] = min(minedge[j][i - 1],
                minedge[par[j][i - 1]][i - 1]);
        }
    }
}
```

```
pii lca(int x, int y)
{
    int maxlen = INT_MIN;
    int minlen = INT_MAX;
    if (d[x]>d[y])
        swap(x,y);
    for (int i = 20; i>=0; i--)
    {
        if (d[y]-d[x] >= (1<<i))
        {
            minlen = min(minlen,minedge[y][i]);
            maxlen = max(maxlen,maxedge[y][i]);
            y = par[y][i];
        }
    }
    if (x==y)
        return {minlen, maxlen};
    for (int i = 20; i>=0; i--)
    {
        if (par[x][i] != par[y][i])
        {
            minlen = min(minlen,min(minedge[x][i],minedge[y][i]));
            maxlen = max(maxlen,max(maxedge[x][i],maxedge[y][i]));
            x = par[x][i];
            y = par[y][i];
        }
    }
    minlen = min(minlen,min(minedge[x][0],minedge[y][0]));
    maxlen = max(maxlen,max(maxedge[x][0],maxedge[y][0]));

    int lca_point = par[x][0];
    return {minlen,maxlen};
}

void tobedone()
{
    dfs(1,0);
    precomputation();
}
```

5.3 MST Kruskal Algorithm

Based on Union-Find implementation
$\mathcal{O}(E \log E)$ if path-compressed Union Find.

```
int Kruskal()
{
    int mstlen = 0;
    sort(edgelist.begin(),edgelist.end());
    for (auto it:edgelist)
    {
        if (dsu.Find(it.s)==dsu.Find(it.e)) // Cycle Detection
            continue;
        else
        {
            dsu.unite(it.s,it.e);
            mstlen += it.w;
        }
    }
    return mstlen;
}
```

5.4 MST Prim Algorithm

```
vector <pii> Tree[101010];
// Note that we use {weight, destination} pair here.
// This is to use priority_queue!
bool visit[101010];
priority_queue <pii, vector<pii>, greater<pii>> pq;
void add(int i)
{
    visit[i] = true;
    for (auto it:Tree[i])
        pq.push(it);
}

int Prim(int start)
{
    int mstlen = 0;
    add(start);
    while(!pq.empty())
    {
        int cur = pq.top().second;
        int weight = pq.top().first;
        pq.pop();
```

```
        if (visit[cur])
            continue;
        else
        {
            mstlen+=weight;
            add(cur);
        }
    }
    return mstlen;
}
```

5.5 Dinic's Algorithm

```
struct Edge
{
    int u, v;
    ll cap, flow;
    Edge() {}
    Edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic
{
    int N;
    vector<Edge> E;
    vector<vector<int>> g;
    vector<int> d, pt;

    Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

    void AddEdge(int u, int v, ll cap)
    {
        if (u != v)
        {
            E.push_back(Edge(u, v, cap));
            g[u].push_back(E.size() - 1);
            E.push_back(Edge(v, u, 0));
            g[v].push_back(E.size() - 1);
        }
    }

    bool BFS(int S, int T)
    {
        queue<int> q({S});
        fill(d.begin(), d.end(), N + 1);
        d[S] = 0;
        while(!q.empty())
        {
            int u = q.front();
            q.pop();
            if (u == T) break;
            for (int k: g[u])
            {
                Edge &e = E[k];
                if (e.flow < e.cap && d[e.v] > d[e.u] + 1)
                {
                    d[e.v] = d[e.u] + 1;
                    q.push(e.v);
                }
            }
        }
        return d[T] != N + 1;
    }

    ll DFS(int u, int T, ll flow = -1)
    {
        if (u == T || flow == 0) return flow;
        for (int &i = pt[u]; i < g[u].size(); i++)
        {
            Edge &e = E[g[u][i]];
            Edge &oe = E[g[u][i]^1];
            if (d[e.v] == d[e.u] + 1)
            {
                ll amt = e.cap - e.flow;
                if (flow != -1 && amt > flow) amt = flow;
                if (ll pushed = DFS(e.v, T, amt))
                {
                    e.flow += pushed;
                    oe.flow -= pushed;
                    return pushed;
                }
            }
        }
        return 0;
```

