~~Stanford~~ $\underbrace{\text{University}}_{\textit{of rochester}}$ ICPC Team Notebook (2016-17-18)

Contents

1 Data Structures

1.1 Binary-Indexed Tree

```
//point update, range query, 0-indexed
template <class T>
struct bit {
	vector<T> b;
	void init(int n){b.resize(n+1);}
	bit(){}
	bit(int n){init(n);}
	inline void update(int i, T v) {
		for(++i; i<(int)b.size(); i+=i&-i)
			b[i] += v;
	}
	//sum of the first i values
	T prefix(int i) const {
		T a = 0;
		for(; i; i^=i&-i)
			a += b[i];
		return a;
	}
	inline T query(int l, int r) const {
		return prefix(r+1)-prefix(l);
	}
};
```

1.2 2D Binary-Indexed Tree

```
//2d fenwick tree, 1-indexed
template <class T>
struct bit_2d {
	int N,M;
	vector<vector<T>> b;

	void init(int n, int m){N=n+1,M=m+1,b=vector<vector<T>>(N,vector<T>(M));}
	bit_2d(){}
	bit_2d(int n, int m){init(n,m);}

	inline void update(int i, int j, T v) {
		for(;i < N; i += i&-i)
			for(int k = j; k < M; k += k&-k)
				b[i][k] += v;
	}

	//sum of the 'prefix' i x j rectangle
	inline T prefix(int i, int j) {
		T a = 0;
		for(;i;i ^= i&-i)
			for(int k = j;k;k ^= k&-k)
				a += b[i][k];
		return a;
	}

	inline T query(int a, int b, int c, int d) {
		return prefix(c,d)+prefix(a-1,b-1)-prefix(a-1,d)-prefix(c,b-1);
	}
};
```

1.3 Range BIT

```
//range update, point query, 0-indexed
template <class T>
struct bit {
	vector<T> b;
	void init(int n){b.resize(n+1);}
	bit(){}
	bit(int n){init(n);}
	inline void ud(int i, T v) {
		for(; i<b.size(); i+=i&-i)
			b[i] += v;
	}
```

```
	//update [l,r] with value v
	inline void update(int l, int r, T v) {
		ud(l+1,v);
		if(r+2<b.size())ud(r+2,-v);
	}

	//get value at i
	inline int query(int i) const {
		T a = 0;
		for(++i; i; i^=i&-i)
			a += b[i];
		return a;
	}
};
```

1.4 RMQ

```
// O(n log n) space, O(1) index query R(max)Q
// returns the INDEX of the max element
struct rMq_ind {
	vector<vector<int> > t;
	int *A;
	rMq_ind(){}
	rMq_ind(int* a, int n):t(32-__builtin_clz(n),vector<int>(n)) {
		A = a;
		for(int i = 0; i < n; ++i)
			t[0][i] = i;
		for(int k = 1, p = 1; k < (int)t.size(); ++k, p<<=1)
			for(int i = 0; i < n; ++i)
				t[k][i] = (i+p<n && a[t[k-1][i+p]] > a[t[k-1][i]])?t[k-1][i+p]:t[k-1][
					i];
	}
	//inclusive max query
	inline int query(int l, int r) const {
		int p = 31-__builtin_clz(r-l+1), i = t[p][l], j = t[p][r+1-(1<<p)];
		return (A[i]>A[j])?i:j;
	}
};
```

1.5 Segment Tree

```
struct segt {
	int N = 100005;
	vector<ll> t,lazy;
	vector<int> l,r;

	void build(int i, int j, int v) {
		l[v] = i, r[v] = j;
		//CHANGE ME
		t[v] = lazy[v] = 0;
		if(i == j) return;
		build(i,(i+j)/2,v<<1);
		build((i+j)/2+1,j,v<<1|1);
	}

	void init(int n) {
		N = n;
		t.resize(4*n), lazy.resize(4*n);
		l.resize(4*n), r.resize(4*n);
		build(0,n-1,1);
	}

	segt(){}
	segt(int n){init(n);}

	// --- CHANGE ME ---
	inline ll merge(ll a, ll b) {
		return max(a,b);
	}

	// --- CHANGE ME ---
	//propagate lazy value downwards
	inline void prop(int v) {
		if(l[v]!=r[v]) {
			lazy[v<<1] += lazy[v];
			lazy[v<<1|1] += lazy[v];
		}
		t[v] += lazy[v];
		lazy[v] = 0;
	}

	void update(int i, int j, ll val, int v = 1) {
		if(j < l[v] || r[v] < i) return;
		if(i <= l[v] && r[v] <= j) {
			// --- CHANGE ME ---
			//apply lazy update to v's range
			lazy[v] += val;
			return;
		}
		prop(v);
		update(i,j,val,v<<1);
		update(i,j,val,v<<1|1);
		prop(v<<1), prop(v<<1|1);
		t[v] = merge(t[v<<1],t[v<<1|1]);
	}

	ll query(int i, int j, int v = 1) {
		if(j < l[v] || r[v] < i) {
			// --- CHANGE ME ---
			//return result for empty range
			return 0;
		}
		prop(v);
		if(i <= l[v] && r[v] <= j)
			return t[v];
		return merge(query(i,j,v<<1),query(i,j,v<<1|1));
	}
};
```

1.6 KD-Tree

```
// - constructs from n points in O(n lg^2 n) time
// - handles nearest-neighbor query in O(lg n) if points are well distributed
// - O(log n) average nearest neighbor, O(n) worst in pathological case

#include <bits/stdc++.h>

using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

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

// sorts points on x-coordinate
bool on_x(const point &a, const point &b) {
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b) {
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b) {
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox {
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)       return pdist2(point(x0, y0), p);
            else if (p.y > y1)  return pdist2(point(x0, y1), p);
            else                return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
```

```
            if (p.y < y0)       return pdist2(point(x1, y0), p);
            else if (p.y > y1)  return pdist2(point(x1, y1), p);
            else                return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)       return pdist2(point(p.x, y0), p);
            else if (p.y > y1)  return pdist2(point(p.x, y1), p);
            else                return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode {
    bool leaf;      // true if this is a leaf node (has one point)
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp) {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();   first->construct(vl);
            second = new kdnode();  second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree {
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p) {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//            if (p == node->pt) return sentry;
//            else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

// --------------------------------------------------------------------------
// some basic test code here

int main() {
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i)
        vp.push_back(point(rand()%100000, rand()%100000));
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

// --------------------------------------------------------------------------
```

1.7 Wavelet Tree

```
template <class T>
struct wavelet{
        struct node {
                vector<int> b;
                T lo,hi,md;
        };
        vector<node> t;

        void build(const vector<T> &c, T *A, T *B, int v, int i, int j) {
                t[v].b.resize(B-A+1);
                t[v].lo = c[i], t[v].hi = c[j], t[v].md = c[(i+j)/2];
                for(int i = 0; A+i != B; ++i)
                        t[v].b[i+1] = t[v].b[i] + (A[i]<=t[v].md);
                if(i == j) return;
                T *p = stable_partition(A,B,[=](int x){return x <= t[v].md;});
                build(c,A,p,v<<1,i,(i+j)/2);
                build(c,p,B,v<<1|1,(i+j)/2+1,j);
        }

        void init(T *A, int n) {
                vector<T> c(A,A+n);
                sort(c.begin(),c.end());
                c.erase(unique(c.begin(),c.end()),c.end());
                int N = c.size();
                t.resize(N<<2);
                build(c,A,A+n,1,0,N-1);
        }

        wavelet(){}
        wavelet(T *A, int n){init(A,n);}

        //kth smallest element in [l, r]
        T kth(int l, int r, int k, int v = 1) {
                if(t[v].lo == t[v].hi) return t[v].lo;
                int lb = t[v].b[l], rb = t[v].b[r+1], il = rb-lb;
                return (k < il) ? kth(lb,rb-1,k,v<<1) : kth(l-lb,r-rb,k-il,v<<1|1);
        }

        //number of elements in [l,r] <= to a
        int leq(int l, int r, T a, int v = 1) {
                if(a < t[v].lo) return 0;
                if(t[v].hi <= a) return r-l+1;
                int lb = t[v].b[l], rb = t[v].b[r+1];
                return leq(lb,rb-1,a,v<<1) + leq(l-lb,r-rb,a,v<<1|1);
        }

        //number of elements in [l, r] equal to a
        int count(int l, int r, T a, int v = 1) {
                if(a < t[v].lo || a > t[v].hi) return 0;
                if(t[v].lo == t[v].hi) return r-l+1;
                int lb = t[v].b[l-1], rb = t[v].b[r];
                if(a <= t[v].md) return count(lb,rb-1,a,v<<1);
                return count(l-lb,r-rb,a,v<<1|1);
```

```
	}
};
```

