Team notebook

IUH.CopyPaste - May 30

March 24, 2022

Contents

1 Data Structure

1.1 DSU

```
int root(int v) {
    return par[v] < 0 ? v : (par[v] = root(par[v]));
}
void merge(int x, int y) {
    if ((x = root(x)) == (y = root(y)) {
        return;
    }
    if (par[y] < par[x]) {
        swap(x, y);
    }
    par[x] += par[y];
    par[y] = x;
}
```

1.2 Fenwick Tree 2D

```
struct FenwickTree2D {
    vector<vector<int>> bit;
    int n, m;

    // init(...) { ... }

    int sum(int x, int y) {
        int ret = 0;
        // i = (i & (i + 1)) - 1 build tu 0 -> n - 1
        for (int i = x; i >= 1; i -= i & -i)
            for (int j = y; j >= 1; j -= j & -j)
                ret += bit[i][j];
```

```
        return ret;
    }

    void add(int x, int y, int delta) {
        // i = i | (i + 1) build tu 0 -> n - 1
        for (int i = x; i <= n; i += i & -i )
            for (int j = y; j <= m; j = j & -j)
                bit[i][j] += delta;
    }
};
```

1.3 Mo Algorithm

```
// ch   : x, y > 0
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
  if (pow == 0) {
      return 0;
    }
    int hpow = 1 << (pow - 1);
    int seg = (x < hpow) ? ((y < hpow ) ? 0 : 3) : ((y < hpow) ? 1 : 2);
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    int64_t subSquareSize = int64_t(1) << (2 * pow - 2);
    int64_t ans = seg * subSquareSize;
    int64_t add = gilbertOrder(nx, ny, pow - 1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
    return ans;
}

struct Query {
    int l, r, idx;
    int64_t ord;
  inline void calcOrder() {
        ord = gilbertOrder(l, r, 21, 0);
    }
} q[N];

inline bool operator<(const Query &a, const Query &b) {
  return a.ord < b.ord;
}
```

1.4 Segment Tree 2D

```
// 4 * n * 4 * m
void build_y (int vx, int lx, int rx, int vy, int ly, int ry) {
    if (ly == ry)
        if (lx == rx)
            t[vx][vy] = a[lx][ly];
        else
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    else {
        int my = (ly + ry) / 2;
        build_y (vx, lx, rx, vy*2, ly, my);
        build_y (vx, lx, rx, vy*2+1, my+1, ry);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}

void build_x (int vx, int lx, int rx) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        build_x (vx*2, lx, mx);
        build_x (vx*2+1, mx+1, rx);
    }
    build_y (vx, lx, rx, 1, 0, m-1);
}

int sum_y (int vx, int vy, int tly, int try_, int ly, int ry) {
    if (ly > ry)
        return 0;
    if (ly == tly && try_ == ry)
        return t[vx][vy];
    int tmy = (tly + try_) / 2;
    return sum_y (vx, vy*2, tly, tmy, ly, min(ry,tmy))
        + sum_y (vx, vy*2+1, tmy+1, try_, max(ly,tmy+1), ry);
}

int sum_x (int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
    if (lx > rx)
        return 0;
    if (lx == tlx && trx == rx)
        return sum_y (vx, 1, 0, m-1, ly, ry);
    int tmx = (tlx + trx) / 2;
    return sum_x (vx*2, tlx, tmx, lx, min(rx,tmx), ly, ry)
        + sum_x (vx*2+1, tmx+1, trx, max(lx,tmx+1), rx, ly, ry);
}

void update_y (int vx, int lx, int rx, int vy, int ly, int ry, int x, int y, int
     new_val) {
    if (ly == ry) {
        if (lx == rx)
            t[vx][vy] = new_val;
        else
            t[vx][vy] = t[vx*2][vy] + t[vx*2+1][vy];
    }
    else {
        int my = (ly + ry) / 2;
        if (y <= my)
            update_y (vx, lx, rx, vy*2, ly, my, x, y, new_val);
```

```
        else
            update_y (vx, lx, rx, vy*2+1, my+1, ry, x, y, new_val);
        t[vx][vy] = t[vx][vy*2] + t[vx][vy*2+1];
    }
}

void update_x (int vx, int lx, int rx, int x, int y, int new_val) {
    if (lx != rx) {
        int mx = (lx + rx) / 2;
        if (x <= mx)
            update_x (vx*2, lx, mx, x, y, new_val);
        else
            update_x (vx*2+1, mx+1, rx, x, y, new_val);
    }
    update_y (vx, lx, rx, 1, 0, m-1, x, y, new_val);
}
build_x(1, 0, n - 1);
// get tu hang lx -> rx, tu cot ly -> ry
sum_x(1, 0, n - 1, lx, rx, ly, ry) << '\n';
update_x(1, 0, n - 1, x, y, v);
```

1.5 Segment Tree Lazy

```
struct Node {
    int lazy;
    int val;
} node[N * 4];

void down(int id) {
    int t = node[id].lazy;
    node[id * 2].lazy += t;
    node[id * 2 + 1].lazy += t;

    node[id * 2].val += t;
    node[id * 2 + 1].val += t;

    node[id].lazy = 0;
}

void build(int id, int l, int r) {
    if (l == r) {
        node[id].val = a[l];
        return;
    }
    int m = (l + r) >> 1;
    build(id * 2, l, m);
    build(id * 2 + 1, m + 1, r);
    node[id].val = max(node[id * 2].val, node[id * 2 + 1].val);
}

void update(int id, int l, int r, int u, int v, int val) {
    if (v < l || r < u) {
        return;
    }
    if (u <= l && r <= v) {
        node[id].val += val;
        node[id].lazy += val;
        return;
    }
    int m = (l + r) >> 1;
    down(id);
    update(id * 2, l, m, u, v, val);
    update(id * 2 + 1, m + 1, r, u, v, val);
    node[id].val = max(node[id * 2].val, node[id * 2 + 1].val);
}

int get(int id, int l, int r, int u, int v) {
    if (v < l || r < u) {
        return -1e9;
    }
    if (u <= l && r <= v) {
        return node[id].val;
    }
    int m = (l + r) >> 1;
    down(id);
    return max(get(id * 2, l, m, u, v), get(id * 2 + 1, m + 1, r, u, v));
}
```

1.6 Sparse Table

```
const int N = 1e5 + 11;
const int K = 20;

int lg[N], st[N][K], n, a[N];

lg[1] = 0;
for (int i = 2; i < N; ++i) lg[i] = lg[i / 2] + 1;

for (int i = 0; i < n; ++i) st[i][0] = a[i];

for (int j = 1; j < K; ++j) {
    for (int i = 0; i + (1 << j) <= n; ++i) {
        st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
}
// get min tu l -> r
int j = lg[r - l + 1];
cout << min(st[l][j], st[r - (1 << j) + 1][j]) << '\n';
```