```
    }

    ll MaxFlow(int S, int T)
    {
        ll total = 0;
        while (BFS(S, T))
        {
            fill(pt.begin(), pt.end(), 0);
            while (ll flow = DFS(S, T))
                total += flow;
        }
        return total;
    }
};
```

5.6 Cosaraju SCC Algorithm

```
struct Cosaraju
{
    int V, E;
    vector <vector<int>> G;
    vector <vector<int>> rG;
    vector <vector <int>> scc;
    vector <int> dfs_stack;
    vector <bool> visit;
    Cosaraju(int n = 0)
    {
        V = n;
        G.resize(V+5);
        rG.resize(V+5);
        visit.resize(V+5);
        scc.resize(V+5);
    }
    void re_init(int n)
    {
        V = n;
        G.resize(V+5);
        rG.resize(V+5);
        visit.resize(V+5);
        scc.resize(V+5);
    }
    void AddEdge(int u, int v)
    {
        G[u].push_back(v);
        rG[v].push_back(u);
    }
    void dfs(int r)
    {
        visit[r] = true;
        for (auto it:G[r])
            if (!visit[it])
                dfs(it);
        dfs_stack.push_back(r);
    }
    void rev_dfs(int r, int scc_num)
    {
        visit[r] = true;
        scc[scc_num].push_back(r);
        for (auto it:rG[r])
            if (!visit[it])
                rev_dfs(it, scc_num);
    }
    void find_scc()
    {
        fill(all(visit),0);
        for (int i = 1; i<=V; i++)
            if (!visit[i])
                dfs(i);
        fill(all(visit),0);
        int scc_count = 0;
        while(!dfs_stack.empty())
        {
            int tp = dfs_stack.back();
            dfs_stack.pop_back();
            if (!visit[tp])
            {
                rev_dfs(tp, scc_count);
                scc_count++;
            }
        }
        scc.resize(scc_count);
        for (int i = 0; i<scc_count; i++)
            sort(all(scc[i]));
        sort(all(scc),[](vector<int>&a, vector<int>&b)->bool{return a[0]<b[0];}
```

```
    }
};
```

5.7 MCMF

```
const int INF = 0x7f7f7f7f7f7f7f7f;
const int MX = 820;
struct MCMF
{
    vector <int> G[MX];
    int cap[MX][MX] = {0};
    int cost[MX][MX] = {0};
    int flow[MX][MX] = {0};
    pair<int, int> MinCostMaxFlow(int source, int sink)
    // Maxflow, mincost of flow
    {
        int maxflow = 0, mincost = 0;
        while(true)
        {
            int fflow = 0;
            int dist[MX], prev[MX];
            bool inQ[MX];
            fill(inQ, inQ+MX, 0);
            fill(dist, dist + MX, INF);
            fill(prev, prev + MX, -1);
            deque <int> q;
            dist[source] = 0;
            inQ[source] = true;
            q.push_back(source);
            while (!q.empty())
            {
                int here = q.front();
                q.pop_front();
                inQ[here] = false;
                for (int i = 0; i < G[here].size(); i++)
                {
                    int nxt = G[here][i];
                    int cst = cost[here][nxt];
                    if(cap[here][nxt]-flow[here][nxt] > 0
                    && dist[nxt] > dist[here] + cst)
                    {
                        dist[nxt] = dist[here] + cst;
                        prev[nxt] = here;
                        if (!inQ[nxt])
                        {
                            q.push_back(nxt);
                            inQ[nxt] = true;
                            if (dist[q.back()]<dist[q.front()])
                            {
                                q.push_front(q.back());
                                q.pop_back();
                            }
                        }
                    }
                }
            }
            if(prev[sink] == -1)
                break;
            fflow = INF;
            for(int i=sink; i!=source; i=prev[i])
                fflow = min(maxflow, cap[prev[i]][i] - flow[prev[i]][i]);
            for(int i=sink; i!=source; i=prev[i])
            {
                mincost += fflow * cost[prev[i]][i];
                flow[prev[i]][i] += fflow;
                flow[i][prev[i]] -= fflow;
            }
            maxflow += maxflow;
        }
        return {maxflow, mincost};
    }
    void AddEdge(int u, int v, int cp, int cs)
    {
        if (u != v)
        {
            G[u].push_back(v); G[v].push_back(u);
            cap[u][v] = cp;
            cost[u][v] = cs;
            cost[v][u] = -cs;
        }
    }
};
```

6 Shortest Path

모든 간선의 가중치를 필요할 때마다 0으로 잘 초기화했는지 확인하기.

6.1 Dijkstra

$\mathcal{O}(E \log V)$ Single-Start-Shortest-Path.
가중치에 음수 없는거 항상 확인하고 쓰기.