1.8 Lazy Treap

```
//to un-lazy: ignore lazy,update,push,recalc
namespace treap {
	typedef int data;
	struct node {
		data v, lazy=0;
		int p,sz=1;
		node *l=0, *r=0;
		node(data v):v(v),p(rand()) {};
		~node(){if(l) delete l; if(r) delete r; }
	};

	//lazy update to all values in subtree of d
	inline void update(node* d, data val) {
		if(d) d->lazy += val;
	}

	//push lazy value of d to children
	inline void push(node *d) {
		if(d && d->lazy) {
			if(d->l) d->l->lazy += d->lazy;
			if(d->r) d->r->lazy += d->lazy;
			d->v += d->lazy;
			d->lazy = 0;
		}
	}

	//node size
	inline int size(node *d) {
		return d?d->sz:0;
	}

	//recalc size from children
	inline void recalc(node *d) {
		d->sz = 1 + size(d->l) + size(d->r);
	}

	//split into nodes <= v and nodes > v
	void split(node *d, data v, node *&l, node *&r) {
		l = r = 0;
		if(!d) return;
		push(d);
		if(v <= d->v) {
			split(d->l,v,l,d->l);
			r = d;
		} else {
			split(d->r,v,d->r,r);
			l = d;
		}
		recalc(d);
	}

	//split such that l has size sz
	void split_size(node *d, int sz, node *&l, node *&r) {
		l = r = 0;
		if(!d) return;
		push(d);
		if(size(d->l) >= sz) {
			split(d->l,sz,l,d->l);
			r = d;
		} else {
			split(d->r,sz,d->r,r);
			l = d;
		}
		recalc(d);
	}

	//all values in l must be less than all those in r
	node* merge(node *l, node *r) {
		if(!l || !r) return l?l:r;
		push(l), push(r);
		if(l->p < r->p) {
			l->r = merge(l->r,r);
			recalc(l);
			return l;
		}
		r->l = merge(l,r->l);
		recalc(r);
		return r;
	}

	//insert value v
	void insert(node *&d, int v) {
		node *l,*r;
		split(d,v,l,r);
		d = merge(merge(l,new node(v)),r);
	}

	//erase value v
	void erase(node *&d, int v) {
		node *l,*m,*r;
		split(d,v,l,m);
		split(m,v+1,m,r);
		if(m) delete m;
		d = merge(l,r);
	}

	//value of element at 0-based index k
	data kth(node* d, int k) {
		push(d);
		if(size(d->l) == k) return d->v;
		if(k < size(d->l)) return kth(d->l,k);
		return kth(d->r,k-size(d->l)-1);
	}

	//number of elements strictly less than v
	int index(node* d, data v) {
		if(!d) return 0;
		push(d);
		if(v == d->v) return size(d->l);
		if(v < d->v) return index(d->l,v);
		return 1 + size(d->l) + index(d->r,v);
	}

	//does d contain value v?
	bool contains(node* d, data v) {
		if(!d) return false;
		push(d);
		if(v == d->v) return true;
		if(v < d->v) return contains(d->l,v);
		return contains(d->r,v);
	}

	void print(node* d) {
		if(!d) return;
		print(d->r);
		printf("%d ",d->sol);
		print(d->l);
	}
}

treap::node *root = 0;
```

2 Flow and Matching

2.1 Max Flow

```
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef long long ll;

template <class X>
struct dinic {
	struct edge {
		int u, v;
		X cap, flow;
		edge() {}
		edge(int u, int v, X cap): u(u), v(v), cap(cap), flow(0) {}
	};

	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 add_edge(int u, int v, X cap) {
		if (u == v) return;
		g[u].emplace_back(E.size());
		E.emplace_back(edge(u, v, cap));
		g[v].emplace_back(E.size());
		E.emplace_back(edge(v, u, 0));
	}

	bool bfs(int S, int T) {
```

```
		fill(d.begin(), d.end(), N + 1);
		int qf=0,qb=1;
		d[S] = 0;
		pt[0] = S;
		while(qf!=qb) {
			int u = pt[qf++];
			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;
					pt[qb++] = e.v;
				}
			}
		}
		return d[T] != N + 1;
	}

	X dfs(int u, int T, X flow = -1) {
		if(u == T || !flow) 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) {
				X amt = e.cap - e.flow;
				if(flow != -1 && amt > flow) amt = flow;
				if(X pushed = dfs(e.v, T, amt)) {
					e.flow += pushed;
					oe.flow -= pushed;
					return pushed;
				}
			}
		}
		return 0;
	}

	X flow(int S, int T) {
		X tot = 0;
		while(bfs(S, T)) {
			// if using fp arithmetic, limit this to N passes explicitly
			fill(pt.begin(), pt.end(), 0);
			while(X flow = dfs(S, T))
				tot += flow;
		}
		return tot;
	}
};
```

2.2 Bipartite Matching

```
// init with hopcroft_karp(left size, right size)
// add_edge(i,j) adds edge from i-th in left to j-th in right (0-indexed)
// solve() returns size of matching, i-th in left is matched to E[i]-th in right
struct hopcroft_karp {
	int N,M;
	vector<vector<int>> E;
	vector<int> dist,match,Q;
	hopcroft_karp(int n, int m):N(n),M(m),E(N),dist(N+M),match(N+M,-1),Q(N+M){}

	inline void add_edge(int i, int j) {E[i].push_back(j);}

	bool bfs() {
		fill(&dist[0],&dist[0]+N+M,-1);
		int qf = 0, qb = 0, u;
		bool ok = false;
		for(int i = 0; i < N; ++i)
			if(match[i] == -1)
				Q[qb++] = i, dist[i] = 0;
		while(qf != qb) {
			if((u = Q[qf++]) < N) {
				for(int v : E[u])
					if(dist[N+v] == -1)
						dist[Q[qb++] = N+v] = dist[u] + 1;
			} else {
				if(match[u] == -1) ok = true;
				else if(dist[match[u]] == -1)
					dist[Q[qb++] = match[u]] = dist[u] + 1;
			}
		}
		return ok;
	}

	bool dfs(int u) {
		for(int &i = Q[u]; i < E[u].size(); ++i) {
			int v = N+E[u][i];
			if(dist[v] == dist[u]+1 && (match[v] == -1 || (dist[match[v]] == dist[v]+1 &&
			    dfs(match[v])))) {
				match[v] = u, match[u] = v-N;
				return true;
			}
		}
		return false;
	}

	int solve() {
		int ans = 0;
		while(bfs()) {
			fill(&Q[0],&Q[0]+N,0);
			for(int i = 0; i < N; ++i)
				if(match[i] == -1 && dfs(i))
					++ans;
		}
		return ans;
	}
};
```

2.3 Min-cost Flow

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

struct mcmf {
	int N;
	#define BND 1LL<<61
	vector<vector<ll> > cap, fl, cost;
	vector<bool> found;
	vector<ll> dist, pi, width;
	vector<pii> dad;

	mcmf(int N) :
		N(N), cap(N,vector<ll>(N)), fl(N,vector<ll>(N)), cost(N,vector<ll>(N)),
		found(N), dist(N), pi(N), width(N), dad(N) {}

	void add_edge(int from, int to, ll cap, ll cost) {
		this->cap[from][to] = cap;
		this->cost[from][to] = cost;
	}

	inline void relax(int s, int k, ll cap, ll cost, int dir) {
		ll val = dist[s] + pi[s] - pi[k] + cost;
		if(cap && val < dist[k]) {
			dist[k] = val;
			dad[k] = pii(s, dir);
			width[k] = min(cap, width[s]);
		}
	}

	ll dijkstra(int s, int t) {
		fill(found.begin(),found.end(),0);
		fill(dist.begin(),dist.end(),BND);
		fill(width.begin(),width.end(),0);
		dist[s] = 0;
		width[s] = BND;
		while(s != -1) {
			int best = -1;
			found[s] = true;
			for(int k = 0; k < N; k++) {
				if (found[k]) continue;
				relax(s,k,cap[s][k]-fl[s][k],cost[s][k],1);
				relax(s,k,fl[k][s],-cost[k][s],-1);
				if(best == -1 || dist[k] < dist[best]) best = k;
			}
			s = best;
		}
		for (int k = 0; k < N; k++)
			pi[k] = min(pi[k] + dist[k], BND);
		return width[t];
	}

	pair<ll,ll> flow(int s, int t) {
		ll totflow = 0, totcost = 0;
		while(ll amt = dijkstra(s, t)) {
			totflow += amt;
			for(int x = t; x != s; x = dad[x].first) {
				if(dad[x].second == 1) {
					fl[dad[x].first][x] += amt;
					totcost += amt * cost[dad[x].first][x];
				} else {
					fl[x][dad[x].first] -= amt;
					totcost -= amt * cost[x][dad[x].first];
				}
```