1.7 Splay Tree

```cpp
template<typename T> struct Node {
	Node<T> *p, *ch[2];
	T value;
	int size;
	Node() {
		SET(value, 0);
		p = ch[0] = ch[1] = NULL;
		size = 1;
	}
	Node(T v) {
		value = v;
		p = ch[0] = ch[1] = NULL;
		size = 1;
	}
};
template<typename T> inline int getSize(Node<T> *node) {
	return (node ? node->size : 0);
}
template<typename T> struct SplayTree {
	Node<T> *root;
	SplayTree() {
		root = NULL;
	}
	void dig(Node<T> *node) {
		if (node) {
			dig(node->ch[0]);
			cout << node->value << " ";
			dig(node->ch[1]);
		}
	}
	void dump(Node<T> *node = NULL) {
		Node<T> *w = (node ? node : root);
		dig(w);
		cout << endl;
	}
	void rotate(Node<T> *node, int b) {
		Node<T> *c = node->ch[b], *cc = (c ? c->ch[1 - b] : NULL);
		if (!c) return;

		Node<T> *dad = node->p;
		if (!dad) {
			root = c;
		} else {
			dad->ch[0] == node ? dad->ch[0] = c : dad->ch[1] = c;
		}

		node->size += getSize(cc) - getSize(c);
		c->size += getSize(node) - getSize(cc);

		c->p = dad;
		node->p = c;
		c->ch[1 - b] = node;
		node->ch[b] = cc;
		if (cc) cc->p = node;
	}
	void splay(Node<T> *node) {
		while (node->p != NULL) {
			Node<T> *dad = node->p, *gra = (dad ? dad->p : NULL);
			int a = (node == dad->ch[1]);
			if (!gra) {
				rotate(dad, a);
			} else {
				int b = (dad == gra->ch[1]);
				if (a == b) {
					rotate(gra, a);
					rotate(dad, a);
				} else {
					rotate(dad, a);
					rotate(gra, b);
				}
			}
		}
		root = node;
	}
	Node<T> *find(int index, Node<T> *where = NULL) {
		Node<T> *w = (where ? where : root);
		if (index > getSize(w)) {
			return NULL;
		}

		int leftSize = getSize(w->ch[0]);
		if (index <= leftSize) {
			return find(index, w->ch[0]);
		}
		index -= leftSize + 1;
		return (index ? find(index, w->ch[1]) : w);
	}
	void insert(T value, int index, Node<T> *where = NULL) {
		Node<T> *w = (where ? where : root);
		Node<T> *newNode = new Node<T>(value);
		if (index > getSize(w)) {
			if (root == NULL){
				root = newNode;
				return;
			}

			Node<T> *pos = root;
			for (; pos->ch[1] != NULL; pos = pos->ch[1]);
			newNode->p = pos;
			for (Node<T> *x = pos; x != NULL; x->size++, x = x->p);
			splay(pos->ch[1] = newNode);
			return;
		}
```

```
        Node<T> *pos = find(index, w);
        if (!pos->ch[0]) {
            newNode->p = pos;
            for (Node<T> *x = pos; x != NULL; x->size++, x = x->p);
            splay(pos->ch[0] = newNode);
        } else {
            for (pos = pos->ch[0]; pos->ch[1] != NULL; pos = pos->ch[1]);
            newNode->p = pos;
            for (Node<T> *x = pos; x != NULL; x->size++, x = x->p);
            splay(pos->ch[1] = newNode);
        }
    }
    void erase(Node<T> *node) {
        int v[2] = {node->ch[0] != NULL, node->ch[1] != NULL};
        Node<T> *dad = node->p;
        if (!v[0] && !v[1]){
            if (dad){
                if (dad->ch[0] == node) {
                    dad->ch[0] = NULL;
                } else {
                    dad->ch[1] = NULL;
                }
                for (Node<T> *x = dad; x != NULL; x->size--, x = x->p);
                splay(dad);
            } else {
                root = NULL;
            }
            delete node;
            return;
        }
        Rep(i, 2) {
            if (!v[i]) continue;
            Node<T> *pos = node->ch[i], *uNode;
            for (; pos->ch[1 - i] != NULL; pos = pos->ch[1 - i]);
            node->value = pos->value;
            if (pos == node->ch[i]){
                node->ch[i] = pos->ch[i];
                uNode = node;
            } else {
                uNode = pos->p;
                uNode->ch[1 - i] = pos->ch[i];
            }
            if (pos->ch[i]) (pos->ch[i])->p = uNode;
            delete pos;
            for (pos = uNode; pos != NULL; pos->size--, pos = pos->p);
            if (dad) splay(dad);
            break;
        }
    }
};
```

1.8 Treap

```
struct treap {
    int key, prio, size;
    treap *l, *r;
    treap() {
        key = prio = 0; size = 1;
        l = r = 0;
    }
    treap(int _key, int _prio, treap* _l, treap* _r) {
        key = _key; prio = _prio; size = 1;
        l = _l; r = _r;
        if (l) size += l->size;
        if (r) size += r->size;
    }
};
int seed;
inline int randPrio() {
    seed = (seed * 1001 + 100621) % 999983;
    return seed;
}
inline int size(treap* &t) {
    return t ? t->size : 0;
}
inline int prio(treap* &t) {
    return t ? t->prio : -1;
}
inline void upd(treap* &t) {
    t->size = size(t->l) + size(t->r) + 1;
}
treap* search(treap* &t, int pos) {
    if (pos == size(t->l) + 1) return t;
    if (pos <= size(t->l)) return search(t->l, pos);
    return search(t->r, pos - size(t->l) - 1);
}
inline void rotl(treap* &t) {
    treap* l = t->l; t->l = l->r; l->r = t; t = l;
    upd(t->r); upd(t);
}
inline void rotr(treap* &t) {
    treap* r = t->r; t->r = r->l; r->l = t; t = r;
    upd(t->l); upd(t);
}
inline void balance(treap* &t) {
    if (prio(t->l) > t->prio) rotl(t);
        else if (prio(t->r) > t->prio)
            rotr(t);
}
void insert(treap* &t, int key, int prio, int pos) {
    if (!t) {
        t = new treap(key, prio, 0, 0);
        return;
```

```
	}
	t->size++;
	if (pos > size(t))
		insert(t->r, key, prio, pos);
	else if (pos <= size(t->l) + 1)
		insert(t->l, key, prio, pos);
	else insert(t->r, key, prio, pos - size(t->l) - 1);
	balance(t);
}
treap* merge(treap* l, treap* r) {
	if (!l) return r;
	if (!r) return l;
	if (l->prio < r->prio) {
		treap* t = merge(l, r->l);
		return new treap(r->key, r->prio, t, r->r);
	} else {
		treap* t = merge(l->r, r);
		return new treap(l->key, l->prio, l->l, t);
	}
}
void erase(treap* &t, int pos) {
	if (pos == size(t->l) + 1) {
		if (!t->l && !t->r) {
			delete t;
			t = 0;
		} else
			t = merge(t->l, t->r);
		return;
	}
	t->size--;
	if (pos <= size(t->l)) erase(t->l, pos);
		else erase(t->r, pos - size(t->l) - 1);
}

inline void split(treap* t, treap* &l, treap* &r, int pos) {
	insert(t, -1, INF, pos);
	l = t->l; r = t->r;
	delete t;
}

void dump(treap* node) {
	if (node) {
		dump(node->l);
		/* TODO */
		dump(node->r);
	}
}
```

2 Graph

2.1 2SAT

```
int n;
vector<vector<int>> adj, adj_t;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    used[v] = true;
    for (int u : adj[v]) {
        if (!used[u])
            dfs1(u);
    }
    order.push_back(v);
}

void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : adj_t[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
}

bool solve_2SAT() {
    order.clear();
    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!used[i])
            dfs1(i);
    }

    comp.assign(n, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1)
            dfs2(v, j++);
    }

    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
}

void add_disjunction(int a, bool na, int b, bool nb) {
```

```
    // na and nb signify whether a and b are to be negated
    a = 2*a ^ na;
    b = 2*b ^ nb;
    int neg_a = a ^ 1;
    int neg_b = b ^ 1;
    adj[neg_a].push_back(b);
    adj[neg_b].push_back(a);
    adj_t[b].push_back(neg_a);
    adj_t[a].push_back(neg_b);
}
```

2.2 Bridges Cut Point SCC

```
int n; // number of nodes

vector<int> adj[MAXN]; // adjacency list of graph

bool visited[MAXN];
int tin[MAXN], low[MAXN];
int timer;
stack <int> st_vex;

vector <pair <int, int>> bridges;
void IS_BRIDGE(int v,int to){
        bridges.push_back({v,to});
}

vector <int> cut_points;
void IS_CUTPOINT(int v){
        cut_points.push_back(v);
}

vector <vector <int>> sccs;
void IS_SCC(int v){
        int u;
        sccs.push_back(vector<int>());
        do {
                u = st_vex.top();
        tin[u] = low[u] = 1e9;
        sccs.back().push_back(u);
        st_vex.pop();
    } while (u != v);
}

void dfs(int v, int p = -1) {
    visited[v] = true;
    st_vex.push(v);
    tin[v] = low[v] = timer++;
    int child=0;

    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);

            if (low[to] > tin[v])
                IS_BRIDGE(v, to);

            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++child;
        }
    }
    if(p == -1 && child > 1)
        IS_CUTPOINT(v);

    if (tin[v] == low[v])
        IS_SCC(v);
}

void find() {
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

2.3 Dinic

```
const int N = 1003, oo = 0x3c3c3c3c;
int n, m, S, T;
int d[N], c[N][N], f[N][N];
int Dfs[N], t = 0;
vector<int> a[N];

bool bfs(int S, int T) {
    memset(d, 0, sizeof d);
    queue<int> qu;
    qu.push(S);
    d[S] = 1;
    while (qu.size()) {
        int u = qu.front();
        qu.pop();
        if (u == T)
            return true;
        for (int v : a[u])
            if (!d[v] && f[u][v] < c[u][v]) {
                qu.push(v);
                d[v] = d[u] + 1;
```