```
const int INF = 987654321;
const int MX = 105050;
struct Edge
{
    int dest, w;
    bool operator<(const Edge &p) const
    {
        return w > p.w;
    }
};

bool relax(Edge edge, int u, int dist[])
{
    bool flag = 0;
    int v = edge.dest, w = edge.w;
    if (dist[v] > dist[u] + w && (dist[u]!=INF))
    {
        flag = true;
        dist[v] = dist[u]+w;
    }
    return flag;
}

int dijkstra(int dist[], int start, vector<Edge> graph[])
{
    fill(dist,dist+MX,INF);
    dist[start] = 0;
    priority_queue<Edge> pq;
    pq.push({start,0});
    while(!pq.empty())
    {
        Edge x = pq.top();
        int v = x.dest, w = x.w;
        pq.pop();
        if (w>dist[v])
            continue;
        for (auto ed : graph[v])
            if (relax(ed, v, dist))
                pq.push({ed.dest,dist[ed.dest]});
    }
}
```

6.2 Bellman Ford

$\mathcal{O}(EV)$ Single-Start-Shortest-Path.
Not working for graph with minus cycle $\rightarrow$ must detect.

```
struct Edge
{
    int u, v, w;
};
vector <Edge> edgelist;
int V, E;
int dist[V+1];
bool relax_all_edge()
{
    bool flag = false;
    for (auto it:edgelist)
    {
        int u = it.u, v = it.v, w = it.w;
        if (dist[v] > dist[u] + w && (dist[u]!=INF))
        {
            flag = true;
            dist[v] = dist[u]+w;
        }
    }
    return flag;
}
int bellman_ford()
{
    fill(dist,dist+V+2,INF);
    dist[1] = 0;
    for (int i = 0; i<V-1; i++)
    {
        relax_all_edge();
    }
    if (relax_all_edge())
        return -1;
    else
        return 0;
```

```
}
```

6.3 SPFA Algorithm

Average $\mathcal{O}(E)$, worst $\mathcal{O}(VE)$.

```
// Optimized SPFA, https://hongjun7.tistory.com/170 , style modifications.
struct Edge
{
    int dest, w;
    bool operator<(const Edge &p) const
    {
        return w > p.w;
    }
};
bool inQ[MX];
void spfa(int dist[], int start, vector<Edge> graph[])
{
    fill(dist, dist + MX, INF);
    deque <int> q;
    dist[start] = 0;
    inQ[start] = true;
    q.push_back(start);
    while (!q.empty())
    {
        int here = q.front();
        q.pop_front();
        inQ[here] = false;
        for (int i = 0; i < graph[here].size(); i++)
        {
            int next = graph[here][i].dest;
            int cost = graph[here][i].w;
            if(dist[next] > dist[here] + cost)
            {
                dist[next] = dist[here] + cost;
                if (!inQ[next])
                {
                    q.push_back(next);
                    inQ[next] = true;
                    if (dist[q.back()]<dist[q.front()])
                    {
                        q.push_front(q.back());
                        q.pop_back();
                    }
                }
            }
        }
    }
}
```

6.4 Floyd-Warshall

Works on adjacency matrix, in $\mathcal{O}(V^3)$.

```
int d[120][120];
int n;
void Floyd_Warshall()
{
    fill(d, d+sizeof(d),INF);
    //---Edges Here---//
    for (int i = 1; i<=n; i++)
        for (int j = 1; j<=n; j++)
            for (int k = 1; k<=n; k++)
                d[j][k] = MIN(d[j][k],d[j][i]+d[i][k]);
}
```

7 Dynamic

7.1 DP optimization Technique

7.1.1 Convex Hull Trick

$\mathrm{dp}[i] = \min_{j<i} (\mathrm{dp}[j] + a[i]b[j])$ 이고, b가 단조 감소하며, (현재 구현 기준) a가 단조 증가할 때, 직선 `y = b[j]*x + dp[j]`들을 기준으로 해석해서 시간 복잡도를 줄인다.