```
                }
            }
            return make_pair(totflow,totcost);
        }
};
```

2.4 LP Solver

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.   Then, call Solve(x).

#include <bits/stc++.h>

using namespace std;

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
        int m, n;
        VI B, N;
        VVD D;

        LPSolver(const VVD &A, const VD &b, const VD &c) :
                m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
                for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
                for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] = b[i]; }
                for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
                N[n] = -1; D[m + 1][n] = 1;
        }

        void Pivot(int r, int s) {
                double inv = 1.0 / D[r][s];
                for (int i = 0; i < m + 2; i++) if (i != r)
                        for (int j = 0; j < n + 2; j++) if (j != s)
                                D[i][j] -= D[r][j] * D[i][s] * inv;
                for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
                for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
                D[r][s] = inv;
                swap(B[r], N[s]);
        }

        bool Simplex(int phase) {
                int x = phase == 1 ? m + 1 : m;
                while (true) {
                        int s = -1;
                        for (int j = 0; j <= n; j++) {
                                if (phase == 2 && N[j] == -1) continue;
                                if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s])
                                        s = j;
                        }
                        if (D[x][s] > -EPS) return true;
                        int r = -1;
                        for (int i = 0; i < m; i++) {
                                if (D[i][s] < EPS) continue;
                                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                                        (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B
                                        [r]) r = i;
                        }
                        if (r == -1) return false;
                        Pivot(r, s);
                }
        }

        DOUBLE Solve(VD &x) {
                int r = 0;
                for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
                if (D[r][n + 1] < -EPS) {
                        Pivot(r, n);
                        if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<DOUBLE>::
                                infinity();
                        for (int i = 0; i < m; i++) if (B[i] == -1) {
                                int s = -1;
                                for (int j = 0; j <= n; j++)
                                        if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j]
                                                < N[s]) s = j;
                                Pivot(i, s);
                        }
                }
                if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
                x = VD(n);
                for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
                return D[m][n + 1];
        }
};

int main() {

        const int m = 4;
        const int n = 3;
        DOUBLE _A[m][n] = {
                { 6, -1, 0 },
                { -1, -5, 0 },
                { 1, 5, 1 },
                { -1, -5, -1 }
        };
        DOUBLE _b[m] = { 10, -4, 5, -5 };
        DOUBLE _c[n] = { 1, -1, 0 };

        VVD A(m);
        VD b(_b, _b + m);
        VD c(_c, _c + n);
        for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

        LPSolver solver(A, b, c);
        VD x;
        DOUBLE value = solver.Solve(x);

        cerr << "VALUE: " << value << endl; // VALUE: 1.29032
        cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
        for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
        cerr << endl;
        return 0;
}
```

2.5 Stable Marriage

```
//solves SPOJ STABLEMP
#include <bits/stdc++.h>
using namespace std;

//male[i][j] is the j-th most desired female for male i
//analogous for female[i][j]
const int N = 505;
int male[N][N],female[N][N],propose[N];
int wife[N],husband[N],tmp[N];
int bachelors[N],bsz=0;
void marry(int n) {
        fill(wife,wife+n,-1);
        fill(husband,husband+n,-1);
        fill(propose,propose+n,0);
        bsz = n;
        for(int i = 0; i < n; ++i) {
                bachelors[i] = i;
                copy(female[i],female[i]+n,tmp);
                for(int j = 0; j < n; ++j)
                        female[i][tmp[j]] = j;
        }
        while(bsz) {
                int i = bachelors[--bsz], j = male[i][propose[i]++];
                if(husband[j] == -1)
                        wife[i] = j, husband[j] = i;
                else if(female[j][husband[j]] > female[j][i])
                        bachelors[bsz++] = husband[j], husband[j] = i, wife[i] = j;
                else
                        ++bsz;
        }
}

int main() {
        int T;
        scanf("%d",&T);
        while(T--) {
                int n,t;
                scanf("%d",&n);
                for(int i = 0; i < n; ++i) {
```

```
			scanf("%d",&t);
			for(int j = 0; j < n; ++j)
				scanf("%d",female[i]+j),--female[i][j];
		}
		for(int i = 0; i < n; ++i) {
			scanf("%d",&t);
			for(int j = 0; j < n; ++j)
				scanf("%d",male[i]+j),--male[i][j];
		}
		marry(n);
		for(int i = 0; i < n; ++i)
			printf("%d %d\n",i+1,wife[i]+1);
	}
}
```

3 Geometry

3.1 2D Floating-point Geometry

```
#include <bits/stdc++.h>
using namespace std;

typedef long double ld;
constexpr ld EPS = 1e-10;
struct point {
	ld x, y;
	point(){}
	point(ld x, ld y) : x(x), y(y) {}
	point operator + (const point &p) const { return point(x+p.x, y+p.y); }
	point operator - (const point &p) const { return point(x-p.x, y-p.y); }
	point operator * (ld c) const { return point(x*c, y*c); }
	point operator / (ld c) const { return point(x/c, y/c); }
	bool operator == (const point &p) const { return fabsl(x-p.x) + fabsl(y-p.y) < EPS; }
	bool operator < (const point &p) const { return (x==p.x) ? y<p.y : x<p.x; }
};

ostream& operator<<(ostream &o, const point &p) {
	return o << "(" << p.x << "," << p.y << ")",o;
}

ld dot(point p, point q) { return p.x*q.x + p.y*q.y; }
ld cross(point p, point q) { return p.x*q.y - p.y*q.x; }
ld norm(point p) { return p.x*p.x + p.y*p.y; }
ld mag(point p) { return sqrtl(p.x*p.x + p.y*p.y); }
ld dist(point p, point q) { return mag(p - q); }
ld sq_dist(point p, point q) { return norm(p - q); }

// +----------------------------------------------+
// |               LINES AND SEGMENTS             |
// +----------------------------------------------+

//closest point to p on line (a,b)
point projectPL(point p, point a, point b) {
	return a + (b-a) * dot(p-a, b-a) / norm(b-a);
}

//how far along (a,b) is p? (projected)
//0 at a, 1 at b
ld project_scale(point p, point a, point b) {
	return dot(p-a, b-a) / norm(b-a);
}

//how far along (a,b) is p? (projected)
// 0 at a, |b-a| at b
ld project_dist(point p, point a, point b) {
	return dot(p-a, b-a) / mag(b-a);
}

//closest point to p on segment (a,b)
point projectPS(point p, point a, point b) {
	if (a == b) return a;
	ld r = dot(p-a, b-a) / dot(a-b, a-b);
	if (r < 0) return a;
	if (r > 1) return b;
	return a + (b-a)*r;
}

//reflect p over line (a,b)
point reflectPL(point p, point a, point b){
	return (a + (b-a)*dot(p-a, b-a)/norm(b-a))*2 - p;
}

//is p on line (a,b) ?
bool onPL(point p, point a, point b){
	return fabsl(cross(p-a, b-a)) < EPS;
}

//is p on segment (a,b) ?
bool onPS(point p, point a, point b){
	return fabsl(cross(a-p, b-p)) < EPS && ( (a<p) != (b<p) || p==a || p==b);
}

// are lines (a,b) and (c,d) parallel?
bool parallelLL(point a, point b, point c, point d) {
  return fabsl(cross(b-a,d-c)) < EPS;
}

// are lines (a,b) and (c,d) equal?
bool equalLL(point a, point b, point c, point d) {
  return onPL(c,a,b) && onPL(d,a,b) && onPL(a,c,d) && onPL(b,c,d);
}

//p -> a -> b: -1 = Clockwise, 0 = Colinear, 1 = Counterclockwise
//p with a->b: -1 = Left, 0 = On, 1 = Right
int sidePL(point p, point a, point b) {
	ld d = cross(p-a,b-a);
	if(fabsl(d) < EPS) return 0;
	return (d<0)?-1:1;
}

//does segment (a,b) intersect line (c,d) ?
bool intersectSL(point a, point b, point c, point d){
	ld x = cross(a-c,a-d), y = cross(b-c,b-d);
	return x == 0 || y == 0 || (x<0) != (y<0);
}

//do segments (a,b) and (c,d) intersect?
int intersectSS(point a, point b, point c, point d) {
	if(equalLL(a,b,c,d)) {
		if(b < a) swap(a, b);
		if(d < c) swap(c, d);
		if(c < a) swap(a, c), swap(b, d);
		if(c == b || (c == d && c < b)) return 2; //POINT
		else if(c < b) return 1; //SEGMENT
		else return 0; //NONE
	}
	else if(intersectSL(a,b,c,d) && intersectSL(c,d,a,b))
		return 2; //POINT
	else return 0; //NONE
}

//intersection point of distinct lines (a,b) and (c,d)
point intersectLL(point a, point b, point c, point d) {
  return a + (b-a)*(cross(c-a, c-d))/(cross(b-a, c-d));
}

//return perpendicular to (a,b) through midpoint
pair<point,point> perpendicularS(point a, point b) {
	ld m = (a.x + b.x + a.y + b.y)/2;
	return make_pair(point(m-b.y, m-a.x), point(m-a.y, m-b.x));
}

//return perpendicular to (a,b) through p
pair<point,point> perpendicularPL(point p, point a, point b){
	return make_pair(p, a + (b-a)*dot(p-a, b-a)/norm(b-a));
}

point rotate90CC(point p) {
	return point(-p.y,p.x);
}

// +----------------------------------------------+
// |                CIRCLES AND ARCS              |
// +----------------------------------------------+

//center of arc with radius r through p and q
point centerA(point p, point q, ld r) {
	point m = (p+q)/2;
	auto l = perpendicularS(p, q);
	ld d = sqrtl(r*r - norm(q - p)/4);
	return m + (l.second-l.first)/mag(l.second-l.first)*d;
}

//angle of arc with radius r through p and q
ld angleA(point p, point q, ld r)       {
	return 2*asinl(norm(q-p)/(4*r*r));
}

//length of arc with radius r through p and q
ld lengthA(point p, point q, ld r) {
	return 2 * r * asinl(norm(q-p)/(4*r*r));
}

//circumcircle of 3 points as <center,radius>
pair<point,ld> circumcirclePPP(point a, point b, point c) {
	auto l = perpendicularS(a,b), m = perpendicularS(a,c);
	auto p = intersectLL(l.first, l.second, m.first, m.second);
```

```
	return make_pair(p, mag(p-a));
}

//incircle of 3 points as <center, radius>
pair<point,ld> incirclePPP(point a, point b, point c){
	ld d = 1.0/(mag(a-b)+mag(a-c)+mag(b-c));
	return make_pair((a * mag(b-c) + b * mag(a-c) + c * mag(a-b))*d,cross(b-a,c-a)*d);
}

// rotate p around origin by t radians
point rotateP(point p, ld t) {
  return point(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

//vector of all points line (a,b) intersects circle (c,r)
//errs on the side of accepting a single intersection
vector<point> intersectLC(point a, point b, point c, ld r) {
	vector<point> ans;
	point p = projectPL(c,a,b);
	ld d1 = mag(p-c), d2 = r*r - d1*d1, d = 1.0/mag(b-a);
	if(d2 < -EPS) return ans;
	if(d2 < EPS) {
		ans.push_back(p);
		return ans;
	}
	ans.push_back(p+(b-a)*d2*d);
	ans.push_back(p-(b-a)*d2*d);
	return ans;
}

// intersect circles with (center,radius) equal to (c,r) and (d,s)
vector<point> intersectCC(point c, ld r, point d, ld s) {
	vector<point> ans;
	ld d1 = mag(c-d);
	if(d1 > r+s || d1+min(r,s) < max(r,s)) return ans;
	ld d2 = (d1*d1-r*r+s*s)/(2*d1);
	ld d3 = sqrtl(r*r-d2*d2);
	point v = (d-c)/d1;
	ans.push_back(c+v*d2 + rotate90CC(v)*d3);
	if(d3 > EPS) ans.push_back(c+v*d2 - rotate90CC(v)*d3);
	return ans;
}

// returns a vector of tangents (internal tangents iff inner = true, external otherwise)
// each vector has two points, the tangent point on c and the tangent point on d
vector<vector<point>> tangentCC(point c, ld r, point d, ld s, bool inner) {
	vector<vector<point>> ans;
	if (inner) s = -s;
	point dist = d-c;
	ld dr = r-s, d2 = norm(dist), h2 = d2-dr*dr;
	if (d2 == 0 || h2 < 0)
		return ans;
	for (ld sign : {-1,1}) {
		point v = (dist*dr + rotate90CC(dist)*sqrt(h2)*sign)/d2;
		ans.push_back({c + v*r, d + v*s});
	}
	if(ans.size() == 2 && ans[0] == ans[1]){
		ans.pop_back();
	}
	return ans;
}

// returns the points of tangency to c from p
// the first tangent is always the one such that if you started at p, walked along
// the tangent line, and then walked along the circle without changing direction,
// you'd be walking clockwise around the circle (unless p is on c in which case
// there is one tangent)
vector<point> tangentPC(point p, point c, ld r) {
	vector<point> ans;
	for(vector<point> v : tangentCC(p, 0, c, r, true)){
		ans.push_back(v[1]);
	}
	return ans;
}

// +-----------------------------------------------+
// |                    POLYGONS                   |
// +-----------------------------------------------+

//1 = Inside, 0 = On, -1 = Outside
int sidePG(point p, vector<point> &g) {
	int c = 0, n = g.size();
	for(int i = 0; i < n; i++)
	if(onPS(p, g[i], g[(i+1) % n]))
		return 0;
	for(int i = 0; i < n; i++){
		point a = g[i];
		point b = g[(i+1) % n];
		c ^= ((a.y<=p.y) != (b.y<=p.y)) && ((b.y>a.y) != ((a.x-b.x)*(p.y-a.y)<(a.x-p.x)*(b.y-a
			.y)));
	}
	return c*2-1;
}

ld areaG(vector<point> &g) {
	ld area = 0;
	for(int i = 0; i < (int)g.size(); i++)
		area += cross(g[i], g[(i+1)%g.size()]);
	return fabsl(area / 2.0);
}

// +-----------------------------------------------+
// |             COMPARISON FUNCTIONS              |
// +-----------------------------------------------+

// "globals" we might need to capture
point POINT, DIR, LN_A, LN_B;

//Sort radially around POINT assuming they all lie on the same halfplane
bool cmp1(point a, point b){
	return cross(a-POINT,b-POINT) > 0;
}

// Sort around POINT starting and ending from a line in the direction of DIR
bool cmp2(point a, point b){
	if(a==b) return false;
	point p = POINT, q = POINT+DIR;
	if(cross(a-p,a-q)*cross(b-p,b-q) >= 0){
		if(cross(a-p,a-q) == 0 && dot(a-p,a-q) > 0) return true;
		if(cross(b-p,b-q) == 0 && dot(b-p,b-q) > 0) return false;
		return cross(b-p,a-p) < 0;
	}
	return cross(a-p,a-q) > 0;

}

// Sort according to projections on LN_A -> LN_B
bool cmp3(point a, point b){
	point p = projectPL(a, LN_A, LN_B), q = projectPL(b, LN_A, LN_B);
	return (LN_A < LN_B) != (q < p);
}

// Sort lines by angle starting and ending from a line in the direction of LINE
struct ln {
	point p, q;
	bool operator==(const ln& l) const { return p==l.p && q==l.q; }
};
bool cmp4(const ln &l, const ln &m){
	if(l==m) return false;
	point p = LN_A, q = LN_B, a = LN_A + l.q - l.p, b = LN_A + m.q - m.p;
	if(cross(a-p, b-p) == 0 && (a<p == b<p))
		return sidePL(l.p, l.q, m.p) >= 0;
	if(cross(a-p,a-q) * cross(b-p,b-q) >= 0)
		return (cross(a-p,a-q) == 0 && dot(a-p,a-q) < 0) || (!(cross(b-p,b-q) == 0 && dot(b-p,
			b-q) < 0) && cross(b-p,a-p) < 0);
	return cross(a-p,a-q) > 0;
}

int main() {

}
```

