CUET Augnee Team Codebook

Chittagong University of Engineering and Technolgy

May 8, 2020

This collaborative document is the central place for the algorithms you will need for the ACM ICPC programming contest.

Contents

1 The Ritual

1.1 When Choosing a Problem

* Find out which balloons are the popular ones! * Pick one with a nice, clean solution that you are totally convinced will work to do first.

1.2 Before Designing Your Solution

* Highlight the important information on the problem statement - input bounds, special rules, formatting, etc. * Look for code in this notebook that you can use! * Convince yourself that your algorithm will run with time to spare on the biggest input. * Create several test cases that you will use, especially for special or boundary cases.

1.3 Prior to Submitting

* Check maximum input, zero input, and other degenerate test cases. * Cross check with team mates' supplementary test cases. * Read the problem output specification one more time - your program's output behaviour is fresh in your mind. * Does your program work with negative numbers? * Make sure that your program is reading from an appropriate input file. * Check all variable initialisation, array bounds, and loop variables (i vs j, m vs n, etc.). * Finally, run a diff on the provided sample output and your program's output. * And don't forget to submit your solution under the correct problem number!

1.4 After Submitting

* Immediately print a copy of your source. * Staple the solution to the problem statement and keep them safe. Do not lose them!

1.5 If It Doesn't Work...

* Remember that a run-time error can be division by zero. * If the solution is not complex, allow a team mate to start the problem afresh. * Don't waste a lot of time - it's not shameful to simply give up!!!

2 Ad-hoc Codes

2.1 Mod Functions

```
using ll = long long;

#define MOD 1000000009

inline void normal(ll &a) {if(a>=MOD)a %= MOD; (a < 0) && (a += MOD); }
inline ll modMul(ll a, ll b) {normal(a), normal(b); return (a*b)%MOD; }
inline ll modAdd(ll a, ll b) {normal(a), normal(b); return (a+b)%MOD; }
inline ll modSub(ll a, ll b) {normal(a), normal(b); a -= b; normal(a); return a; }
inline ll modPow(ll b, ll p) { ll r = 1; while(p) { if(p&1) r = modMul(r, b); b = modMul(b, b);
p >>= 1; } return r; }
inline ll modInverse(ll a) { return modPow(a, MOD-2); }
inline ll modDiv(ll a, ll b) { return modMul(a, modInverse(b)); }
```

2.2 Peripheral Functions

```
#define rep(i, n) for(int i = 0; i < n; ++i)
#define REP(i, n) for(int i = 1; i <= n; ++i)

inline bool EQ(double a, double b) { return fabs(a-b) < 1e-9; }
inline bool isLeapYear(ll year) { return (year%400==0) || (year%4==0 && year%100!=0); }
inline bool isInside(pii p,ll n,ll m) { return (p.first>=0&&p.first<n&&p.second>=0&&p.second<m); }
inline bool isInside(pii p,ll n) { return (p.first>=0&&p.first<n&&p.second>=0&&p.second<n); }
inline bool isSquare(ll x) { ll s = sqrt(x); return (s*s==x); }
inline bool isFib(ll x) { return isSquare(5*x*x+4)|| isSquare(5*x*x-4); }
inline bool isPowerOfTwo(ll x) { return ((1ll<<(ll)log2(x))==x); }
inline ll gcd(ll a, ll b) {return __gcd(a, b);}
inline ll lcm(ll a, ll b) {return (a * (b / gcd(a, b))); }
```

2.3 Matrix Power

```
struct mat {
    ll a[3][3];
    mat() { mem(a, 0); }
    mat operator * (const mat &b) const {
    mat ret;
rep(i, 3) rep(j, 3) rep(k, 3)
ret.a[i][j] = add(ret.a[i][j], mult(a[i][k], b.a[k][j]));
return ret; }
};

mat power(mat a, ll b) {
    mat ret;
rep(i, 3) rep(j, 3) ret.a[i][i] = 1;
    while(b) {
        if(b&1) ret = ret*a;
        b >>= 1 ;
        a = a*a;
    }
    return ret;
}
```