```
ll A[100020];
ll B[100020];
ll dp[100020];
//dp[i] = dp[j]+A[i]*B[j];
typedef struct linear
{
    ll a,b;
    double xpos;
    linear(ll x=0, ll y=0, double z=0): a(x),b(y),xpos(z){}
    ll cal(ll n){return a*n+b;}
}linear;
double cross(linear l, linear m)
{
    return (double)(l.b-m.b)/(double)(m.a-l.a);
}
```

```
linear s[100020];
int main()
{
    ll n;
    scanf("%lld",&n);
    for(int i=0; i<n; i++)
        scanf("%lld",A+i);
    for(int i=0; i<n; i++)
        scanf("%lld",B+i);
    s[0]=linear(B[0],0,-1e18);
    ll fpos = 0, pt = 0;
    // pt: 스택의 맨 위 fpos: 대입할 직선 결정
    for(int i=1; i<n; i++)
    {
        // 시작할 위치가 A[i](좌표) 보다 처음으로 크거나 같아지는 순간
        while(s[fpos].xpos<A[i]&&fpos<=pt)
            fpos++;
        dp[i]=s[--fpos].cal(A[i]);
        linear newlin = linear(B[i],dp[i],0);
        while(pt>0&&cross(s[pt],newlin)<=s[pt].xpos)
        {
            if(pt==fpos)
                fpos--;
            pt--;
        }
        newlin.xpos = cross(s[pt],newlin);
        s[++pt] = newlin;
    }
    printf("%lld",dp[n-1]);
}
```

7.1.2 Li Chao Tree

```
using pii = pair<int, int>;
#define int ll
const int INF = 2e18;
struct Line // Linear function ax + b
{
    int a, b;
    int eval(int x)
    {
    return a*x + b;
    }
};

struct Node
{
    int left, right;
    int start, end;
    Line f;
};
Node new_node(int a, int b)
{
    return {-1,-1,a,b,{0,-INF}};
}
vector <Node> nodes;

struct LiChao
{
    void init(int min_x, int max_x)
    {
        nodes.push_back(new_node(min_x, max_x));
    }
    void insert(int n, Line new_line)
    {
        int xl = nodes[n].start, xr = nodes[n].end;
        int xm = (xl + xr)/2;
        Line llo, lhi;
        llo = nodes[n].f, lhi = new_line;
        if (llo.eval(xl) >= lhi.eval(xl))
            swap(llo, lhi);
        if (llo.eval(xr) <= lhi.eval(xr))
        {
            nodes[n].f = lhi;
            return;
        }
        else if (llo.eval(xm) > lhi.eval(xm))
        {
            nodes[n].f = llo;
            if (nodes[n].left == -1)
            {
                nodes[n].left = nodes.size();
                nodes.push_back(new_node(xl,xm));
            }
            insert(nodes[n].left, lhi);
```

```
        }
        else
        {
            nodes[n].f = lhi;
            if (nodes[n].right == -1)
            {
                nodes[n].right = nodes.size();
                nodes.push_back(new_node(xm+1,xr));
            }
            insert(nodes[n].right,llo);
        }
    }

    int get(int n, int q)
    {
        if (n == -1) return -INF;
        int xl = nodes[n].start, xr = nodes[n].end;
        int xm = (xl + xr)/2;
        if (q > xm)
            return max(nodes[n].f.eval(q), get(nodes[n].right, q));
        else
            return max(nodes[n].f.eval(q), get(nodes[n].left, q));
    }

    int evaluate(int pt)
    {
        return get(0, pt);
    }
};
LiChao CHT;
int32_t main()
{
    usecppio
    int Q;
        CHT.init(-2e12, 2e12);
    cin >> Q;
    while(Q--)
    {
        int tp;
        cin >> tp;
        if (tp == 1)
        {
            int a, b;
            cin >> a >> b;
            CHT.insert(0, {a, b});
        }
        else
        {
            int x;
            cin >> x;
            cout << CHT.evaluate(x) << '\n';
        }
    }
}
```

7.1.3 Knuth Optimization

dp 점화식이 다음 조건을 만족할 때, $O(n^3)$ 을 $O(n^2)$ 로 줄인다.

- dp[i][j] $= \min_{i<k<j}$ (dp[i][k] + dp[k][j]) + C[i][j]
- C[a][c] + C[b][d] $\leq$ C[a][d] + C[b][c] $\leq$ 2C[a][d] when $a \leq b \leq c \leq d$.

이때, dp[i][j]가 최소가 되는 k는 $k_{(i,j-1)} \leq k_{(i,j)} \leq k_{(i+1,j)}$ 를 만족한다.