```
            }
        }
    return false;
}

int visit(int u, int Min) {
    if (u == T)
        return Min;
    if (Dfs[u] != t)
        Dfs[u] = t;
    else
        return 0;

    for (int v : a[u])
        if (f[u][v] < c[u][v])
            if (Dfs[v] != t && d[v] == d[u] + 1)
                if (int x = visit(v, min(Min, c[u][v] - f[u][v]))) {
                    f[u][v] += x;
                    f[v][u] -= x;
                    return x;
                }
    return 0;
}

main() {
    cin >> n >> m >> S >> T;
    for (int i = 1; i <= m; i++) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        a[x].push_back(y);
        a[y].push_back(x);
        c[x][y] += z;
    }
    int Sum = 0;
    while (bfs(S, T)) {
        while (int x = (t++, visit(S, oo))) {
            Sum += x;
            //printf("Sum=%d\n", Sum);
        }
    }
    cout << Sum << endl;
}
```

2.4 Maximum Bipartite Matching

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

int main() {
    // ... reading the graph ...

    mt.assign(k, -1);
    vector<bool> used1(n, false);
    for (int v = 0; v < n; ++v) {
        for (int to : g[v]) {
            if (mt[to] == -1) {
                mt[to] = v;
                used1[v] = true;
                break;
            }
        }
    }
    for (int v = 0; v < n; ++v) {
        if (used1[v])
            continue;
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

3 Math

3.1 Extend Euclid

```
int x, y;
int extend_gcd(int a, int b){
    if(b == 0){
        x = 1, y = 0;
        return a;
    }
    int g = extend_gcd(b, a%b);
    int tmp = x;
```

```
    x = y;
    y = tmp - (a/b)*y;
    return g;
}
```

3.2 FFT

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

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), 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<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

3.3 Inverse Modulo

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

int modulo_inverse_euclidean(int n, int m) {
    int x, y;
    if (gcd_extend(n, m, x, y) != 1) return -1; // not exist
    return (x % m + m) % m; // v x  c   th   m
}
//  m  nhanh
void binPow(ll &a, ll n, ll b, ll mod){
    if(n <= 1)return;
    binPow(a, n/2, b, mod);
    a = a*a;
    a %= mod;
    if(n % 2)a *= b;
    a %= mod;
}
// m  l   s   nguyn   t
ll nghichDaoModulo(ll a, ll mod){
    binPow(a, mod-2, a, mod);
    return a;
}
void multi(ll mod, ll n){
    ll inv[n+5];
    inv[1] = 1;
    for(ll i = 2; i <= n; ++i){
        inv[i] = mod - (mod/i) * inv[mod%i] % mod;
        cout<<inv[i]<<" ";
    }
}
```

3.4 Lehmer - Count Prime Less Than N

```
#define long long long
const int N = 100005;
const int M = 1000000007;
bool np[N];
int p[N], pp = 0;

void eratos() {
    np[0] = np[1] = true;
    for (int i = 2; i * i < N; i++)
        if (!np[i])
            for (int j = i * i; j < N; j += i) np[j] = true;
    for (int i = 2; i < N; i++)
        if (!np[i]) p[++pp] = i;
}

long power(long a, long k) {
    long P = 1;
    while (k) {
        if (k & 1) P = P * a;
        k /= 2;
        a = a * a;
    }
    return P;
}

long power(long a, long k, long M) {
    long P = 1;
    for (a = a % M; k; k /= 2) {
        if (k & 1) P = P * a % M;
        a = a * a % M;
    }
    return P;
}

long root(long n, long k) {
    long x = pow(n, 1.0 / k);
    while (power(x, k) % M == power(x, k, M) && power(x, k) < n) x++;
    while (power(x, k) % M != power(x, k, M) || power(x, k) > n) x--;
    return x;
}

map<long, long> Phi[N];

long phi(long x, int a) {
    if (Phi[a].count(x)) return Phi[a][x];
    if (a == 1) return (x + 1) / 2;
    long Result = phi(x, a - 1) - phi(x / p[a], a - 1);
    return Phi[a][x] = Result;
}

long pi(long x) {
    if (x < N)
        return upper_bound(p + 1, p + pp + 1, x) - (p + 1);
    long a = pi(root(x, 4));
    long b = pi(root(x, 2));
    long c = pi(root(x, 3));
    long Sum = phi(x, a) + (b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++)
        Sum -= pi(x / p[i]);
    for (int i = a + 1; i <= c; i++) {
        long bi = pi(root(x / p[i], 2));
        for (int j = i; j <= bi; j++)
            Sum -= pi(x / p[i] / p[j]) - (j - 1);
    }
    return Sum;
}
```

3.5 Miller Rabin

```
bool millerTest(ll n, ll d){
    ll a = random(2, n-2);
    ll x = (ll)binpow(a, d, n);
    // cout<<x<<" "<<d<<"\n";
    if(x == 1 || x == n-1)return true;
    while(d != n-1){
        x = x*x%n;
        d *= 2;
        // cout<<x<<" "<<d<<"\n";
        if(x == 1)return false;
        if(x == n-1)return true;
    }
    return false;
}

bool isPrime(ll n){
    if(n == 1 || n == 4)
        return false;
    if(n <= 3)return true;
    ll d = n-1;
    while(d % 2 == 0)
        d /= 2;
    ll r = log2((n-1)/d);
    // cout<<d<<" "<<r<<"\n";
    for(int i=0; i<10; i++){
        if(!millerTest(n, d))
            return false;
    }
    return true;
}
```

3.6 Phi Euler

```
int phi(int n) {
    int res = n;
    for(int i=2; i*i<=n; i++){
        if(n % i == 0){
            while(n % i == 0){
                n/=i;
            }
            res -= res/i;
        }
    }
    if(n != 1) // n l  s   nguyn   t
        res -= res/n;
    return res;
}
```

4 Misc

4.1 Integration

```
db integration(db a, db b){
        ll step = 1000000; // multiplied by 2
    db h = (b - a) / step;
    db s = f(a) + f(b);
    for (int i = 1; i <= step - 1; ++i) {
        db x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}
```

4.2 Ternary Search Float

```
db TernarySearchF(db l, db r) {
    while (r - l > 1e-9) {
        db m1 = l + (r - l) / 3;
                db m2 = r - (r - l) / 3;
        if (f(m1) < f(m2))
                        r = m2;
        else
                l = m1;
    }
    return l;
}
```

4.3 Ternary Search Int

```
ll TernarySearchI(ll l, ll r){
  while(l < r){
    ll m = (l+r)>>1;
    if (f(m) < f(m+1))
      r = m;
    else
      l = m+1;
  }
  return l;
}
```

5 String

5.1 KMP

```
void kmp_algo(string a, string b, vector<int> pre){

    vector<int> kmp(lena, 0);
    int j =0;
    for(int i=0; i<lena; i++){
        // cout<<i<<" "<<j<<" ";
        while(j >= 0){
            if(a[i] == b[j]){
                j++;
                break;
            }else{
                if(j == 0)break;
                j = pre[j-1];
            }
        }
        kmp[i] = j;
        if(kmp[i] == lenb){
            cout<<i-lenb+2<<" ";
        }
        if(j >= lenb)j = pre[j-1];
    }

}

void preProcess(string s, vector<int> &pre){
    int j = 0;
    for(int i=1; i<lenb; i++){

        while(j >= 0){
            if(s[i] == s[j]){
                pre[i] = ++j;
                break;
```

```
        }else{
            if(j == 0)break;
            j = pre[j-1];
        }
    }
}
// for(int i=0; i<n; i++)cout<<pre[i]<<" ";
}
```

5.2 Manacher

```
void manacher_algo(string s){
    int n = s.length();
    // center: center position
    // r: center right postion
    int center = 1, r = 2;
    vector<int> lps(n+5, 0);
    lps[1] = 1;
    // cr: current right
    for(int cr=2; cr<n; cr++){

        // cl: current left
        int cl = 2*center-cr;
        // khoang cach
        int d = r - cr;
        if(lps[cl] < d){
            lps[cr] = lps[cl];
        }else if(s[cr] != '|'){
            // tam la 1 ky tu
            int e = (2*d+1)/2;
            lps[cr] = min(lps[cl], e);
            while(1){

                int l = cr - lps[cr] - 1;
                int r = cr + lps[cr] + 1;
                if(l >= 0 && r < n && (s[l] == '|' || s[l] == s[r])){
                    lps[cr]++;
                }else break;
            }
        }else{
            // tam la 1 ky tu |
            int e = (2*d+1)/2;
            lps[cr] = min(lps[cl], e);
            while(1){

                int l = cr - lps[cr] - 1;
                int r = cr + lps[cr] + 1;
                if(l >= 0 && r < n && (s[l] == '|' || s[l] == s[r])){
                    lps[cr]+=2;
                }else break;
            }
        }
        // neu phan mo rong vuot qua r cua center
        if(cr + lps[cr] >= r){
            center = cr;
            r = cr+lps[cr];
        }
    }
    int nmax = 0, vt;
    for(int i=0; i<n; i++){
        if(nmax < lps[i]){
            nmax = lps[i];
            vt = i;
        }
    }
    cout<<nmax<<"\n";
    for(int i=vt-nmax; i<vt+nmax; i++)
        if(s[i] != '|')cout<<s[i];
}

string s = "|";
for(int i=0; i<n; i++){
    s += a[i];
    s += '|';
}
manacher_algo(s);
```