3 Number Theory

3.1 Catalan Numbers

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}, n \geq 0$$

3.2 NOD-SOD

Let $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, then, $NOD(n) = (a_1+1)(a_2+1)\cdots(a_k+1)$ and $SOD = \frac{p_1^{a_1+1}-1}{p_1-1} \cdot \frac{p_2^{a_2+1}-1}{p_2-1} \cdot \ldots \cdot \frac{p_k^{a_k+1}-1}{p_k-1}$

3.3 Totient Function

- If p is a prime number, then $gcd(p,q) = 1$ for all $1 \leq q < p$. Therefore we have: $\phi(p) = p - 1$.
- If p is a prime number and $k \geq 1$, then there are exactly p^k/p numbers between 1 and p^k that are divisible by p. Which gives us: $\phi(p^k) = p^k - p^{k-1}$.
- If a and b are relatively prime, then: $\phi(ab) = \phi(a) \cdot \phi(b)$.
- In general, for not co-prime a and b, the equation $\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$ with $d = \gcd(a,b)$ holds.
- Sum of co-primes of a number n is $\frac{n \cdot \phi(n)}{2}$.

3.4 Sieve Phi

```
#define mx 1000006
bitset <mx> mark;
int phi[mx];
void sievePhi() {
    for (int i = 1; i < mx; i++) ph[i] = i;
    phi[1] = 1, mark[1] = 1;
    for (int i = 2; i < mx; i++) {
        if (mark[i]) continue;
        for (int j = i; j < mx; j += i) {
            mark[j] = 1;
            phi[j] = phi[j] / i * (i - 1);
        }
    }
}
```

3.5 Loop Phi

```
int phi(int n) {
    int ret = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            ret -= ret / i;
        }
    }

    if (n > 1) { //there can be only one prime
    //gt sqrt(n) that divides n
        ret -= ret / n;
    }
    return ret;
}
```

3.6 Extended Euclid

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

bool find_any_solution(int a, int b, int c,
int &x0, int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

3.7 Miller-Rabin Primality Test

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n,u64 a,u64 d,int s){
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n, int iter=5) {
    if (n < 4)
        return n == 2 || n == 3;

    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }

    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n - 3);
        if (check_composite(n, a, d, s))
            return false;
    }
    return true;
}

bool MillerRabinDeterministic(u64 n) {
    if (n < 2)
        return false;
    int r = 0; u64 d = n - 1;
    while ((d & 1) == 0) {d >>= 1; r++; }
    vector <int> v32 = {2, 3, 5, 7};
    vector <int> v64 = {2, 3, 5, 7, 11, 13,
    17, 19, 23, 29, 31, 37};
    for (int a : v64) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true; }
```

3.8 FFT

```
using ll = long long;
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    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]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}

vector<ll> multiply(vector<ll> const& a,
vector<ll> const& b) {
    vector<cd> fa(a.begin(), a.end());
    vector<cd> fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
```

```
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<ll> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

3.9 Applications of Catalan Numbers

- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize $n+1$ factors.
- The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the $2n$ points on a circle to form n disjoint chords.
- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point $(0,0)$ to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting $(0,0)$ to (n,n)).
- Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$).
- The number of non-crossing partitions of a set of n elements.
- The number of ways to cover the ladder $1...n$ using n rectangles (The ladder consists of n columns, where ith column has a height i).

4 Data Structures

4.1 DSU

```
int find_set(int x) {
    if (p[x] == x) return x;
    return p[x] = find_set(p[x]);
}
void merge(int u, int v) {
    u = find_set(u), v = find_set(v);
    if (u == v) continue;
    if (st[u].size()>st[v].size())swap(u, v);
    for (auto x : st[u]) st[v].insert(x);
    par[u] = v;
}
```

4.2 BIT

```
const int M = 1000005;
int  bit[M+2];
///set a[idx]+=val;
void update(int idx,int  val){
    while(idx < M){
        bit[idx] += val;
        idx += (idx&-idx);
    }}
///returns the prefix sum from 0 to idx
int  qry(int idx){
    int  ret = 0;
    while(idx > 0){
        ret += bit[idx];
        idx -= (idx&-idx);
    }
    return ret;}
```