3.2 3D Geometry

```
#include <bits/stdc++.h>
using namespace std;


typedef long double ld;
constexpr ld EPS = 1e-10;
struct point {
	ld x, y, z;
	point(){}
	point(ld x, ld y, ld z) : x(x), y(y), z(z){}
	point operator + (const point &p) const { return point(x+p.x, y+p.y, z+p.z); }
	point operator - (const point &p) const { return point(x-p.x, y-p.y, z-p.z); }
	point operator * (ld c) const { return point(x*c, y*c, z*c); }
	point operator / (ld c) const { return point(x/c, y/c, z/c); }
	bool operator == (const point &p) const { return fabsl(x-p.x) + fabsl(y-p.y) + fabsl(z-p.z) <
		EPS; }
	bool operator < (const point &p) const { return (x==p.x) ? ((y==p.y) ? z < p.z : y < p.y) : x
		< p.x; }
};

point cross(point p, point q) {return point(p.y * q.z - q.y * p.z, p.z * q.x - q.z * p.x, p.x * q.y -
	q.x * p.y);}
ld dot(point p, point q) { return p.x*q.x + p.y*q.y + p.z*q.z; }
ld norm(point p) { return p.x*p.x + p.y*p.y + p.z*p.z; }
ld mag(point p) { return sqrtl(p.x*p.x + p.y*p.y + p.z*p.z); }
```

```
ld dist(point p, point q) { return mag(p - q); }
ld sq_dist(point p, point q) { return norm(p - q); }

// distance from point (x, y, z) to plane aX + bY + cZ + d = 0
ld DistPtPl(point p, ld a, ld b, ld c, ld d) {
        return abs(a*p.x + b*p.y + c*p.z + d) / sqrt(a*a + b*b + c*c);
}

// distance between parallel planes aX + bY + cZ + d1 = 0 and
// aX + bY + cZ + d2 = 0
ld DistPlPl(double a, double b, double c, double d1, double d2) {
        return abs(d1 - d2) / sqrt(a*a + b*b + c*c);
}

// distance from point p to line x y

ld DistPtLn(point p, point x, point y) {
        double pd2 = norm(x-y);
        point z;
        if (pd2 == 0) {
                z = x;
        } else {
                double u = dot(p-x, y-x) / pd2;
                z = x + (y-x) * u;
        }
        return mag(z-p);
}

// distance from point p to segment x y
ld DistPtSg(point p, point x, point y) {
        double pd2 = norm(x-y);
        point z;
        if (pd2 == 0) {
                z = x;
        } else {
                double u = dot(p-x, y-x) / pd2;
                z = x + (y-x) * u;
                if (u < 0) {
                        z = x;
                }
                if (u > 1.0) {
                        z = y;
                }
        }
        return mag(z-p);
}

//Volume of the tetrahedron defined by these three points an the origin
ld Volume(point a, point b, point c){
        return dot(a, cross(b, c))/6;
}

int main(){
        while(true){
                ld a, b, c, d, e, f, g, h, i;
                cin >> a >> b >> c >> d >> e >> f >> g >> h >> i;
                cout << Volume(point(a, b, c), point(d, e, f), point(g, h, i));
        }
}
```