5.3 Suffix Automation

```
struct state {
    int len, link;
    map<char, int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
```

```
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

5.4 Trie

```
struct Node{
    map<char, Node*> childs;
    bool isLast;
};


Node* newNode(){
    Node* p = new Node;
    for(char c='a'; c<='z'; c++){
        p->childs[c] = NULL;
    }
    p->isLast = false;
    return p;
}

struct Node* root = newNode();
void insert(string a){
    Node* cur = root;
    for(int i=0; i<a.size(); i++){
        if(cur->childs[a[i]] == NULL){ // not exist
            cur->childs[a[i]] = newNode();
        }
        cur = cur->childs[a[i]];
    }
    cur->isLast = true;
}

int main(){
    for(int i=1; i<s.size(); i++){
        Node* cur = root;
        // max O(size(a))
        for(int j=i; j>=1; j--){
            if(cur->childs[s[j]] == NULL)
                break;

            // if(cur->childs[s[j]]->isLast)
                // do something
            cur = cur->childs[s[j]];
        }
    }
}
```

5.5 Z Algo

```
vector<int> z_algo(string s, int n) {
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i=1; i<n; i++) {
        if (z[i-l] < r-i+1){
            z[i] = z[i-l];
        }else {
            r = max(r, i);
            while (s[r-i] == s[r]){
                r += 1;
            }
            z[i] = r - i;
            r--;
            l = i;
        }
    }

    return z;
}
```

Thư viện hình học

```
int cmp(double a, double b) {
   if (abs(a - b) < EPS) {
      return 0;
   }
   return (a > b ? 1 : -1);
}

struct Point {
   double x, y;

   Point(double _x = 0, double _y = 0) {
      x = _x, y = _y;
   }

   bool operator == (const Point& that) const {
      return (cmp(x, that.x) == 0 && cmp(y, that.y) == 0);
   }

   bool operator < (const Point& that) const {
      if (cmp(x, that.x) != 0) {
         return cmp(x, that.x) < 0;
      }
      return cmp(y, that.y) < 0;
   }
};
```

Kiểm tra khi đi từ điểm p0 qua điểm p1 rồi đến điểm p2 thì góc rẽ ở p1 có ngược chiều kim đồng hồ hay không. Trả về:

- 0: Nếu ba điểm thẳng hàng
- 1: Nếu góc rẽ ở điểm p1 ngược chiều kim đồng hồ (rẽ trái)
- -1: Nếu góc rẽ ở điểm p1 xuôi chiều kim đồng hồ (rẽ phải)

```
int ccw(Point p0, Point p1, Point p2) {
   double dx1 = p1.x - p0.x, dy1 = p1.y - p0.y;
   double dx2 = p2.x - p0.x, dy2 = p2.y - p0.y;
   return cmp(dx1 * dy2 - dx2 * dy1, 0);
}
```

Trả về vị trí của điểm p0 so với đoạn thẳng có 2 đầu mút p1, p2:

- -2: Không xác định được đoạn thẳng vì điểm p1 và p2 trùng nhau
- -1: Không nằm trên đoạn thẳng nối bởi 2 điểm p1, p2
- 0: Nằm trên đoạn thẳng
- 1: Nằm trên đường thẳng nhưng ngoài đoạn, ở gần điểm nhỏ hơn trong 2 điểm p1, p2
- 2: Nằm trên đường thẳng nhưng ngoài đoạn, ở gần điểm lớn hơn trong 2 điểm p1, p2


```
int segmentPos(Point p0, Point p1, Point p2) {
   if (p1 == p2) {
      return -2;
   } else if (ccw(p0, p1, p2) != 0) {
      return -1;
   } else if (p2 < p1) {
      swap(p1, p2);
   }
     if (p0 < p1) {
      return 1;
   } else if (p2 < p0) {
      return 2;
   }
   return 0;
}

bool linePos(Point p0, Point p1, Point p2) {
   return (ccw(p0, p1, p2) == 0);
}

bool getLine(Point p0, Point p1, double& a, double& b, double& c) {
   if (p0 == p1) {
      return false;
   }
   a = p1.y - p0.y;
   b = p0.x - p1.x;
   c = -(a * p0.x + b * p0.y);
   return true;
}
```

Tìm giao điểm của 2 đường thẳng p0.p1 và p2.p3, giao điểm được trả về tại p4 (nếu có). Hàm trả về:

- -1: Nếu 2 đường thẳng đã cho trùng nhau
- 0: Nếu chúng song song
- 1: Nếu chúng cắt nhau tại điểm p4

```
int getIntersection(Point p0, Point p1, Point p2, Point p3, Point&
p4) {
   double a0, b0, c0, a1, b1, c1;
   getLine(p0, p1, a0, b0, c0);
   getLine(p2, p3, a1, b1, c1);
   double d = a0 * b1 - a1 * b0;
   double dx = b0 * c1 - b1 * c0;
   double dy = -(c1 * a0 - c0 * a1);
   if (cmp(d, 0) == 0) {
```

```
      return (cmp(dx, 0) == 0 && cmp(dy, 0) == 0 ? -1 : 0);
   }
   p4.x = dx / d;
   p4.y = dy / d;
   return 1;
}

double dist(Point p0, Point p1) {
   double dx = p1.x - p0.x, dy = p1.y - p0.y;
   return sqrt(dx * dx + dy * dy);
}

bool isSegmentCut(Point p0, Point p1, Point p2, Point p3) {
   if (ccw(p0, p1, p2) * ccw(p0, p1, p3) < 0 &&
         ccw(p2, p3, p0) * ccw(p2, p3, p1) < 0) {
      return true;
   }
   return (segmentPos(p0, p2, p3) == 0 ||
         segmentPos(p1, p2, p3) == 0 ||
         segmentPos(p2, p0, p1) == 0 ||
         segmentPos(p3, p0, p1) == 0 ? true : false);
}

Point p0;

bool degreeCmp(Point p1, Point p2) {
   int d = ccw(p0, p1, p2);
   if (d != 0) {
      return (d > 0);
   }
   return cmp(dist(p0, p1), dist(p0, p2)) < 0;
}
```

Tìm bao lồi của tập điểm p[] sử dụng thuật toán Graham Scan, O(N logN).
Sắp xếp lại tập điểm p[] và trả về số điểm thuộc bao lồi tại n, khi ấy các điểm thuộc bao lồi sẽ là n điểm đầu tiên (0..n - 1) của p[].

```
void grahamScan(Point p[], int& n) {
   if (n == 1) {
      return;
   }
   int j = 0;
   For(i, 1, n - 1) {
      if (cmp(p[j].y, p[i].y) > 0 ||
            (cmp(p[j].y, p[i].y) == 0 && cmp(p[j].x, p[i].x) < 0)) {
         j = i;
      }
   }
   swap(p[0], p[j]);
   p0 = p[0];
   sort(p + 1, p + n, degreeCmp);
   int k = 2;
   For(i, 2, n - 1) {
      while (k > 1 && ccw(p[i], p[k - 1], p[k - 2]) >= 0) {
         k--;
      }
      swap(p[k++], p[i]);
   }
   n = k;
}
```

Kiểm tra điểm p0 có nằm trong đa giác lồi hay không, O(N).

```
bool insideConvexPolygon(Point p0, Point p[], int n) {
   int k, l;
   if (n == 1) {
      return false;
   }
   if (n == 2) {
      if (ccw(p0, p[0], p[1]) == 0) {
         return (segmentPos(p0, p[0], p[1]) == 0);
      } else {
         return false;
      }
   }
   k = l = 0;
   Rep(i, n) {
      int j = ccw(p0, p[i], p[(i + 1) % n]);
      k = (j < 0) ? 1 : k;
      l = (j > 0) ? 1 : l;
   }
   return (k + l != 2);
}

bool notHaveCommonPointBetween2ConvexPolygons(Point p1[], int n1,
Point p2[], int n2) {
   Rep(i, n1) {
      if (insideConvexPolygon(p1[i], p2, n2)) {
         return false;
      }
   }
   Rep(i, n2) {
      if (insideConvexPolygon(p2[i], p1, n1)) {
         return false;
      }
   }
   Rep(i, n1) {
      Rep(j, n2) {
```

```
        if (isSegmentCut(p1[i], p1[(i + 1) % n1], p2[j], p2[(j + 1)
% n2])) {
            return false;
        }
      }
   }
   return true;
}
```

Tính diện tích đa giác bị giới hạn bởi {đường chéo nối từ đỉnh i đến đỉnh j} và {các cạnh của đa giác nhận được khi đi từ đỉnh i đến đỉnh j, theo chiều tăng dần của chỉ số các đỉnh}.