4.3 Mo's Algorithm

```
/** * MO's algorithm
* Handles offline query in O(Q \sqrt{N})
* Maintain proper block_sz ~ \sqrt{N}
* Careful with < in query
* Query indices are presumed to be 0-indexed
* Array indices are also 0-indexed**/

const int block_sz = 550; // N ~ 3e5
int freq[N], mo_cnt = 0;
int ret[N];

inline void add(int idx) {
    ++freq[a[idx]];
    if(freq[a[idx]] == 1) ++mo_cnt;}

inline void erase(int idx) {
    --freq[a[idx]];
    if(freq[a[idx]] == 0) --mo_cnt;}
inline int get_ans() {return mo_cnt;}
struct query {
    int l, r, idx;
    query() { }
query(int _l, int _r, int _i) : l(_l), r(_r), idx(_i) {}
    bool operator < (const query &p) const {
     if(l/block_sz != p.l/block_sz) return l < p.l;
     return ((l/block_sz) & 1) ? r > p.r : r < p.r;
    }};
void mo(vector<query> &q) {
    sort(q.begin(), q.end());
    memset(ret, -1, sizeof ret);
    // l = 1, r = 0 if 1-indexed array
    int l = 0, r = -1;
    for(auto &qq : q) {
        while(qq.l < l) add(--l);
        while(qq.r > r) add(++r);
        while(qq.l > l) erase(l++);
        while(qq.r < r) erase(r--);
        ret[qq.idx] = max(ret[qq.idx], get_ans());
}}
```

4.4 Order Statistics Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,
rb_tree_tag,
tree_order_statistics_node_update> ordered_set1;
typedef tree<int,null_type,greater<int>,
rb_tree_tag,
tree_order_statistics_node_update> ordered_set2;
long long int n,a[1000009];
/// order_of_key(x) returns number of elements
//strictly less than x
/// find_by_order(x) return (x-1)th largest element
ordered_set1 r;
ordered_set2 l;
main(){
   cin>>n;
   for(int i=0;i<n;i++)
   {
       scanf("%lld",&a[i]);
       r.insert(a[i]);
   }
  long long  int ans=0;
   for(int i=0;i<n;i++) {
       r.erase(a[i]);
       ans+=1LL*r.order_of_key(a[i])
       *1LL*l.order_of_key(a[i]);
       l.insert(a[i]);
   }
   cout<<ans<<endl;}
```

5 Graph

5.1 LCA

```
#define mx 1003
int n;
int T[mx];
int L[mx];
int P[mx][22];
bitset <mx> mark;
VI adj[mx];
VI sorted;

void top_sort(int u) {
    mark[u] = 1;
    for (auto v: adj[u]) {
        if (!mark[v]) {
            top_sort(v);
        }
    }
    sorted.push_back(u);
}

void dfs(int from, int u, int dep) {
    T[u] = from;
    L[u] = dep;

    for (auto v: adj[u]) {
        if (v == from) continue;
        dfs(u, v, dep + 1);
    }
}

void lca_init() {
    RESET(P, -1);
    for (int i = 1; i <= n; i++)
        P[i][0] = T[i];

    for (int j = 1; 1 << j <= n; j++) {
        for (int i = 1; i <= n; i++) {
        if (P[i][j - 1] != -1) {
        P[i][j] = P[P[i][j - 1]][j - 1];
        }
        }
    }
}

int lca_query(int p, int q) {
    if (L[p] < L[q]) swap(p, q);
    int log = 1;
    while (true) {
        int next = log + 1;
        if (1 << next > L[p]) break;
        log++;
    }

    for (int i = log; i >= 0; i--) {
        if (L[p] - (1 << i) >= L[q]) {
            p = P[p][i];
        }
    }

    if (p == q) return p;
    for (int i = log; i >= 0; i--) {
        if (P[p][i] != -1
            && P[p][i] != P[q][i]) {
            p = P[p][i], q = P[q][i];
        }
    }