3.3 2D Convex Hull

```
//graham-scan 2d convex hull, use long long for integer types
template <class T>
struct convex_hull {
        typedef pair<T,T> point;

        int N = 0;
        vector<point> pts;

        static inline bool cmp(const point &p, const point &q, const point &r, bool kr) {
                T a = q.first-p.first, b = q.second-p.second, c = r.first-p.first, d = r.second - p.
                    second, t = a*d-b*c;
                c = d*d+c*c, a = a*a+b*b;
                return t?t > 0:kr?a<c:c<a;
        }

        //add a new point
        void add_point(T x, T y) { pts.emplace_back(x,y); }

        //calculate the convex hull
        vector<point> calc(bool keep_redundant = false) {
                vector<point> hull;
                int N = pts.size();
                for(int i = 1; i < N; ++i)
                        if(pts[i] < pts[0])
                                swap(pts[i],pts[0]);
                sort(&pts[1], &pts[0]+N,[=](const point &p, const point &q) { return cmp(pts[0],p,q,
                    keep_redundant); });
                hull = {pts[0],pts[1],pts[2]};
                for(int i = 3; i < N; ++i) {
                        while(hull.size() >= 2 && !cmp(hull[hull.size()-2],hull.back(),pts[i],
                                keep_redundant))
                                  hull.pop_back();
                        hull.push_back(pts[i]);
                }
                if(!cmp(hull[hull.size()-2],hull.back(),hull[0],keep_redundant))
                        hull.pop_back();
                retrun hull;
        }
};
```

3.4 Pair of Closest Points

```
//return pair<point,point> of two closest points in p
point p[N],strip[N];
typedef pair<point,point> ppp;

double mag(point a) {
        return sqrt(a.x*a.x+a.y*a.y);
}

bool cmp(const pt& a, const pt& b) {
        return a.y < b.y;
}

inline double ds(ppp& p) {
        return (p.first==p.second)?1e200:mag(p.first-p.second);
}

//return pair<point,point> of two closest points
ppp closest(int i, int j) {
        if(i+1 == j)
                return ppp(p[i],p[i]);
        int w = 0, m = (i+j)/2;
        ppp a = closest(i,m),b = closest(m,j);
        if(ds(a) > ds(b)) swap(a,b);
        double d = ds(a);
        for(int l=i; l!=j; ++l)
                if(fabs(p[l].x-p[m].x) < d)
                        strip[w++] = p[l];
        sort(strip,strip+w,cmp);
        for(int l = 0; l < w; ++l)
                for(m = min(w-1,l+7);m!=l;--m)
                        if(mag(strip[l]-strip[m]) < d)
                                a = ppp(strip[l],strip[m]),d=ds(a);
        return a;
}
//if duplicate points, return those
ppp closest(int n) {
        sort(p,p+n);
        for(int i = 1; i < n; ++i)
                if(p[i]==p[i-1])
                        return ppp(p[i],p[i]);
        return closest(0,n);
}
```

3.5 Triangulation

```
//triangulate a polygon in O(n^2)

bool same_side(point& a, point& b, point& u, point& v) {
        return cross(b-a,u-a) > 0 == cross(b-a,v-a) > 0;
}

bool is_ear(int i, vector<point>& p, vector<int>& prv, vector<int>& nxt) {
        point& a = p[prv[i]],b = p[i],c = p[nxt[i]];
        if(cross(b-a,c-b) <= 0) return false;
        for(int j = nxt[nxt[i]]; nxt[j] != i; j=nxt[j])
                if(same_side(a,b,c,p[j]) && same_side(b,c,a,p[j]) && same_side(c,a,b,p[j]))
                        return false;
        return true;
}

//store the result in r as list of vector<int>
//each entry of r has 3 elements, the indices of the verts of the triangle it represents
void triangulate(vector<vector<int>>& r, vector<int>& p) {
        int n = p.size();
        vector<bool> ear;
        vector<int> nxt,prv;
```

```
	for(int i = 0; i < n; ++i) {
		prv.push_back((i+n-1)%n);
		nxt.push_back((i+1)%n);
	}
	for(int i = 0; i < n; ++i)
		ear.push_back(is_ear(i,p,prv,nxt));
	int i = 0;
	while(nxt[i] != prv[i]) {
		if(ear[i]) {
			r.push_back({prv[i],i,nxt[i]});
			nxt[prv[i]] = nxt[i];
			prv[nxt[i]] = prv[i];
			ear[nxt[i]] = is_ear(nxt[i],p,prv,nxt);
			ear[prv[i]] = is_ear(prv[i],p,prv,nxt);
		}
		i = nxt[i];
	}
}
```

3.6 Delaunay Triangulation

```
// slow bad nondegenerate delaunay triangulation
// Running time: O(n^4)
// INPUT:    x[] = x-coordinates
//           y[] = y-coordinates
//
// OUTPUT:   triples = a vector containing m triples of indices
//                     corresponding to triangle vertices

#include<vector>
using namespace std;

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {
	int n = x.size();
	vector<T> z(n);
	vector<triple> ret;

	for (int i = 0; i < n; i++)
	    z[i] = x[i] * x[i] + y[i] * y[i];

	for (int i = 0; i < n-2; i++) {
	    for (int j = i+1; j < n; j++) {
		for (int k = i+1; k < n; k++) {
		    if (j == k) continue;
		    double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j]-z[i]);
		    double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k]-z[i]);
		    double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j]-y[i]);
		    bool flag = zn < 0;
		    for (int m = 0; flag && m < n; m++)
			flag = flag && ((x[m]-x[i])*xn +
					(y[m]-y[i])*yn +
					(z[m]-z[i])*zn <= 0);
		    if (flag) ret.push_back(triple(i, j, k));
		}
	    }
	}
	return ret;
}

int main()
{
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

3.7 3D Convex Hull

```
//modified from an example solution of an NAIPC 2017 problem "Stars in a Can"

#define EPS (1e-8)
#define sign(x) (((x)>EPS)-((x)<(-EPS)))
#define T ld
struct vec {
	T x,y,z;    //coordinates/data
	vec(T x, T y, T z=0.):x(x),y(y),z(z){}
	vec() {x=y=z=0;}

	// vector ops
	vec& operator=(const vec& b) { x=b.x; y=b.y; z=b.z; return *this; }
	vec operator+(const vec& b) const { return vec(x+b.x, y+b.y, z+b.z); }
	vec operator-(const vec& b) const { return vec(x-b.x, y-b.y, z-b.z); }
	T operator*(const vec& b) const { return x*b.x + y*b.y + z*b.z; }
	vec operator^(const vec& b) const { return vec(y*b.z - z*b.y,z*b.x - x*b.z,x*b.y - y*b.x); }
	// scalar mult
	vec operator*(T k) const { return vec(x*k,y*k,z*k); }
	vec operator/(T k) const { return vec(x/k,y/k,z/k); }
	vec operator-() const { return vec(-x,-y,-z); }  // negation

	T sqlen() const { return (*this) * (*this); }

	bool operator<(const vec& v) const {
		if (x != v.x) return x < v.x;
		if (y != v.y) return y < v.y;
		return z < v.z;
	}
};
vec operator*(T k, vec v) { return v*k; }
#undef T

#define INSIDE (-1)
#define ON (0)
#define OUTSIDE (1)

typedef vector<vector<vec> > hull;

bool eq(ld a, ld b) {
	return abs(b-a) <= EPS;
}

ld len(const vec& a) {
	return sqrtl(a.sqlen());
}

int side(vec& a, vec& b, vec& c, vec& x) {
	vec norm = (b-a) ^ (c-a);
	vec me = x-a;
	return sign(me * norm);
}

bool is_colinear(vec& a, vec& b, vec& c) {
	vec w = (b-a)^(c-a);
	return eq(w.sqlen(),0);
}

vec projection(vec& a, vec& b, vec& c, vec& x) {
	if (side(a,b,c,x) == ON) return x;
	vec norm = (b-a) ^ (c-a);
	vec ans = x - norm * ((norm * (x-a)) / (norm * norm));
	return ans;
}

struct ph {
    size_t operator()(const pii& k) const {
	return k.first*123456789 ^ k.second*21212121;
    }
};

hull find_hull(vec* P, int N) {
	random_shuffle(P, P+N);

	// Find 4 non-degenerate points (make a tetrahedron)
	for(int j = 2; j < N; ++j)
		if(!is_colinear(P[0],P[1],P[j])) {
			swap(P[j], P[2]);
			break;
		}
	for(int j = 3; j < N; ++j)
		if (side(P[0],P[1],P[2],P[j]) != 0) {
			swap(P[j], P[3]);
			break;
		}

	// Canonicalize them
	if (side(P[0],P[1],P[2],P[3]) == OUTSIDE)
		swap(P[0], P[1]);

	vector< vector<int> > H {{0,1,2},{0,3,1},{0,2,3},{3,2,1}},H2;
	// incrementally add points
	unordered_map<pii, int, ph> D;
```

```
        for (auto & f : H)
                for(int i = 0; i < 3; ++i)
                        ++D[pii(f[i],f[(i+1)%3])];
        for(int j = 4; j < N; ++j) {
                H2.clear();
                H2.reserve(H.size());
                for (auto & f : H) {
                        int s = side(P[f[0]],P[f[1]],P[f[2]],P[j]);
                        if (s == INSIDE || s == ON) {
                                H2.push_back(f);
                        } else {
                                for(int i = 0; i < 3; ++i)
                                        --D[pii(f[i],f[(i+1)%3])];
                        }
                }

                const auto tmp = H2;
                for(auto & f : tmp) {
                        for(int i = 0; i < 3; ++i) {
                                int a = f[i],b=f[(i+1)%3];
                                if (D[pii(a,b)] + D[pii(b,a)]==1) {
                                        // add a new face
                                        H2.push_back({a, j, b});
                                        ++D[pii(a,j)];
                                        ++D[pii(j,b)];
                                        ++D[pii(b,a)];
                                }
                        }
                }
                swap(H,H2);
        }
        hull C;
        for(auto v : H)
                C.push_back({P[v[0]], P[v[1]], P[v[2]]});
        return C;
}

vec p[1001];

int main() {
        int n,T;
        cin >> T;
        cout << setprecision(10) << fixed;
        while(T--) {
                cin >> n;
                for(int i = 0; i < n; ++i)
                        cin >> p[i].x >> p[i].y >> p[i].z;
                hull h = find_hull(p,n);
                ld surface = 0, volume = 0;
                for(auto t : h) {
                        surface += len((t[1]-t[0])^(t[2]-t[0]))/2.0;
                        volume += abs(((t[1]-p[0])^(t[0]-p[0]))*(t[2]-p[0]))/6.0;
                }
                cout << surface << " " << volume << "\n";
        }
}
```

4 Graphs

4.1 Bridge Edges