```
for(i = 1; i <= n; i++)
{
    dp[i][i] = 0;
    p[i][i] = i;
}

for(j = 2; j <= n; j++)
{
    for(i = 1; i <= n-j+1; i++){
        s = i, e = i+j-1;
        dp[s][e] = vMax;
        for(k = p[s][e-1]; k <= p[s+1][e]; k++)
        {
            if(dp[s][e] > dp[s][k] + dp[k+1][e])
            {
                dp[s][e] = dp[s][k] + dp[k+1][e];
                p[s][e] = k;
            }
        }
        dp[s][e] += cost[s][e];
    }
```

```
}
```

7.2 Longest Increasing Subsequence

Find LIS in $\mathcal{O}(n \log n)$ time.

```
vector <int> sequence;
vector <int> L;
int lis_len;
int position[BIG];
int lis[BIG];
int lis_pushed[BIG];
int n;
void FindLIS(vector <int> &seq)
{
    L.push_back(seq[0]);
    position[0] = 0;
    for (int i = 1; i<n; i++)
    {
        int u = L.size();
        if (seq[i] > L[u-1])
        {
            position[i] = u;
            L.push_back(seq[i]);
        }
        else
        {
            int pos = lower_bound(L.begin(),L.end(),seq[i])-L.begin();
            L[pos] = seq[i];
            position[i] = pos;
        }
    }
    lis_len=L.size();
    int lookingfor = lis_len-1;
    for (int i = n-1; i>=0; i--)
    {
        if (lis_pushed[position[i]]==0 && lookingfor == position[i])
        {
            lis[position[i]] = seq[i];
            lis_pushed[position[i]]=1;
            lookingfor--;
        }
    }
}
```

Using multiset...

```
vector <int> sequence;
int n, lislen;
multiset<int> increase;

void find_lis()
{
    for (int i = 0; i<n; i++)
    {
        auto it = lower_bound(all(increase),sequence[i]);
        if (it == increase.begin())
            increase.insert(sequence[i]);
        else
        {
            --it;
            increase.erase(it);
            increase.insert(sequence[i]);
        }
    }
    lislen = increase.size();
}
```

7.3 Largest Sum Subarray

Computes sum of largest sum subarray in $\mathcal{O}(N)$

```
void consecsum(int n)
{
    dp[0] = number[0];
    for (int i = 1; i<n; i++)
        dp[i] = MAX(dp[i-1]+number[i],number[i]);
}

int maxsum(int n)
{
    consecsum(n);
    int max_sum=-INF;
    for (int i = 0; i<n; i++)
        dp[i] > max_sum ? max_sum = dp[i] : 0;
    return max_sum;
}
```

7.4 0-1 Knapsack

```
int dp[N][W];
```

```
int weight[N];
int value[N];
void knapsack()
{
    for (int i = 1; i<=n; i++)
    {
        for (int j = 0; j<=W; j++)
            dp[i][j] = dp[i-1][j];
        for (int j = weight[i]; j<=W; j++)
            dp[i][j] = max(dp[i][j], dp[i-1][j-weight[i]]+value[i]);
    }
}
```

7.5 Longest Common Subsequence

```
//input : two const char*
//output : their LCS, in c++ std::string type
string lcsf(const char *X,const char *Y)
{
    int m = (int)strlen(X);
    int n = (int)strlen(Y);
    int L[m+1][n+1];
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i-1] == Y[j-1])
                L[i][j] = L[i-1][j-1] + 1;
            else
                L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }
    int index = L[m][n];

    char lcsstring[index+1];
    lcsstring[index] = 0;

    int i = m, j = n;
    while (i > 0 && j > 0)
    {
        if (X[i-1] == Y[j-1])
        {
            lcsstring[index-1] = X[i-1];
            i--; j--; index--;
        }
        else if (L[i-1][j] > L[i][j-1])
            i--;
        else
            j--;
    }
    string lcsstr = lcsstring;
    return lcsstr;
}
```

7.6 Edit Distance