    return T[p];
}
```

5.2 HLD: point update, Range Sum

```
const int mx=30005;
// maximum number of nodes of a tree
vector<int>adj[mx];
int arr[mx]; //this array will store
//the current node value
//a[idx]=node value at idx.
int n;
int par[mx],level[mx];
int max_subtree[mx];
int sparse_par[mx][17];
int chain_head[mx];
int chain_indx[mx];
int chain_size[mx];
int node_serial[mx]
int serial_node[mx];
int chain_no,indx;
ll tree[mx];
int dfs(int u, int from, int cnt){
    sparse_par[u][0]=from;
    level[u]=cnt;
    int node=-1, maxi=0;
    int total=1,sz=adj[u].size();
    for(int i=0; i<sz; i++)  {
        int v=adj[u][i];
        if(v!=from) {
            int temp=dfs(v,u,cnt+1);
            total+=temp;
            if(temp>maxi)
            {
                maxi=temp;
                node=v;
            }
        }  }
    max_subtree[u]=node;
    return total;  }
void build_table(int n){
    for(int j=1; 1<<j<=n; j++){
```

```
        for(int i=0; i<n; i++){
        sparse_par[i][j]=
        sparse_par[sparse_par[i][j-1]][j-1];
        }  }  }
int LCA_query(int p, int q){
    if(level[p]<=level[q]) swap(p,q);
    int log=log2(level[p]);
    for(int i=log; i>=0; i--){
        if(level[p]-(1<<i)>=level[q])
            p=sparse_par[p][i];
    }
    if(p==q) return p;
    for(int i=log; i>=0; i--) {
        if(sparse_par[p][i]!=sparse_par[q][i])
        {
            p=sparse_par[p][i];
            q=sparse_par[q][i];
        }    }
    return sparse_par[p][0];
}
void HLD(int u, int sz){
    if(chain_head[chain_no]==-1)
        chain_head[chain_no]=u;
    chain_indx[u]=chain_no;
    chain_size[chain_no]=sz;
    node_serial[u]=indx;
    serial_node[indx]=u;
    indx++;
    if(max_subtree[u]==-1) return ;
    HLD(max_subtree[u],sz+1);
    int len=adj[u].size();
    for(int i=0; i<len; i++){
        int v=adj[u][i];
        if(v!=sparse_par[u][0]
            && v!=max_subtree[u])
        {
            chain_no++;
            HLD(v,1);
        }
    }   }
void update(int idx, int val){
    while(idx<=indx) {
        tree[idx]+=val;
        idx+=(idx&-idx);
    }
}
ll query(int a, int b){
    ll ret=0;
    ll ret2=0;
    while(b) {
        ret+=tree[b];
        b-=(b & -b);
    }
    a--;
    while(a){
        ret2+=tree[a];
        a-=(a&-a);
    }
    return ret-ret2;
}
ll query_tree(int a, int b){
    ll ret=0;
    while(chain_indx[a]!=chain_indx[b]) {
        ret+=query(node_serial
            [chain_head[chain_indx[a]]],
            node_serial[a]);
        a=sparse_par[chain_head
            [chain_indx[a]]][0];
    }
    ret+=query(node_serial[b],
        node_serial[a]);
    return ret;
}
void update_tree(int a, int val){
    update(node_serial[a],arr[a]*-1);
    update(node_serial[a],val);
    arr[a]=val;
}
inline void allclear(int n){
    chain_no=1;
    indx=1;
    for(int i=0; i<=n; i++)
        adj[i].clear();
    memset(tree,0,sizeof(tree));
    memset(chain_head,-1,sizeof chain_head);
}

/*
* call alclear(n+2) to reset every thing
* take the graph input at adj vector
* dfs(0,0,1)  * build_table(n)  * HLD(0,1)
* for(int i=1;i<indx;i++)update(i,arr[serial_node[i]])
point updates * lca=LCA_query(l,r)
returns lca of node l and r
* sum of values from l to r = query_tree(l,lca)
+query_tree(r,lca)-arr[lca];
* update_tree(idx,val)
change node[idx]=val;
*/
```

5.3 Cut Node, Bridge