```
double diagonalLineArea(Point p[], int n, int i, int j) {
   double s = 0;
   for (int k = i; ; k = (k + 1) % n) {
      int l = (k == j) ? i : (k + 1) % n;
      s += (p[l].x - p[k].x) * (p[l].y + p[k].y);
      if (k == j) {
         break;
      }
   }
   return abs(s) / 2.0;
}
```

Kiểm tra hai đoạn thẳng có thực sự cắt nhau hay không (giao điểm không trùng với 4 điểm đầu mút).

```
bool isRealCut(Point p0, Point p1, Point p2, Point p3) {
   return ccw(p0, p1, p2) * ccw(p0, p1, p3) < 0
       && ccw(p2, p3, p0) * ccw(p2, p3, p1) < 0;
}
```

Kiểm tra đường chéo của một đa giác (không nhất thiết là đa giác lồi) có cắt cạnh nào của chính đa giác đó hay không.

```
bool isDiagonalLineCutPolygon(Point p[], int n, Point p0, Point p1)
{
   Rep(i, n) {
      int j = (i + 1) % n;
      if (isRealCut(p0, p1, p[i], p[j])) {
         return true;
      }
   }
   return false;
}
```

Kiểm tra đường chéo của một đa giác (không nhất thiết là đa giác lồi) có nằm trong đa giác đó hay không.

```
bool isDiagonalLineInsidePolygon(Point p[], int n, int i, int j) {
   if (isDiagonalLineCutPolygon(p, n, p[i], p[j])) {
      return false;
   }
   return cmp(-diagonalLineArea(p, n, 1, 0) + diagonalLineArea(p, n,
i, j)
         + diagonalLineArea(p, n, j, i), 0) == 0;
}
```

Kiểm tra điểm có nằm trong đa giác (không nhất thiết là đa giác lồi) hay không, O(N).

```
bool insidePolygon(Point p[], int n, Point p0) {
   Rep(i, n) {
      int j = (i + 1) % n;
      if (segmentPos(p0, p[i], p[j]) == 0) {
         return true;
      }
   }
   p0.y += EPS;
   int k = 0;
   Rep(i, n) {
      int j = (i + 1) % n;
      if ((p0.y - p[i].y) * (p0.y - p[j].y) < 0) {
         double a, b, c;
         getLine(p[i], p[j], a, b, c);
         double x1 = -(c + b * p0.y) / a;
         k += (cmp(x1, p0.x) >= 0);
      }
   }
   return (k % 2 > 0);
}

void add(Point p3[], int& n3, Point p0) {
   if (n3 > 0 && p0 == p3[n3 - 1])
      return;
   if (n3 > 1 && ccw(p3[n3 - 2], p3[n3 - 1], p0) == 0) {
      p3[n3 - 1] = p0;
      return;
   }
   p3[n3++] = p0;
}
```

Cho một đa giác không tự cắt (không nhất thiết là đa giác lồi). Dùng đường thẳng p1.p2 để phân chia đa giác đó thành nhiều phần. Sử dụng p[] và n cho cả 2 mục đích in/out. Kết quả trả về là n điểm đầu tiên (0..n-1) của p[].
Lưu ý: Đường thẳng p1.p2 không được đi qua p0.

```
void divide(Point p[], int& n, Point p0, Point p1, Point p2) {
```

```
   double a, b, c;
   getLine(p1, p2, a, b, c);
   int k;
   Rep(i, n) {
      if ((p[i].x * a + p[i].y * b + c) * (p0.x * a + p0.y * b + c)
> 0) {
         k = i;
         break;
      }
   }
   Point p3[1010];
   int n3 = 0;
   add(p3, n3, p[k]);
   For(l, 1, n) {
      int i = (k + l) % n;
      int j = (i + n - 1) % n;
      if ((p[j].x * a + p[j].y * b + c) * (p[i].x * a + p[i].y * b +
c) <= 0) {
         Point p4;
         if (getIntersection(p[j], p[i], p1, p2, p4) == 1) {
            add(p3, n3, p4);
         }
      }
      if (i != k) {
         if ((p[i].x * a + p[i].y * b + c) * (p0.x * a + p0.y * b +
c) > 0) {
            add(p3, n3, p[i]);
         }
      }
   }
   n = n3;
   Rep(i, n) {
      p[i] = p3[i];
   }
}
```

Tìm giao của 2 đa giác lồi.

```
void getIntersectionOf2ConvexPolygons(Point p1[], int n1, Point
p2[], int n2,
      Point p0[], int& n0) {
   n0 = 0;
   Rep(i, n1) {
      if (insideConvexPolygon(p1[i], p2, n2)) {
         p0[n0++] = p1[i];
      }
   }
   Rep(i, n2) {
      if (insideConvexPolygon(p2[i], p1, n1)) {
         p0[n0++] = p2[i];
         }
   }
   Rep(i, n1) {
      Rep(j, n2) {
         int k = (i + 1) % n1;
         int l = (j + 1) % n2;
         Point p3;
         getIntersection(p1[i], p1[k], p2[j], p2[l], p3);
         if (segmentPos(p3, p1[i], p1[k]) == 0) {
            if (segmentPos(p3, p2[j], p2[l]) == 0) {
               p0[n0++] = p3;
            }
         }
      }
   }
   grahamScan(p0, n0);
}
```

Tính góc giữa 2 vector khác 0 p0.p1 và p0.p2 (tính theo Radian)

```
double getAngle(Point p0, Point p1, Point p2) {
   double d1 = dist(p0, p1);
   double d2 = dist(p0, p2);
   return acos(((p1.x - p0.x) * (p2.x - p0.x) + (p1.y - p0.y) *
(p2.y - p0.y))
         / (d1 * d2));
}
```

Kiểm tra điểm p0 có nằm trong góc tạo bởi 2 tia p1.p2 và p1.p3 hay không.

```
bool insideAngle(Point p0, Point p1, Point p2, Point p3) {
   if (p0 == p1)
      return true;
   if (ccw(p0, p1, p2) * ccw(p0, p1, p3) > 0)
      return false;
   return (getAngle(p1, p0, p2) < PI + EPS && getAngle(p1, p0, p3) <
PI + EPS);
}
```

Kiểm tra điểm p0 có nằm trong tam giác p1.p2.p3 hay không.

```
bool insideTriangle(Point p0, Point p1, Point p2, Point p3) {
   if (segmentPos(p0, p1, p2) == 0) {
      return true;
   }
   if (segmentPos(p0, p2, p3) == 0) {
      return true;
   }
   if (segmentPos(p0, p1, p3) == 0) {
      return true;
   }
   return (ccw(p0, p1, p2) * ccw(p0, p1, p3) < 0
```

```
        && ccw(p0, p2, p1) * ccw(p0, p2, p3) < 0);
}
```

Kiểm tra điểm p0 có nằm trong đa giác lồi p[] hay không, O(logN).

```
bool insideConvexPolygonLogN(Point p0, Point p[], int n) {
   if (!insideAngle(p0, p[0], p[1], p[n - 1])) {
      return false;
   }
   int i = 1;
   int j = n - 1;
   while (j - i > 1) {
      int k = (i + j) >> 1;
      if (insideAngle(p0, p[0], p[i], p[k])) {
         j = k;
      } else {
         i = k;
      }
   }
   return insideTriangle(p0, p[0], p[i], p[j]);
}
```

Tìm giao của 2 đường tròn, các giao điểm được trả về trong vector vp. Ngoài ra hàm còn trả về giá trị là số lượng giao điểm (-1 nếu có vô số giao điểm – tức 2 đường tròn trùng nhau).

```
int intersectionsOf2Circles(Point p0, double r0, Point p1, double
r1, vector<Point>& vp) {
   vp.clear();
   double d = dist(p0, p1);
   if (d > r0 + r1) {
      return 0;
   }
   if (d < abs(r0 - r1)) {
      return 0;
   }
   if (cmp(r0 - r1, 0) == 0 && cmp(p0.x - p1.x, 0) == 0 && cmp(p0.y
- p1.y, 0) == 0) {
      return -1;
   }
   double a = (r0 * r0 - r1 * r1 + d * d) / 2 / d;
   double h = sqrt(r0 * r0 - a * a);
   Point p2(p0.x + a * (p1.x - p0.x) / d, p0.y + a * (p1.y - p0.y) /
d);
   vp.PB(Point(p2.x + h * (p1.y - p0.y) / d, p2.y - h * (p1.x -
p0.x) / d));
   if (cmp(h, 0)) {
      return 1;
   }
   vp.PB(Point(p2.x - h * (p1.y - p0.y) / d, p2.y + h * (p1.x -
p0.x) / d));
   return 2;
}
```

Tìm giao của đường thẳng và đường tròn, các giao điểm được trả về trong vector p, ngoài ra hàm trả về giá trị là số lượng giao điểm.