```
//find bridge edges in a connected graph
struct find_bridge {
        //will contain the bridges in (u,v) pairs
        vector<pii> ans_edges;
        //will contain the bridges in indices by insertion order
        vector<int> ans_indices;

        int c=0,ctr=0;
        vector<vector<pii>> G;
        vector<bool> vis;
        vector<int> reach,lab;
        void init(int n) {
                G.resize(n);
                vis.resize(n);
                lab.resize(n);
                reach.resize(n);
        }

        find_bridge(){}
        find_bridge(int n){init(n);}

        void add_edge(int u, int v) {
                G[u].emplace_back(v,c);
                G[v].emplace_back(u,c++);
        }

        //calculate answers and store in ans_edges and ans_indices
        //ignore the return value
        int calc(int f = -1, int v = 0) {
                vis[v] = 1, reach[v] = lab[v] = ctr++;
                for(auto &p : G[v]) {
                        int w = p.first, i = p.second;
                        if(w == f) continue;
                        if(!vis[w] && calc(v,w) > lab[v])
                                ans_edges.emplace_back(v,w), ans_indices.push_back(i); //found bridge
                                        (v,w)
                        reach[v] = min(reach[v],reach[w]);
                }
                return reach[v];
        }
};
```

4.2 Dijkstra's

```
/* One source, one graph */
template <class T> //weight type
struct ijk {
        typedef pair<T,int> edge;
        typedef vector<vector<edge>> graph;
        typedef priority_queue<edge,vector<edge>,greater<edge>> pq;
        static const T INF = numeric_limits<T>::max();
        graph G;
        vector<T> d; //distances are stored here

        void init(int n) {
                G = graph(n);
                d = vector<T>(n);
        }

        void add_edge(int i, int j, T w) {
                G[i].emplace_back(w,j);
        }

        int dist(int s, int t = -1) {
                pq Q;
                fill(d.begin(),d.end(),INF);
                d[s] = 0, Q.emplace(0,s);
                while(Q.size()) {
                        edge p = Q.begin();
                        Q.pop();
                        if(p.second == t) break;
                        if(p.first != d[p.second]) continue;
                        for(const edge &e : G[p.second]) {
                                T w = p.first + e.first;
                                int v = e.second;
                                if(d[v] > w)
                                        e.emplace(d[v] = w,v);
                        }
                }
                return (~t)?d[t]:0;
        }
}

/* multiple sources, one graph
 * each  call to dist() adds another vector to d
 * e.g: if v is the third vertex queried as dist(v), then dist(v,x) = d[2][x]
 * */

template <class T> //weight type
struct ijk {
        typedef pair<T,int> edge;
        typedef vector<vector<edge>> graph;
        typedef priority_queue<edge,vector<edge>,greater<edge>> pq;

        static const T INF = numeric_limits<T>::max();
        graph G;
        vector<vector<T>> d; //distances are stored here
        int N;

        void init(int n) {
                N = n;
                G = graph(n);
        }

        ijk() {}
        ijk(int n) { init(n); }

        void add_edge(int i, int j, T w) {
                G[i].emplace_back(w, j);
        }
```

```
	int dist(int s, int t = -1) {
		pq Q;
		fill(d.begin(),d.end(),INF);
		d[s] = 0, Q.emplace(0,s);
		while(Q.size()) {
			edge p = Q.begin();
			Q.pop();
			if(p.second == t) break;
			if(p.first != d[p.second]) continue;
			for(const edge &e : G[p.second]) {
				T w = p.first + e.first;
				int v = e.second;
				if(d[v] > w)
					e.emplace(d[v] = w,v);
			}
		}
		return (~t)?d[t]:0;
	}
};
```

4.3 Euler Path

```
#include <bits/stdc++.h>
using namespace std;

//note: destroys the graph in the process
// and returns path/circuit backwards

// ----- DIGRAPH -----
void dg_euler(int v, vector<int>& path, vector<vector<int>>& G) {
	while(G[v].size()) {
		int u = G[v].back();
		G[v].pop_back();
		dg_euler(u,path,G);
	}
	path.push_back(v);
}

// ----- UNDIRECTED GRAPH -----
struct euler_path {
	vector<int> e1,e2;
	vector<vector<int>> G;
	vector<bool> used = {};

	void init(int N){G.resize(N);}

	inline void add_edge(int u, int v) {
		G[u].push_back(e1.size()), G[v].push_back(e2.size());
		e1.push_back(u), e2.push_back(v);
		used.push_back(0);
	}

	void get_path(vector<int> &path, int v = 0) {
		while(G[v].size()) {
			int i = G[v].back();
			G[v].pop_back();
			if(used[i]) continue;
			used[i] = 1;
			int u = (v==e1[i])?e2[i]:e1[i];
			get_path(path,u);
		}
		path.push_back(v);
	}

	euler_path(int N){init(N);}
};


// ----- USAGE -----
vector<vector<int>> G;
int main() {
	euler_path e(10);
	// add edges
	vector<int> path;
	e.get_path(path);

	//DIRECTED CASE
	G = {{2,3},{0},{1},{4},{0}};
	path.resize(0);
	dg_euler(1,path,G);
	for(auto u : path)
		cout << u << "\n";
	cout << endl;
}
```

4.4 Kruskal's Algorithm

```
template <class T>
struct kruskal {
	struct edge {
		int a,b;
		T w;
		bool in_mst;
		bool operator<(const edge& e) const { return w < e.w; }
	};

	vector<edge> E;

	void init(int n) { uf.resize(n,-1); }
	kruskal(int n) { init(n); }

	void add_edge(int a, int b, T w) { E.push_back((edge){a,b,w,false}); }

	T mst() {
		sort(E.begin(),E.end());
		T w = 0;
		for(edge &e : E) {
			int a = id(e.a), b = id(e.b);
			if(a == b) continue;
			w += e.w;
			e.in_mst = true;
			uf[a] = b;
		}
		return w;
	}

	vector<int> uf;
	int id(int u) { return (~uf[u]) ? uf[u] = id(uf[u]) : u; }
};
```

4.5 Strongly-connected Components

```
//find strongly connected components in a graph
//scc ids are indexed topologically, e.g.  edge a -> b implies scc[a] <= scc[b]
struct scc {
	int sz=0; //the number of sccs
	vector<int> id; //id[v] is the scc id of v
	int N,ls=0;
	vector<int> L;
	vector<vector<int>> G,R;

	void init(int n) { N = n, G.resize(N), R.resize(N), id.resize(N), L.resize(N); }

	scc(){}
	scc(int n){init(n);}

	void dfs1(int v) {
		if(id[v]) return;
		id[v] = 1;
		for(auto u : G[v])
			dfs1(u);
		L[ls++] = v;
	}

	void dfs2(int v, int r) {
		if(~id[v]) return;
		id[v] = r;
		for(int u : R[v])
			dfs2(u,r);
	}

	void add_edge(int u, int v) {
		G[u].push_back(v), R[v].push_back(u);
	}

	//calculate the strongly connected components
	void calc() {
		for(int v = 0; v < N; ++v)
			dfs1(v);
		fill(id.begin(),id.end(),-1);
		for(int i = N-1; i >= 0; --i) {
			if(id[L[i]] == -1)
				dfs2(L[i],sz++);
		}
	}

	//get the digraph of sccs, call AFTER calc
	//remember that indices are ordered topologically
	vector<vector<int>> scc_digraph() {
```

```
		vector<vector<int>> B(N);
		for(int u = 0; u < N; ++u)
			for(int v : G[u])
				if(id[u] != id[v])
					B[id[u]].push_back(id[v]);
		for(int i = 0; i < sz; ++i) {
			sort(B[i].begin(),B[i].end());
			B[i].erase(unique(B[i].begin(),B[i].end()),B[i].end());
		}
		return B;
	}
};
```

4.6 Topological Sorting

```
struct top_sort {
	vector<vector<int>> G; //stores G; take it out if you want to use it elsewhere
	vector<int> order; //stores the topological sort
	vector<short> seen;
	void init(int n) { G.resize(n); seen.resize(n); }
	top_sort(int n) { init(n); }

	//add a new edge
	inline void add_edge(int u, int v) { G[u].push_back(v); }

	bool visit(int u) {
		if(seen[u] == 2) return true;
		if(seen[u] == 1) return false;
		for(int v : G[u])
			if(!visit(v))
				return false;
		seen[u] = 2;
		order.push_back(u);
		return true;
	}
	//topologically sort
	bool sort() {
		for(int i = 0; i < G.size(); ++i)
			if(!visit(i))
				return false;
		reverse(order.begin(),order.end());
		return true;
	}
};
```

5 Math

5.1 Floating-Point Matrix