```
void dfsCut(int par, int u) {
    low[u] = dfstime[u] = ++cnt;
    for (auto v : adj[u]) {
        if (dfstime[v] == 0) {
            if (u == dfsroot) rc++;
            dfsCut(u, v);
            if (low[v] >= dfstime[u])
                cutnode[u] = true;
            if (low[v] > dfstime[u])
                brdg.emplace_back(u, v);
            low[u] = min(low[u], low[v]);
        } else if (v != par) {
        low[u] = min(low[u], dfstime[v]);
        }
    }
}

int main() {
    cnt = 0; cutnode.assign(n+2, 0);
    for (int i = 1; i <= n; i++) {
        if (num[i] > 0) continue;
        dfsroot = i; rc = 0;
        dfsCut(-1, i);
        cutnode[dfsroot] = (rc > 1);
    }
}
```

5.4 Tarjan SCC

```
void tarjanSCC(int u) {
    low[u] = dfstime[u] = ++cnt;
    S.push_back(u); mark[u] = 1;
    for (auto v : adj[u]) {
        if (dfstime[v] == 0)
            tarjanSCC(v);
        if (mark[v])
        low[u] = min(low[u], low[v]);
    }

    if (low[u] == dfstime[u]) {
        printf("SCC %d:", ++numSCC);
        while (true) {
            int v = S.back();
            S.pop_back(); mark[v] = 0;
            printf(" %d", v);
            if (u == v) break;
        } puts("");
    }
}

int main() {
    dfstime.assign(n + 2, 0);
    low.assign(n + 2, 0);
    mark = 0;
    cnt = numSCC = 0;
    for (int i = 1; i <= n; i++) {
        if (dfstime[i] > 0) continue;
        tarjanSCC(i);
    }
}
```

6 Geometry

6.1 Point

```
struct point_i {
int x, y;
point_i () { x = y = 0.0; }
point_i (int _x, int _y) { x = _x, y = _y;}
int normSq() {
    return sqr(x) + sqr(y);
}};

struct point {
double x, y;
point () { x = y = 0.0; }
point (double _x,double _y) {x=_x, y=_y;}
double normSq() {//same as dot product A.A
    return x*x + y*y;
}

bool operator < (point &a) const {
    if(fabs(x-a.x) > EPS) return x < a.x;
    return y < a.y; }

bool operator == (point a) const {
    return EQ(x, a.x) && EQ(y, a.y); }};
```

6.2 2D Vector

```
struct vec {
double x, y;
vec () { x = y = 0.0; }
vec (double _x, double _y)
    {x=_x, y=_y; }
vec (point a, point b)
    {x = b.x-a.x,y = b.y-a.y;}
vec operator + (const point &rhs) {
    vec tmp;
    tmp.x = x+rhs.x; tmp.y = y+rhs.y;
    return tmp; }

vec operator - (const point &rhs) {
vec tmp;tmp.x = x-rhs.x; tmp.y = y-rhs.y;
return tmp; }

vec operator * (const double &a) {
    vec tmp;
    tmp.x = x*a; tmp.y = y*a;
    return tmp; }

vec operator / (const double &a) {
    vec tmp;
    tmp.x = x/a; tmp.y = y/a;
    return tmp; }

double operator * (const vec &rhs)
    { return x*rhs.x + y*rhs.y; }//dot pro
double operator ^ (const vec &rhs)
    { return x*rhs.y - y*rhs.x; }//crs pro
};
```

6.3 Line

```
struct line {
double a, b, c;
line () { a = b = c = 0.0; }
line (point p1, point p2) {
    if(EQ(p1.x, p2.x)) { //vertical line
        a = 1.0, b = 0.0, c = -p1.x; return;
    }
    a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    b = 1.0;
    c = -(double) (a * p1.x) - p1.y; } };s
```

6.4 Operations