```
int edit_dist[1010][1010];
int Editdist(string &s, string &t)
{
    int slen = s.length();
    int tlen = t.length();
    for (int i = 1; i<=slen; i++)
        edit_dist[i][0] = i;
    for (int i = 1; i<=tlen; i++)
        edit_dist[0][i] = i;
    for (int i = 1; i<=tlen; i++)
    {
        for (int j = 1; j<=slen; j++)
        {
            if (s[j-1]==t[i-1])
                edit_dist[j][i] = edit_dist[j-1][i-1];
            else
                edit_dist[j][i] = min(edit_dist[j-1][i]+1,
                    min(edit_dist[j-1][i-1]+1,edit_dist[j][i-1]+1));
        }
    }
    return edit_dist[slen][tlen];
}
```

8 String

8.1 KMP Algorithm

Pi 배열의 정의 : $str[0]$ 부터 $str[i]$ 까지 중 접두사가 접미사와 같은 부분만큼의 길이.

```
// Original Author : bowbowbow (bowbowbow.tistory.com)
vector<int> getPi(string p)
{
    int j = 0;
    int plen = p.length();
    vector<int> pi;
    pi.resize(plen);
    for(int i = 1; i< plen; i++)
    {
        while((j > 0) && (p[i] !=  p[j]))
            j = pi[j-1];
        if(p[i] == p[j])
        {
            j++;
            pi[i] = j;
        }
    }
    return pi;
}
vector <int> kmp(string s, string p)
{
    vector<int> ans;
    auto pi = getPi(p);
    int slen = s.length(), plen = p.length(), j = 0;
    for(int i = 0 ; i < slen ; i++)
    {
        while(j>0 && s[i] != p[j])
            j = pi[j-1];
        if(s[i] == p[j])
        {
            if(j==plen-1)
            {
                ans.push_back(i-plen+1);
                j = pi[j];
            }
            else
                j++;
        }
    }
    return ans;
}
```

8.2 Manacher's Algorithm

A[i] = i 번을 중심으로 하는 가장 긴 팰린드롬이 되는 반지름.

```
//original Author : Myungwoo (blog.myungwoo.kr)
int N,A[MAXN];
char S[MAXN];

void Manachers()
{
    int r = 0, p = 0;
    for (int i=1;i<=N;i++)
    {
        if (i <= r)
            A[i] = min(A[2*p-i],r-i);
        else
            A[i] = 0;
        while (i-A[i]-1 > 0 && i+A[i]+1 <= N
        && S[i-A[i]-1] == S[i+A[i]+1])
            A[i]++;
        if (r < i+A[i])
            r = i+A[i], p = i;
    }
}
```

8.3 Trie

```
struct Trie
{
    int trie[NODE_MAX][CHAR_N];
    int nxt = 1;
    void insert(const char* s)
    {
        int k = 0;
        for (int i = 0; s[i]; i++)
        {
            int t = s[i] - 'a';
            if (!trie[k][t])
            {
                trie[k][t] = nxt;
```

```
                nxt++;
            }
            k = trie[k][t];
        }
        trie[k][26] = 1;
    }
    bool find(const char* s, bool exact = false)
    {
        int k = 0;
        for (int i = 0; s[i]; i++)
        {
            int t = s[i] - 'a';
            if (!trie[k][t])
                return false;
            k = trie[k][t];
        }
        if (exact)
        {
            return trie[k][26];
        }
        return true;
    }
};
```

8.4 Rabin-Karp Hashing

Hashmap[k]에, 길이가 len인 부분 문자열의 해시값이 k 가 되는 시작점 인덱스 i 를 push.