```
typedef vector<vector<double> > matrix;
constexpr double EPS = 1e-10;

//rref matrix, return determinant
double rref(matrix &M) {
	int n = M.size(), m = M[0].size(), r = 0;
	double det = 1;
	for(; r < min(n,m); ++r) {
		int i = r;
		for(; i < n && abs(M[i][r]) < EPS; ++i);
		if(i == n) break;
		if(i != r) swap(M[i],M[r]), det = -det;
		double v = 1.0/M[r][r];
		det = det * M[r][r];
		M[r][r] = 1;
		for(int j = r+1; j < m; ++j)
			M[r][j] = M[r][j] * v;
		for(i = 0; i < n; ++i) {
			if(i == r) continue;
			for(int j = m-1; j >= r; --j)
				M[i][j] = M[i][j] - M[i][r] * M[r][j];
		}
	}
	return det * (n == m && n == r);
}
```

5.2 Finite Field Matrix

```
typedef vector<vector<ll>> matrix;

ll pow(const ll a, const ll b, const ll P) {
	if(!b) return 1;
	if(b&1) return a * pow(a,b-1,P) % P;
	ll t = pow(a, b/2, P);
	return t * t % P;
}

//reduce matrix mod P
ll rref(matrix &M, const ll P) {
	int n = M.size(), m = M[0].size(), r = 0;
	ll det = 1;
	for(; r < min(n,m); ++r) {
		int i = r;
		for(; i < n && !M[i][r]; ++i);
		if(i == n) break;
		if(i != r) swap(M[i],M[r]), det = P - det;
		ll v = pow(M[r][r],P-2,P);
		det = det * M[r][r] % P;
		M[r][r] = 1;
		for(int j = r+1; j < m; ++j)
			M[r][j] = M[r][j] * v % P;
		for(i = 0; i < n; ++i) {
			if(i == r) continue;
			for(int j = m-1; M[i][r]; --j) {
				M[i][j] = (M[i][j] - M[i][r] * M[r][j]) % P;
				if(M[i][j] < 0) M[i][j] += P;
			}
		}
	}
	return det * (n == m && n == r);
}

//add matrices mod P
matrix add(const matrix& A, const matrix& B, ll P) {
	int n = A.size(), m = A[0].size();
	matrix C(n,vector<ll>(m));
	for(int i = 0; i < n; ++i)
		for(int j = 0; j < m; ++j) {
			C[i][j] = A[i][j] + B[i][j];
			if(C[i][j] >= P) C[i][j] -= P;
		}
	return C;
}

//multiply matrices mod P
matrix multiply(const matrix& A, const matrix& B, ll P) {
	int n = A.size(), m = A[0].size(), l = B[0].size();
	matrix C(n,vector<ll>(l));
	for(int i = 0; i < n; ++i)
		for(int j = 0; j < l; ++j)
			for(int k = 0; k < m; ++k)
				C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % P;
	return C;
}

//matrix exponentials mod p
matrix pow(const matrix& A, ll e, ll P) {
	int n = A.size();
	matrix C = matrix(n, vector<ll>(n));
	for(int i = 0; i < n; ++i)
		C[i][i] = 1;
	matrix W = A;
	while(e) {
		if(e&1) C = multiply(C,W,P);
		e >>= 1;
		W = multiply(W,W,P);
	}
	return C;
}
```

5.3 Primes

```
//factorize integers in [0,N)
int fact[N];
void factor_all() {
	for(int i = 1; i < N; ++i)
		fact[i] = i;
	for(int i = 2; i < N; ++i)
		if(fact[i] == i)
			for(ll j = 1LL * i * i; j < N; j += i)
				fact[j] = i;
}

//factored list of integers
map<int,int> factor(int n) {
	map<int,int> a;
```

```
	while(n != 1) {
		++a[fact[n]];
		n/=fact[n];
	}
	return a;
}

//return list of divisors given factorization
vector<int> divisors(const map<int,int>& f) {
	int m = 1;
	for(auto p : f)
		m *= p.second+1;
	vector<int> ans;
	for(--m; m>=0; --m) {
		int w=m,a=1;
		for(auto p : f) {
			int e = w%(p.second+1);
			w /= p.second+1;
			while(e--)
				a*=p.first;
		}
		ans.push_back(a);
	}
	return ans;
}

// returns <x, y, gcd(a,b)> such that ax + by = gcd(a,b)
tuple<ll,ll,ll> extended_euclid(ll a, ll b) {
	ll s = 0, x = 1, t = 1, y = 0, r = b, g = a, tmp;
	while (r) {
		tmp = x - (g / r) * s; x = s; s = tmp;
		tmp = y - (g / r) * t; y = t; t = tmp;
		tmp = g % r; g = r; r = tmp;
	}
	if (a < 0) x = -x, y = -y, g = -g;
	return make_tuple(x, y, g);
}

bool miller_rabin_primality(ll N) {
	// deterministic for all <= 2 ^ 64
	static const int p[12] = {2,3,5,7,11,13,17,19,23,29,31,37};
	if (N <= 1) return false;
	for (int i = 0; i < 12; ++i) {
		if (p[i] == N) return true;
		if (N % p[i] == 0) return false;
	}
	ll c = N - 1, g = 0;
	while (!(c & 1)) c >>= 1, ++g;
	for (int i = 0; i < 12; ++i) {
		ll k = modpow(p[i], c, N);
		for (int j = 0; j < g; ++j) {
			ll kk = modmul(k, k, N);
			if (kk == 1 && k != 1 && k != N - 1) return false;
			k = kk;
		}
		if (k != 1) return false;
	}
	return true;
}

mt19937 gen(time(0));
ll pollard_rho(ll N) {
	if (N % 2 == 0) return 2;
	ll xx = uniform_int_distribution<ll>()(gen) % N, x = xx;
	ll c = uniform_int_distribution<ll>()(gen) % N, d = 1;
	for (int iters = 0; iters < 2000; ++iters) {
		x = (modmul(x, x, N) + c) % N;
		xx = (modmul(xx, xx, N) + c) % N;
		xx = (modmul(xx, xx, N) + c) % N;
		d = __gcd(abs(x - xx), N);
		if (d != 1 && d != N) break;
	}
	return d;
}
```

6 Miscellaneous

6.1 2-SAT

```
// variables in a clause are represented in the a,b arrays as follows:
// variables range from 0 to n-1.
// a non-negated variable v is 2v+1, negated is 2v

// the formula is (a[0] OR b[0]) AND ... AND (a[c] OR b[c])
// solve_2sat returns whether the formula is satisfiable
// if satisfiable, afterwards ans[v]=1 iff v is true in the satisfying assignment found

//maximum number of distinct variables in your statement
struct two_sat {
	vector<bool> vis = {};
	int cz=0,N;
	vector<int> cc,L,a,b;
	vector<vector<int>> G,R;

	two_sat(){}
	two_sat(int n):N(n),G(2*n),R(2*n),vis(2*n),cc(2*n){}

	void visit(int v) {
		vis[v] = 1;
		for(auto u : G[v])
			if(!vis[v])
				visit(u);
		L.push_back(v);
	}

	inline void assign(int v, int r, vector<bool>& a) {
		if(cc[v]) return;
		cc[v] = r;
		for(auto u : R[v])
			assign(u,r,a);
		a[v/2]=v&1;
	}

	inline void add(int x1, bool v1, int x2, bool v2) {
		x1 = x1<<1|v1, x2 = x2<<1|v2;
		G[x1^1].push_back(x2);
		G[x2^1].push_back(x1);
		R[x1].push_back(x2^1);
		R[x2].push_back(x1^1);
	}

	bool solve(vector<bool>& ans) {
		ans.resize(N);
		for(int i = 0; i < 2*N; ++i)
			if(!vis[i]) visit(i);
		vis = vector<bool>(N);
		for(int i = L.size()-1; i >= 0; --i) {
			if(!cc[L[i]])
				assign(L[i],++cz,ans);
		}
		for(int i = 0; i < N; ++i)
			if(cc[2*i]==cc[2*i+1])
				return false;
		return true;
	}
};
```

6.2 Convex Hull Trick

```
// min-convex hull trick
// for max, invert m, b in add_line and result of query_min
template <typename T>
struct convex_hull_trick {
	vector<double> start;
	int n = 0;
	vector<T> M, B;

	double meet(T m1, T b1, T m2, T b2) {
		return double(b2-b1)/(m1-m2);
	}

	//add mx + b in order of DECREASING slope
	//currently does not handle same-slope
	void add_line(T m, T b) {
		if(n == 0) {
			start.push_back(-1e300);
		} else {
			double mt;
			while((mt = meet(M[n-1], B[n-1], m, b)) <= start[n-1]) {
				start.pop_back(), M.pop_back(), B.pop_back(), --n;
			}
			start.push_back(mt);
		}
		M.push_back(m), B.push_back(b), ++n;
	}

	// get min of mx+b over all (m, b) lines given
	T query_min(T x) {
		int lo = 0, hi = n;
		while(hi-lo-1) {
			int md = (lo+hi)/2;
			if(x >= start[md]) lo = md;
			else hi = md;
```

```
		}
		return M[lo]*x+B[lo];
	}
};
```

6.3 Divide and Conquer Optimization

```
//initialize me
int L,R;
ll dp[K][N], A, INF = 1<<62;