```
int intersectionOfCircleAndLine(Point p0, double r0, Point p1,
Point p2,
      vector<Point>& p) {
   p.clear();
   Point u = p2 - p1;
   Point proj = projection(p0 - p1, u) + p1;
   if ((proj - p0).length() < r0 - eps) {
      ld add = sqrt(r0 * r0 - (proj - p0) * (proj - p0));
      u = u * (add / u.length());
      p.pb(proj + u);
      p.pb(proj - u);
      return 2;
   } else if (abs((proj - p0).length() - r0) <= eps) {
      p.pb(proj);
      return 1;
   } else
      return 0;
}
```

Tìm tiếp điểm khi kẻ tiếp tuyến từ điểm p1 đến đường tròn (p0, r), các tiếp điểm được trả về trong vector p. Ngoài ra hàm còn trả về giá trị là số lượng tiếp điểm:

0: Nếu p1 nằm trong đường tròn

1: Nếu p1 nằm trên đường tròn

2: Nếu p1 nằm ngoài – tức có thể vẽ 2 tiếp tuyến tới đường tròn

```
int getLineFromPointToCircle(Point p0, double r, Point p1,
vector<Point>& p) {
   double d = dist(p0, p1);
   if (cmp(d, r) < 0) {
      return 0;
   }
   if (cmp(d, r) == 0) {
      return 1;
   }
   double t1 = atan2(p1.y - p0.y, p1.x - p0.x);
   double t0 = acos(r / d);
   p.PB(Point(p0.x + r * cos(t1 - t0), p0.y + r * sin(t1 - t0)));
   p.PB(Point(p0.x + r * cos(t1 + t0), p0.y + r * sin(t1 + t0)));
   return 2;
}
```

Lấy đường trung trực của đoạn thẳng p0.p1.

Nếu p0 và p1 trùng nhau thì không tồn tại đường trung trực, trả về FALSE, ngược lại trả về TRUE.

```
bool getCenterLine(Point p0, Point p1, double& a, double& b,
double& c) {
   if (p0 == p1) {
      return false;
   }
   a = p1.x - p0.x;
   b = p1.y - p0.y;
   Point p2;
   p2.x = (p0.x + p1.x) / 2;
   p2.y = (p0.y + p1.y) / 2;
   c = -(p2.x * a + p2.y * b);
   return true;
}
```

Lấy đường tròn đi qua 3 điểm. Trả về FALSE nếu không tồn tại – tức 3 điểm thẳng hàng, ngược lại trả về TRUE.

```
bool getCircle(Point p0, Point p1, Point p2, Point& p3, double& r)
{
   if (ccw(p0, p1, p2) == 0) {
      return false;
   }
   double a0, b0, c0, a1, b1, c1;
   getCenterLine(p0, p1, a0, b0, c0);
   getCenterLine(p0, p2, a1, b1, c1);
   double d = a0 * b1 - a1 * b0;
   double dx = b0 * c1 - b1 * c0;
   double dy = -(c1 * a0 - c0 * a1);
   p3.x = dx / d;
   p3.y = dy / d;
   r = dist(p3, p0);
   return true;
}
```

Tìm khoảng cách giữa 2 điểm gần nhau nhất trong tập điểm.

```
struct YComparator {
   bool operator () (const Point p0, const Point p1) const {
      if (p0.y != p1.y) {
         return p0.y < p1.y;
      }
      return p0.x < p1.x;
   }
};

double closestPairDist(Point a[], int n) {
   sort(a, a + n);

   set <Point, YComparator> b;
   int j = 0;
   double ret = 1E100;

   Rep(i, n) {
      while (a[i].x - a[j].x > ret) {
         b.erase(a[j++]);
      }

      Point c = a[i];
      c.y -= ret;
      VAR(lowerIt, b.lower_bound(c));
      c.y += 2 * ret;
      VAR(upperIt, b.upper_bound(c));

      for (VAR(it, lowerIt); it != upperIt; it++) {
         ret = min(ret, dist(a[i], *it));
      }
      b.insert(a[i]);
   }
   return ret;
}
```

3D convex hull

```
/*
 V - E + F = 2
 E <= 3V - 6
 F <= 2V - 4
 */

inline double det(double a, double b, double c, double d) {
   return a * d - b * c;
}
struct Point {
   double x, y, z;
   Point() {
   }
   Point(double x, double y, double z) :
      x(x), y(y), z(z) {
   }
   Point operator -(const Point &op) const {
      return Point(x - op.x, y - op.y, z - op.z);
   }
   double operator *(const Point &op) const {
```

```
      return x * op.x + y * op.y + z * op.z;
   }
   double length() {
      return sqrt(x * x + y * y + z * z);
   }
   Point operator %(const Point &op) const {
      return Point(det(y, z, op.y, op.z), -det(x, z, op.x, op.z),
det(x, y,
            op.x, op.y));
   }
};
typedef vector<int> Side;

inline int sign(double x) {
   return x < -eps ? -1 : x > eps ? 1 : 0;
}

vector<Point> arr;
vector<int> rnd;
set<int> used;

Side getFirstSide(vector<Point> &p) {
   int i1 = 0;
   Rep(i,sz(p)) {
      if (p[i].z < p[i1].z || (p[i].z == p[i1].z && p[i].x <
p[i1].x)
            || (p[i].z == p[i1].z && p[i].x == p[i1].x && p[i].y <
p[i1].y)) {
         i1 = i;
      }
   }
   int i2 = i1 == 0 ? 1 : 0;
   Rep(i,sz(p)) {
      if (i != i1) {
         Point zDir(0, 0, 1);
         double curCos = (p[i] - p[i1]) * zDir / (p[i] -
p[i1]).length();
         double bestCos = (p[i2] - p[i1]) * zDir / (p[i2] -
p[i1]).length();
         if (curCos < bestCos) {
            i2 = i;
         }
      }
   }
   int i3 = -1;
   int n = sz(p);
   Rep(ri,n) {
      int i = rnd[ri];
      if (i != i1 && i != i2) {
         Point norm = (p[i1] - p[i]) % (p[i2] - p[i]);
            bool sg[] = { 0, 0, 0 };
            Rep(t,n) {
               int j = rnd[t];
               sg[1 + sign((p[j] - p[i]) * norm)] = true;
               if (sg[0] && sg[2]) {
                  break;
               }
            }
            if (sg[0] ^ sg[2]) {
               i3 = i;
               if (!sg[0]) {
                  swap(i3, i2);
               }
               break;
            }
         }
      }
   vector<int> res;
   res.pb(i1);
   res.pb(i2);
   res.pb(i3);
   return res;
}

inline int getSideKey(int i, int j, int k) {
   int key = (i * 1000 + j) * 1000 + k;
   return key;
}
inline bool isUsed(int i, int j, int k) {
   return used.find(getSideKey(i, j, k)) != used.end();
}

inline double getAngle(const Point &n1, const Point &n2) {
   return atan2((n1 % n2).length(), n1 * n2);
}

inline double getNormsAngle(int i, int j, int k, int t,
vector<Point> &p) {
   Point n1 = (p[j] - p[i]) % (p[k] - p[i]);
   Point n2 = (p[t] - p[i]) % (p[j] - p[i]);
   return getAngle(n1, n2);
}

void dfs(int i, int j, int k, vector<Point> &p, vector<Side>
&sides) {
   if (i < j && i < k) {
      vector<int> side(3);
      side[0] = i;
      side[1] = j;
      side[2] = k;
```