```
const ll MOD = BIG_PRIME;
int L;
char S[STR_LEN];
int safemod(int n)
{
    if(n >= 0)
        return n % MOD;
    return ((-n/MOD+1)*MOD + n) % MOD;
}
vector <int> hashmap[MOD];
void Rabin_Karp(int len)
{
    int Hash = 0;
    int pp = 1;
    for(int i=0; i<=L-len; i++)
    {
        if(i == 0)
        {
            for(int j = 0; j<len; j++)
            {
                Hash = safemod(Hash + S[len-j-1]*pp);
                if(j < len-1)
                    pp = safemod(pp*2);
            }
        }
        else
            Hash = safemod(2*(Hash - S[i-1]*pp) + S[len+i-1]);
        hashmap[Hash].push_back(i);
    }
    return;
}
```

9 Miscellaneous

9.1 Binary and Ternary Search

Preventing stupid mistakes by writing garbage instead of proper binary search.
상황에 따라 lo와 hi 중 어느 쪽이 답인지 달라짐.

```
while(lo+1 < hi)
{
    int mid = (lo+hi)/2;

    if(chk(mid))
        lo = mid;
    else
        hi = mid;
}
```

Ternary search

```
double ternary_search(double l, double r)
{
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps)
    {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);    //return the maximum of f(x) in [l, r]
}
```

9.2 GCC Order Statistics Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;

void test()
{
    ordered_set X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5
}
```

9.3 Useful Bitwise Functions in C++

```
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_clzll(ll x);// number of leading zero
int __builtin_ctzll(ll x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
int __builtin_popcountll(ll x);// number of 1-bits in x

lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);

// compute next perm. ex) 00111, 01011, 01101, 01110, 10011, 10101..
ll next_perm(ll v)
{
  ll t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
```

9.4 Prime numbers

```
< 10^k    prime   # of prime          < 10^k             prime
------------------------------------------------------------------
```

1	7	4	10	9999999967
2	97	25	11	99999999977
3	997	168	12	999999999989
4	9973	1229	13	9999999999971
5	99991	9592	14	99999999999973
6	999983	78498	15	999999999999989
7	9999991	664579	16	9999999999999937
8	99999989	5761455	17	99999999999999997
9	999999937	50847534	18	999999999999999989

9.5 Time Complexity of Algorithms

- Map, Set : $\log n$ 삽입 삭제 탐색 , But very big constant
- PBDS_OST : $\log n$ 삽입 삭제 탐색 , But very very big constant
- Segment Tree : $\log n$ change, $\log n$ query
- Lazy Propagation : $\log n$ segment addition
- Fenwick Tree : Same with Segtree but faster
- DSU : Inverse Ackermann merge
- GCD : $\log n$
- Binary Exponentiation : $\log b$
- Fast Fourier Transform : $n \log n$ 다항식 곱셈
- Closest Pair DnC : $n \log n$
- Graham Scan : $n \log n$
- Topological Sort : n
- LCA : Build time $n \log n$, 쿼리당 $\log n$
- Kruskal, Prim : $n \log n$
- Dijkstra : $E \log V$
- Bellman Ford : EV
- SPFA : EV 이지만 매우 빠름
- Floyd-Warshall : V^3
- Convex Hull Trick : $n^2 \to n$ or $n \log n$
- Knuth Opt : $n^3 \to n^2$
- Longest Increasing Subsequence : $n \log n$
- 0-1 Knapsack : nw
- Longest Common Subsequence : n^2
- Edit Distance : n^2
- KMP : $N + M$ pattern matching
- Manacher : N palindrome finding
- Trie : M insert, M erase, M search
- Binary, Ternary : $\log n$

10 Checkpoints

10.1 Debugging

- $10^5 * 10^5 \Rightarrow$ OVERFLOW. 특히 for 문 안에서 `i * i < n` 할때 조심하기.
- If unsure with overflow, use `#define int long long` and stop caring.
- 행렬과 기하의 i, j 인덱스 조심. 헷갈리면 쓰면서 가기.
- Segment Tree, Trie, Fenwick 등 Struct 구현체 사용할 때는 항상 내부의 n 이 제대로 초기화되었는지 확인하기.
- Testcase가 여러 개인 문제는 항상 초기화 문제를 확인하기. 입력을 다 받지 않았는데 break나 return으로 끊어버리면 안됨.
- iterator 주의 : .end() 는 항상 맨 끝 원소보다 하나 더 뒤의 iterator. erase쓸 때는 iterator++ 관련된 문제들에 주의해야 한다.
- std::sort must compare with Strict weak ordering (Codejam 2020 1A-A)
- Memory Limit : Local variable은 int 10만개 정도까지만 사용. Global Variable의 경우 128MB면 대략 int 2000만 개까지는 잘 들어간다. long long은 절반. stack, queue, map, set 같은 특이한 컨테이너는 100만개를 잡으면 메모리가 버겁지만 vector 100만개는 잡아도 된다.
- Array out of Bound : 배열의 길이는 충분한가? Vector resize를 했다면 그것도 충분할까? 배열의 -1번에 접근한 적은 없는게 확실할까?
- Binary Search : 제대로 짠 게 맞을까? 1 차이 날 때 / lo == hi 일 때 등등. Infinite loop 주의하기.
- Graph : 반례 유의하기. Connected라는 말이 없으면 Disconnected. Acyclic 하다는 말이 없으면 Cycle 넣기, 특히 $A \leftrightarrow B$ 그래프로 2개짜리 사이클 생각하기.
- Set과 map은 매우 느리다.