```
//distance between two points
double dist (point a, point b) {
    return hypot(a.x - b.x, a.y - b.y);}
//rotate the point CCW
point rotate (point p, double theta) {
    double rad = theta*PI/180;//degree to rad
    return point(p.x*cos(rad)-p.y*sin(rad),
    p.x * sin(rad) + p.y * cos(rad)); }
point rotate (point p, point c, double rad){
    p.x -= c.x, p.y -= c.y;
    return point(p.x*cos(rad)-p.y*sin(rad)+c.x,
        p.x*sin(rad)+p.y*cos(rad)+c.y);
}

bool areParallel (line l1, line l2) {
    return EQ(l1.a, l2.a) && EQ(l1.b, l2.b);
}

bool areSame (line l1, line l2) {
   return areParallel(l1, l2)
   && EQ(l1.c, l2.c);
}

bool lineIntersect (line l1, line l2,
    point &p){ //not segments
    if(areParallel(l1, l2)) return 0;
    p.x = (l2.b * l1.c - l1.b * l2.c)
    / (l2.a * l1.b - l1.a * l2.b);
    if(fabs(l1.b) > EPS)
        p.y = -(l1.a * p.x + l1.c);
    else p.y = -(l2.a * p.x + l2.c);
    return 1;}

vec scale(vec v, double s) {
    return vec(v.x * s, v.y * s);
```

```
}

point translate(point p, vec v) {
    return point(p.x + v.x, p.y + v.y);
}

vec perpendicular (vec v) {
    return vec(-(v.y), v.x);
}

double distToLine (point p,
    point a, point b, point &c) {
    //formula c = a + u*ab;
    vec ap(a, p), ab(a, b);
    double u = (ap*ab) / (ab*ab);
    c = translate(a, scale(ab, u));
    return dist(p, c); }

double distToLineSegment (point p,
    point a, point b, point &c) {
    vec ap(a, p), ab(a, b);
    double u = (ap*ab) / (ab*ab);
    if(u < 0.0) {
        c = a;
       return dist(p, a);
    }
    if(u > 1.0) {
        c = b;
        return dist(p, b);
    }
    return distToLine(p, a, b, c);
}

double angle (point a, point o
, point b){//returns AOB in rad
    vec oa(o, a), ob(o, b);
    return acos((oa*ob)
        / sqrt((oa*oa)*(ob*ob)));
}

//r is on which side of line pq
//returns 0 if co-linear
// > 0 if CCW, < 0 if CW
int direction( point p, point q, point r) {
    vec pq(p, q), pr(p, r);
    return (pq^pr);
}

bool onSegment(point a, point b
    , point p) {
    return min(a.x, b.x) <= p.x &&
    p.x <= max(a.x, b.x) &&
    min(a.y, b.y) <= p.y &&
    p.y <= max(a.y, b.y);
}
```

```
bool segmentIntersect(point a, point b,
    point c, point d) {
//return true if two segments intersect

    //two lines are AB and CD
    int d1 = direction(c, d, a);
    int d2 = direction(c, d, b);
    int d3 = direction(a, b, c);
    int d4 = direction(a, b, d);

//if they intersect
    if(d1*d2 < 0 && d3*d4 < 0)
        return 1;

if(d1 == 0 && onSegment(c, d, a)) return 1;
if(d2 == 0 && onSegment(c, d, b)) return 1;
if(d3 == 0 && onSegment(a, b, c)) return 1;
if(d4 == 0 && onSegment(a, b, d)) return 1;
    return 0;
}

double area2Dpolygon(int n,
    point a[]) {
     double area = 0;
    for(int i = 0; i+1 < n; ++i){
        area += a[i].x*a[i+1].y;
        area -= a[i].y*a[i+1].x; }
    area += a[2].x*a[0].y;
    area -= a[2].y*a[0].x;
return fabs(area)/2.0; }
```

6.5 Triangles and Circles

```
double perimeterTriangle(double a,
    double b, double c) {
    return a+b+c;
}

double areaTriangle(double a, double b,
 double c) {
    return sqrt (s *(s-a)*(s-b)*(s-c));
}

double rInCircle(double ab, double bc,
 double ca) {
//radius of inscribed circle in a triangle
return areaTriangle(ab, bc, ca)/
(0.5*perimeterTriangle(ab, bc, ca)); }


double rCircumCircle(double ab, double bc,
 double ca) {
return ab * bc * ca /
(4.0 * areaTriangle(ab, bc, ca)); }
double rCircumCircle(point a,
    point b, point c) {
```