```
      sides.pb(side);
   }
   int key = getSideKey(i, j, k);
   used.insert(key);
   int n = sz(p);
   if (!isUsed(j, k, i))
      dfs(j, k, i, p, sides);
   if (!isUsed(k, i, j))
      dfs(k, i, j, p, sides);

   int bestT = -1;
   double bestAngle = 1e20;
   Point curNorm = (p[j] - p[i]) % (p[k] - p[i]);
   Point dir = p[j] - p[i];
   Rep(t,n) {
      if (t != i && t != j && t != k) {
         Point newNorm = (p[t] - p[i]) % dir;
         double curAng = curNorm * newNorm / newNorm.length();
         if (bestT == -1 || curAng > bestAngle) {
            bestT = t;
            bestAngle = curAng;
         }
      }
   }
   if (!isUsed(i, bestT, j)) {
      dfs(i, bestT, j, p, sides);
   }
}
vector<Side> convexHull3d(vector<Point> p) {
   used.clear();
   rnd.resize(sz(p));
   Rep(i,sz(p))
      rnd[i] = i;
   random_shuffle(rnd.begin(), rnd.end());
   Side side0 = getFirstSide(p);
   vector<Side> sides;

   dfs(side0[0], side0[1], side0[2], p, sides);
   return sides;
}

/* eliminate conflict sides */
inline bool isEmpty(Point x, Point y, Point z) {
   return abs(x * Point(y.y * z.z - y.z * z.y, y.z * z.x - y.x *
z.z, y.x
         * z.y - y.y * z.x)) <= eps;
}
inline bool conflict(Side a, Side b) {
   Point x = arr[a[0]], y = arr[a[1]], z = arr[a[2]];
   Rep(i,3) {
      Point t = arr[b[i]];
      if (!isEmpty(x - t, y - t, z - t))
         return false;
   }
   return true;
}
vector<Side> eliminate(vector<Side> p) {
   vector<Side> res;
   vector<bool> fre;
   fre.resize(sz(p), true);
   Rep(i,sz(p)) {
      if (!fre[i])
         continue;
      res.pb(p[i]);
      For(j,i+1,sz(p) - 1)
         if (fre[j]) {
            if (conflict(p[i], p[j])) {
               fre[j] = false;
               res.back().insert(res.back().end(), p[j].begin(),
                     p[j].end());
            }
         }
   }
   Rep(i,sz(res)) {
      sort(res[i].begin(), res[i].end());
      res[i].resize(unique(res[i].begin(), res[i].end()) -
res[i].begin());
   }
   return res;
}
```

Some useful Functions

```
inline Point projection(Point v, Point u) {
   long double scalar = (v * u) / (u * u);
   u.x *= scalar;
   u.y *= scalar;
   u.z *= scalar;
   return u;
}
inline Point projection(Point p, Point a, Point b, Point c) {
   Point u = (b - a) % (c - a);
   Point v = p - a;
   long double scalar = (v * u) / (u * u);
   u.x *= scalar;
   u.y *= scalar;
   u.z *= scalar;
   return p - u;
}
inline long double dist(Point p, Point a, Point b) {
```

```
    p = p - a;
    Point proj = projection(p, b - a);
    return sqrt(p * p - proj * proj);
}
inline long double area(Point a, Point b, Point c) {
    long double h = dist(a, b, c);
    return (h * (b - c).length()) / 2;
}
inline long double volume(Point x, Point y, Point z) {
    Point base = Point(y.y * z.z - y.z * z.y, y.z * z.x - y.x * z.z,
y.x * z.y
           - y.y * z.x);
    return abs(x.x * base.x + x.y * base.y + x.z * base.z) / 3;
}
```

Số học

Dùng Extended Euclid để tìm nghiệm của phương trình ax + by = gcd(a, b). Giả sử kết quả trả về là (x_0, y_0), họ nghiệm của phương trình sẽ là $\left(x_0 + \frac{kb}{d}, y_0 - \frac{ka}{d}\right)$ với $k \in Z$. Phương trình tổng quát ax + by = d chỉ có nghiệm khi d chia hết cho gcd(a, b).
Hàm positiveEE() tìm nghiệm với a, b nguyên không âm, hàm extendedEuclid() tìm nghiệm với a, b nguyên tùy ý.

```
pll positiveEE(ll a, ll b) {
    if (b == 0) {
        return MP(1, 0);
    }
    pll ret = positiveEE(b, a % b);
    return MP(ret.S, ret.F - ret.S * (a / b));
}

pll extendedEuclid(ll a, ll b) {
    pll ret = positiveEE(abs(a), abs(b));
    return MP((a < 0 ? -ret.F : ret.F), (b < 0 ? -ret.S : ret.S));
}
```

Tính base^{exp} theo module MOD trong O(log exp).

```
ll fastPow(ll base, ll exp, ll MOD = MODULO){
    if (base == 0) {
        return 0;
    }
    ll ret = 1, bb = base;
    while (exp > 0) {
        if (exp & 1) {
            ret = mod(ret * bb, MOD);
        }
        exp >>= 1, bb = mod(bb * bb, MOD);
    }
    return ret;
}
```

Kiểm tra số nguyên tố bằng thuật toán Rabin-Miller, độ chính xác (1 – 0.25^{numTry}), numTry tối đa 20.

Trick: Với N vào khoảng <= 4E9 chỉ cần test với 3 số **2 7 61** là đạt được độ chính xác 100%.

```
const int PRIME[] = {2, 3, 5, 7, 11, 13, 17, 19,
        23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71};

bool rabinMiller(ll x, int numTry = 15) {
    if (x <= 2 || (x & 1) == 0) {
        return (x == 2 ? true : false);
    }
    ll k = x - 1;
    int m = 0;
    for (; (k & 1) == 0; k >>= 1, m++);
    for (int i = 0; i < numTry && PRIME[i] < x; i++) {
        ll t = fastPow(PRIME[i], k, x);
        if (t == 1 || t == x - 1) {
            continue;
        }
        for (int j = 0; j < m && t != x - 1; j++) {
            t = mod(t * t, x);
        }
        if (t != x - 1) {
            return false;
        }
    }
    return true;
}
```

Euler Totient Function

```
int eulerTotient(int x) {
    int ret = x, bound = sqrt(x);
    For(i, 2, bound) {
        if (x % i == 0) {
            ret = ret / i * (i - 1);
            for (; x % i == 0; x /= i);
        }
    }
    if (x != 1) {
        ret = ret / x * (x - 1);
```

```
    }
    return ret;
}
```

Tổng các ước dương của 1 số

Phân tích N thành tích của các thừa số nguyên tố:

$$N = p_1^{k_1} \times p_2^{k_2} \times \ldots \times p_n^{k_n}$$

Thì khi đó, tổng các ước dương của N là:

$$\mathrm{Sum}(N) = \frac{p_1^{k_1+1}-1}{p_1-1} \times \frac{p_2^{k_2+1}-1}{p_2-1} \times \ldots \times \frac{p_n^{k_n+1}-1}{p_n-1}$$

Giải phương trình đồng dư:

$a_1x_1 + a_2x_2 + \ldots + a_nx_n$ đồng dư với b (mod m).

Trong đó, $a_1, a_2, \ldots, a_n$, b và m là các số nguyên dương.

Nếu vô nghiệm, hàm trả về FALSE, ngược lại trả về TRUE và bộ nghiệm $(x_1, x_2, \ldots, x_n)$ được lưu trong vector ret.

```
const int MAXN = 1005;

int g[MAXN], x[MAXN];