10.2 Thinking

- 모든 경우를 다 할 수 없나? 왜 안 되지? 시간 복잡도 잘 생각해 보기. 정해의 Target Complexity를 먼저 생각하고 주요 알고리즘들의 Complexity로 짜맞추기.
  예를들어, 쿼리가 30만개 들어온다면 한 쿼리를 적어도 $\log n$ 에 처리할 방법이 아무튼 있다는 뜻.
- 그 방법이 뭐지? xxxx한 일을 어떤 시간복잡도에 실행하는 적절한 자료구조가 있다면?
  - 필요한 게 정렬성이라면 힙이나 map을 쓸 수 있고
  - multiset / multimap도 사용할 수 있고.. 느리지만.
- 단조함수이며, 충분히 빠르게 검증가능한가 : Binary Search.
- 차원이 높은 문제 : 차원 내려서 생각하기. 3 $\to$ 2.
- 이 문제가 사실 그래프 관련 문제는 아닐까?
  - 만약 그렇다면, '간선' 과 '정점' 은 각각..?
  - 간선과 정점이 몇 개 정도 있는가?
- 이 문제에 Overlapping Subproblem이 보이나?
  $\to$ Dynamic Programming 을 적용.
- Directed Graph, 특히 Cycle에 관한 문제 : Topological Sorting? (ex : SNUPC 2019 kdh9949)
- 답의 상한이 Reasonable 하게 작은가?
- output이 특정 수열/OX 형태 : 작은 예제를 Exhasutive Search. 모르는 무언가를 알기 위해서는 데이터가 필요하다.
- 그래프 문제에서, 어떤 "조건" 이 들어갔을 때 $\to$ 이 문제를 "정점을 늘림으로써" 단순한 그래프 문제로 바꿀 수 있나? (ex : SNUPC 2018 달빛 여우) 이를테면, 홀짝성에 따라 점을 2배로 늘림으로써?
- DP도 마찬가지. 어떤 조건을 단순화하기 위해 상태의 수를 사이사이에 집어넣을 수 있나? (ex : SNUPC 2018 실버런)
- DP State를 어떻게 나타낼 것인가? 첫 i개만을 이용한 답을 알면 $i+1$개째가 들어왔을 때 빠르게 처리할 수 있을까?
- 더 큰 table에서 시작해서 줄여가기. 특히 Memory가 모자라다면 Toggling으로 차원 하나 내릴 수 있는 경우도 상당히 많이 있다. 각 칸의 갱신 시간과 칸의 개수 찾기.
- Square root Decomposition : $O(n \log n)$ 이 생각나면 좋을 것 같지만 잘 생각나지 않고, 제한을 보니 $O(n\sqrt{n})$ 이면 될것도 같이 생겼을 때 생각해 보기. $O(\sqrt{n})$ 버킷 테크닉.
- 복잡도가 맞는데 왜인지 안 뚫리면 : 필요없는 long long을 사용하지 않았나? map 이나 set iterator들을 보면서 상수 커팅. 간단한 함수들을 inlining. 재귀를 반복문으로. 특히 Set과 Map은 끔찍하게 느리다.
- 마지막 생각 : 조금 추하지만 해싱이나 Random 또는 `bitset` 을 이용한 $n^2/64$ 같은걸로 뚫을 수 있나? 컴파일러를 믿고 10^8의 몇 배 정도까지는 내 봐도 될 수도. 의외로 Naive한 문제가 많다.