```
    return rCircumCircle(dist(a, b),
        dist(b, c), dist(c, a));
}

point cCircumCircle(point a, point b,
    point c) {
    b.x -= a.x; b.y -= a.y; c.x -= a.x;
    c.y -= a.y;
    double d = 2.0*(b.x*c.y - b.y*c.x);
    double p = (c.y*(b.x*b.x + b.y*b.y) -
        b.y*(c.x*c.x + c.y*c.y))/d;
    double q = (b.x*(c.x*c.x + c.y*c.y) -
        c.x*(b.x*b.x + b.y*b.y))/d;
    return point(p+a.x, q+a.y);
}
```

6.6 Convex Hull

```
vector< point > ConvexHull(ll n,
    point ara[]){
    ll i, j, k;
    vector< point > cnvx(2*n);
    sort(ara, ara+n);
    for(i=0, k=0; i<n; ++i) {
        while(k>=2 && direction(cnvx[k-2]
            , cnvx[k-1], ara[i]) <= 0)
            k--;
        cnvx[k++]=ara[i];
    }
    for(i=n-2, j=k+1; i>=0; --i){
        while(k>=j && direction(cnvx[k-2]
            , cnvx[k-1], ara[i]) <= 0)
            k--;
        cnvx[k++]=ara[i];
    }
    cnvx.resize(k-1);
    return cnvx;}
```

6.7 Pick's Theorem

Given a certain lattice polygon with non-zero area.

We denote its area by S, the number of points with integer coordinates lying strictly inside the polygon by I and the number of points lying on polygon sides by B.

$$S = I + \frac{B}{2} - 1$$

B can be calculated using $GCD(\mid x_1 - x_2 \mid, \mid y_1 - y_2 \mid) + 1$

7 Strings

7.1 Trie

```
void dfs(int level,int p) {
if(!p) return;
    best = max(best,level*trie[p].pre);
    /* eikhane level holo
    koddur prefix matched. Ar pre holo
    odddur e koi ta string ase. */
    for(int i =0; i<4; i++) {
        dfs(level+1,trie[p].nxt[i]); }
}
```

7.2 Z-Algorithm

```
// z[i]=number of elements prefix such that
// suffix=prefix ; suffix starts from idx i
//Sample:
//"aaaaa" - [0,4,3,2,1]
//"aaabaab" - [0,2,1,0,2,1,0]
//"abacaba" - [0,0,1,0,3,0,1]
//z[0]=0 or full length of string
void zfunction(string &s) {
    ll n = s.size();
    z[0] = n;
//if you want that the whole string
// is a substring of itself.
    ll L = 0, R = 0;
    for (int i = 1; i < n; i++) {
        if (i > R) {
            L = R = i;
            while (R < n &&
                s[R-L] == s[R]) R++;
            z[i] = R-L; R--;
        }
        else {
            int k = i-L;
            if (z[k] < R-i+1) z[i] = z[k];
            else {
                L = i;
                while (R < n &&
                    s[R-L] == s[R]) R++;
                    z[i] = R-L; R--;
        }}}
}
```

7.3 Manacher's Algorithm

```
int n, d1[MX], d2[MX];
void manacher() {
int l = 0, r = -1;
rep(i, n) {
     int k = (i > r ? 1 :
        min(d1[l+r-i], r-i));
     while(i-k >= 0 && i+k < n
        && a[i-k] == a[i+k]) ++k;
     d1[i] = k--;
     if(i+k > r) l = i-k, r = i+k;
}}
l = 0, r = -1;
rep(i, n) {
    int k = (i > r? 0:
        min(d2[l+r-i+1], r-i+1))+1;
     while(i-k >= 0 && i+k-1 < n
        && a[i-k] == a[i+k-1]) ++k;
    d2[i] = --k;
    if(i+k-1 > r) l = i-k, r = i+k-1;
}
```