bool congruenceEquation(vector <int> a, int b, int m, vector <int>
&ret) {
    int n = SIZE(a);
    a.PB(m);
    g[0] = a[0];
    For(i, 1, n) g[i] = gcd(g[i - 1], a[i]);
    ret.clear();
    if (b % g[n]) return false;
    int val = b / g[n];
    Ford(i, n, 1) {
        pll p = extendedEuclid(g[i - 1], a[i]);
        x[i] = p.S * val % m;
        val = p.F * val % m;
    }
    x[0] = val;
    For(i, 0, n) x[i] = (x[i] + m) % m;
    Rep(i, n) ret.PB(x[i]);
    return true;
}
```


Công thức hình học

- $\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y$
- $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y$
- $\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y}$
- $\sin 2x = 2 \sin x \cos x$
- $\cos 2x = \cos^2 x - \sin^2 x = 2\cos^2 x - 1 = 1 - 2\sin^2 x$
- $\tan 2x = \frac{2\tan x}{1 - \tan^2 x}$
- $\sin^2 \mathrm{x} = \frac{1}{2}(1 - \cos 2x)$
- $\cos^2 \mathrm{x} = \frac{1}{2}(1 + \cos 2x)$
- $\sin \mathrm{x} - \sin y = 2 \sin\frac{x-y}{2} \cos\frac{x+y}{2}$
- $\cos \mathrm{x} - \cos y = -2 \sin\frac{x-y}{2} \sin\frac{x+y}{2}$
- $\frac{\mathrm{a}}{\sin \mathrm{A}} = \frac{\mathrm{b}}{\sin \mathrm{B}} = \frac{\mathrm{c}}{\sin \mathrm{C}}$
- $\mathrm{a}^2 = \mathrm{b}^2 + c^2 - 2bc \cos A$
- $\frac{\mathrm{a-b}}{\mathrm{a+b}} = \frac{\tan\left(\frac{\mathrm{A-B}}{2}\right)}{\tan\left(\frac{\mathrm{A+B}}{2}\right)}$

- **Ước số:**
  - Tính chất nhân tính:
    - $f(m*n) = f(m)*f(n)$
    - $n = p1^{a1} + p2^{a2} + \cdots + pk^{ak}$
    - $f(n) = f(p1^{a1}) + f(p2^{a2}) + \cdots + f(pn^{an})$
  - → để tìm công thức tổng quát, ta cần tìm cho lũy thừa SNT
  - Tổng ước
    - $TongUoc(p^k) = p^0 + p^1 + \cdots .+p^k = (1-p)^{k+1}/(1-p)$
    - $TongUoc(n) = TongUoc(p_1^{a1}) * TongUoc(p_2^{a2}) \ldots * TongUoc(p_n^{an})$
    - $TongUoc(n) = Tich\left(\frac{(1-p_i)^{k+1}}{1-p_i}\right)$ (**CSN**)
  - Tích các ước
    - $TichUoc(n) = n^{\frac{SoUoc(n)}{2}}$
  - Số ước
    - $SoUoc(p^k) = k + 1$ (p là SNT)
    - $SoUoc(n) = SoUoc(p_1^{a1})SoUoc(p_2^{a2}) \ldots * SoUoc(p_n^{an})$
    - $SoUoc(n) = Tich\big((a_1+1)(a_2+1) \ldots * (a_k+1)\big)$
    - $SoUoc(1 \to n) = SoUoc(1) + SoUoc(2) + \cdots + SoUoc(n)$
    - $SoUoc(1 \to n) = SoBoi(1 \to n)$
  - Bội
    - $SoBoi(x) = n/x$ (số lượng bội của x nhỏ hơn n)
    - $SoBoi(1 \to n) = \left[\frac{n}{1}\right] + \left[\frac{n}{2}\right] + \cdots + \left[\frac{n}{n}\right]$
- **Phi hàm Euler**:
  - phi(N) là số lượng số nguyên tố cùng nhau với N trong đoạn từ 1 đến N
  - Cách tính phi của n theo cách phân tích thừa số nguyên tố:
    - res = n
    - Tìm ra các thừa số nguyên tố của n
    - với mỗi thừa số nguyên tố trừ res đi số lượng bội của số đó trong khoảng [1, res]
    - phi(n) là res
  - $phi(p^k) = p^k - \frac{p^k}{p} = p^k - p^{k-1}$ (với p là SNT)
  - $phi(n) = Tich\big(p_i^{a_i} - p_i^{a_i-1}\big)$
  - $TongCacSNTCungNhau(n) = \frac{n*phi(n)}{2}$
  - $gcd(a,m) = 1 \Rightarrow a^{phi(m)} = 1\big(mod(m)\big)$
  - $phi(n^k) = \big(p_1^k - p_1^{k-1}\big) * \big(p_1^k - p_1^{k-1}\big) * \ldots * \big(p_n^k - p_n^{k-1}\big)$
- **Hình học**
  - Diện tích đa giác:
    - $S = \frac{1}{2}|Tong(x_i y_{i+1} - x_{i+1} * y_i)|$ (diện tích đa giác bất kì)
  - Tính k/c từ A đến BC:
    - $S = \frac{1}{2}BCAI$ (với I là hình chiếu của A lên BC)
  - Diện tích tam giác khi có 3 độ dài 3 đường a, b, c:
    - $S = \sqrt{(s-a)(s-b)(s-c)s}$, $s = (a+b+c)/2$
  - Đếm số điểm nguyên trong đa giác
    - $S_h = a + \frac{b}{2} - 1$
    - a là số điểm nguyên nằm trong
    - b là số điểm nguyên trên biên
  - Đếm số điểm nguyên thuộc đoạn thẳng
    - $SoDiem = gcd\big([a_x - b_x], [a_y - b_y]\big)$
  - Cho 2 đường thẳng
    - $a_1x + b_1y + c_1 = 0$
    - $a_2x + b_2y + c_2 = 0$
    - Song song khi: $\frac{a_1}{a_2} = \frac{b_1}{b_2} \neq \frac{c_1}{c_2}$
    - Vuông góc khi: $a_1a_2 + b_1b_2 = 0$
    - $cos(\alpha) = \frac{[a_1a_2 + b_1b_2]}{\sqrt{a_1^2+b_1^2} * \sqrt{a_2^2+b_2^2}}$
    - Khi thay 1 điểm D vào đường thẳng
      - nếu > 0 thì nằm bên trái đường thẳng
      - nếu < 0 thì nằm bên phải đường thẳng
      - nếu = 0 thì nằm trên đường thẳng
  - Tìm giao điểm → giải hệ
    - $D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1 * b_2 - b_1 * a_2$
    - $D_x = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix} = c_1 * b_2 - b_1 * c_2$

  - $D_y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} = a_1 * c_2 - c_1 * a_2$
  - $x = \frac{D_x}{D}, y = \frac{D_y}{D}$
- PT đường tròn
  - $(x-a)^2 + (y-b)^2 = R^2$

- **Phép biến đổi Affine:**
  - 2D:
    - Phép tịnh tiền (Dịch chuyển x tới t_x và y lên t_y) :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

    - Phép vị tự (Phóng to x lên t_x và y lên t_y):

$$\begin{bmatrix} t_x & 0 & 0 \\ 0 & t_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

    - Phép xoay (1 góc θ ngược chiều kim đồng hồ):

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - 3D:
    - Phép tịnh tiền (Dịch chuyển x tới t_x và y lên t_y) :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

    - Phép vị tự (Phóng to x lên t_x và y lên t_y):

$$\begin{bmatrix} t_x & 0 & 0 & 0 \\ 0 & t_y & 0 & 0 \\ 0 & 0 & t_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

    - Phép xoay (1 góc θ ngược chiều kim đồng hồ theo trục Ox):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,\theta & -sin\,\theta & 0 \\ 0 & sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Bảng đạo hàm		**Bảng nguyên hàm**	
$x^{\alpha}\ ' = \alpha x^{\alpha-1}$	$(u^{\alpha})' = \alpha.u'.u^{\alpha-1}$	$\int x^{\alpha}dx = \frac{x^{\alpha+1}}{\alpha+1} + c,\ (\alpha \neq -1)$	$\int (ax+b)^{\alpha}dx = \frac{1}{a}.\frac{(ax+b)^{\alpha+1}}{\alpha+1} + c$
$(\sin x)' = \cos x$	$(\sin u)' = u'.\cos u$	$\int \sin x dx = -\cos x + c$	$\int \sin(ax+b)dx = -\frac{1}{a}\cos(ax+b) + c$
$(\cos x)' = -\sin x$	$(\cos u)' = -u'.\sin u$	$\int \cos x dx = \sin x + c$	$\int \cos(ax+b)dx = \frac{1}{a}\sin(ax+b) + c$
$(\tan x)' = \frac{1}{\cos^2 x} = 1 + \tan^2 x$	$(\tan u)' = \frac{u'}{\cos^2 u} = u'.(1+\tan^2 u)$	$\int \frac{1}{\cos^2 x}dx = \tan x + c$	$\int \frac{1}{\cos^2(ax+b)}dx = \frac{1}{a}\tan(ax+b) + c$
$(\cot x)' = \frac{-1}{\sin^2 x} = -(1+\cot^2 x)$	$(\cot u)' = \frac{-u'}{\sin^2 u} = -u'.(1+\cot^2 u)$	$\int \frac{1}{\sin^2 x}dx = -\cot x + c$	$\int \frac{1}{\sin^2(ax+b)}dx = -\frac{1}{a}\cot(ax+b) + c$
$\log_a x\ ' = \frac{1}{x \ln a}$	$\log_a u\ ' = \frac{u'}{u.\ln a}$	$\int \frac{1}{x}dx = \ln\|x\| + c$	$\int \frac{1}{ax+b}dx = \frac{1}{a}\ln\|ax+b\| + c$
$\ln x\ ' = \frac{1}{x}$	$\ln u\ ' = \frac{u'}{u}$		
$a^x\ ' = a^x.\ln a$	$a^u\ ' = a^u.u'.\ln a$	$\int a^x dx = \frac{a^x}{\ln a} + c$	$\int a^{\alpha x+\beta}dx = \frac{a^{\alpha x+\beta}}{\alpha.\ln a} + c$
$e^x\ ' = e^x$	$(e^u)' = u'.e^u$	$\int e^x dx = e^x + c$	$\int e^{ax+b}dx = \frac{1}{a}e^{ax+b} + c$