//remove array[i] from the range
inline void sl_rem(int i) {
	// A -= a[i];
}

//add array[i] to the range
inline void sl_add(int i) {
	// A += a[i];
}

inline ll slide(int l, int r) {
	while(l < L) sl_add(--L);
	while(R < r) sl_add(++R);
	while(L < l) sl_rem(L++);
	while(r < R) sl_rem(R--);
	//return A
}

// dp[k][i] = MAX{j <= i} dp[k-1][j-1] + cost(j .. i)
// ONE INDEX YOUR ARRAY, AND SET dp[?][0] = 0
void compute(int k, int l, int r, int bl, int br) {
	int m = (l+r)/2, opt;
	dp[k][m] = INF;
	for(int i = bl; i <= min(br,m); ++i) {
		slide(i,m);
		ll cost = A + dp[k-1][i-1];
		if(cost < dp[k][m])
			dp[k][m] = cost, opt = i;
	}
	if(l <= m-1) compute(k,l,m-1,bl,opt);
	if(m+1 <= r) compute(k,m+1,r,opt,br);
}
```

6.4 Fast C++ Input

```
#define gc() getchar_unlocked()
#define pc(x) putchar_unlocked(x)

inline void read(int& a) {
	char c = gc();
	while(c == ' ' || c == '\n') c = gc();
	a = 0;
	bool neg = c == '-';
	if(c == neg) c = gc();
	while('0' <= c && c <= '9') {
		a = a*10 + c-'0';
		c = gc();
	}
}

inline void read(string& s) {
	static int bz = 100005;
	static char *bf = new char[bz];
	int z = 0;
	char c = gc();
	while(c == ' ' || c == '\n') c = gc();
	while(c == ' ' || c == '\n') {
		if(++z == bz) {
			char *tmp = new char[2*bz];
			copy(bf,bf+bz,tmp);
			delete[] bf;
			bf = tmp;
		}
		bf[z-1] = c;
		c = gc();
	}
	bf[z] = 0;
	s = bf;
}


inline void print(int a) {
	char bf[12];
	int n = 0;
	if(a==0) bf[n++]='0';
	else while(a) bf[n++] = '0'+a%10, a/=10;
	while(n--) pc(bf[n]);
}

inline void print(const string &s) {
	for(char c : s) pc(c);
}
```

6.5 Longest Increasing Subsequence

```
//find length of longest increasing subsequence
int lis(int *A, int n) {
	if(!n) return 0;
	vector<int> v = {0};
	for(int i = 1; i < n; ++i) {
		if(A[i] > A[v.back()]) {
			v.push_back(i);
		} else {
			int lo = -1, hi = v.size()-1;
			while(hi-lo>1) {
				int md = (lo+hi)/2;
				((A[i] > A[v[md]])?lo:hi) = md;
			}
			v[hi] = i;
		}
	}
	return v.size();
}



//construct longest increasing subsequence
//store results in L (assumed to be empty)
int lis(int *A, int n, vector<int> &L) {
	if(!n) return 0;
	vector<int> v = {n-1}, p = {-1};
	for(int i = n-2; i >= 0; --i) {
		if(A[i] < A[v.back()])
			p[i] = v.back(), v.push_back(i);
		else {
			int lo = -1, hi = v.size()-1;
			while(hi-lo>1) {
				int md = (lo+hi)/2;
				((A[i] < A[v[md]])?lo:hi) = md;
			}
			p[i] = (lo == -1)?-1:v[lo];
			v[hi] = i;
		}
	}
	L.reserve(v.size());
	for(int u = v.back(); ~u; u = p[u])
		L.push_back(u);
	return L.size();
}
```

6.6 Java

```
import java.util.*;
import java.math.*;
import java.io.*;

class FastReader {
	BufferedReader br;
	StringTokenizer st;

	public FastReader() {
		br = new BufferedReader(new InputStreamReader(System.in));
	}

	String next() throws Exception{
		if (st == null || !st.hasMoreElements())
			st = new StringTokenizer(br.readLine());
		return st.nextToken();
	}
	int nextInt() throws Exception { return Integer.parseInt(next()); }
	long nextLong() throws Exception{ return Long.parseLong(next()); }
	double nextDouble() throws Exception { return Double.parseDouble(next()); }
	String nextLine() throws Exception { return br.readLine(); }
}
```

```java
class FastWriter {
        private final BufferedWriter bw;
        public FastWriter() {
                bw = new BufferedWriter(new OutputStreamWriter(System.out));
        }
        public void print(Object object) throws Exception {
                bw.append("" + object);
        }
        public void close()throws IOException
        {
            bw.close();
        }
}

public class Main {
        public static void main(String[] args) throws Exception {
                FastReader fr = new FastReader();
                BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));

                int n = fr.nextInt();
                String s = fr.next(), t = fr.next();

                BigInteger two = new BigInteger("2");
                BigInteger three = two.add(BigInteger.ONE);
                // other BigInteger methods: add(), subtract(), mod(), pow(int), divide()
                int five = (two.add(three)).intValue();
                if(two.compareTo(three) < 0) {
                        bw.append("saw " + s + "\n");
                        bw.append("saw " + t + "\n");
                }
                bw.append(two + "\n");

                ArrayList<Integer> arr = new ArrayList<>();
                arr.add(5);
                arr.add(1);
                Collections.sort(arr);

                bw.close(); // flushes the buffer
        }
}

class Pair implements Comparable<Pair>{
        int a, b;
        Overridepublic int compareTo(Pair x) if(a != x.a)return Integer.compare(x);return
              Integer.compare(b);
```

6.7 Other

```cpp
#include <ext/rope>
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <ext/pb_ds/tree_policy.hpp> // Including tree_order_statistics_node_update

//SPECIAL GNU
__gcd(a, b) //GCD : do not use let a or b be 0
__builtin_popcount(a) //number of 1 bits
__builtin_clz(a) // count leading zeroes
inline int log2(int a) { return 31-__builtin_clz(a); } //floor(log(a))

//ORDER STATISTIC TREE
typedef tree<int, null_type, less<int>,
        rb_tree_tag, tree_order_statistics_node_update> ordered_set;

ordered_set X;
X.insert(1);
cout << *X.find_by_order(0) << endl;
cout << X.order_of_key(1) << endl;

//ROPE
rope <int> v; //use as usual STL container
int n, m;
cin >> n >> m;
for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
int l, r;
for(int i = 0; i < m; ++i) {
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);
}
for(rope <int>::iterator it = v.mutable_begin(); it != v.mutable_end(); ++it)
        cout << *it << " ";

//REGEX
regex r("^[^f]*$");
string result = regex_replace(string start, regex match, string replace);
// regex_match matches the whole string, whereas regex_search checks to see if it matches a substring
```

7 Strings

7.1 Suffix Array

```cpp
// O(n log^2 n) suffix array
// return_val[i] is the starting index of the i-th sorted suffix
vector<int> suff_arr(const string& s) {
        const int n = s.size();
        vector<int> source(s.begin(), s.end()), tmp(n), order(n);
        for(int i = 0; i < n; ++i)
                order[i] = i;
        int gap = 0;
        auto cmp = [&](int i, int j) -> bool {
                if(source[i] != source[j])
                        return source[i] < source[j];
                const int a = (i+gap >= n) ? -1 : source[i+gap];
                const int b = (j+gap >= n) ? -1 : source[j+gap];
                return a < b;
        };
        for(gap = (n > 1); gap < n; gap += max(1, gap)) {
                sort(order.begin(), order.end(), cmp);
                int ctr = tmp[order[0]] = 0;
                for(int i = 1; i < (int)order.size(); ++i) {
                        ctr += cmp(order[i],order[i-1]) || cmp(order[i-1], order[i]);
                        tmp[order[i]] = ctr;
                }
                swap(source, tmp);
        }
        return order;
}

// O(n log n) kasai algorithm
// return_val[i] is LCP(suff_arr[i], suff_arr[i+1])
vector<int> lcp_arr(const string& s, const vector<int>& suff_arr) {
        const int n = s.size();
        vector<int> lcp(n), inv(n);
        for(int i = 0; i < n; ++i)
                inv[suff_arr[i]] = i;
        int k = 0;
        for(int i = 0; i < n; ++i) {
                if(inv[i] == n-1) {
                        k = 0;
                } else {
                        const int j = suff_arr[inv[i]+1];
                        while(max(i,j)+k < n && s[i+k] == s[j+k])
                                ++k;
                        lcp[inv[i]] = k;
                        k -= (k>0);
                }
        }
        return lcp;
}
```

7.2 KMP Algorithm

```cpp
struct KMP {
        int N;
        const string S;
        vector<int> F;

        KMP(const string S): S(S) {
                N = S.size();
                F.assign(N + 1, 0);
                for (int i = 1; i < N; ++i)
                F[i + 1] = advance(F[i], S[i]);
        }

        int advance(int j, char x) {
                while (j && (j >= N || S[j] != x)) j = F[j];
                if (S[j] == x) ++j;
                return j;
        }

        void match(const string T) {
                int j = 0;
                for (auto c : T) {
```

```
                    j = advance(j, c);
                    if (j == N) cout << "match" << endl;
            }
    }

    int minfactor() {
            int factor = N;
            for(int i = F[N]; i; i = F[i])
                    if(N % (N - i) == 0) {
                            factor = N - i;
                            break;
                    }
            return factor;
    }
};
```

7.3 String Hashing

```
struct str_hash {
        vector<int> h1,h2;
        static vector<int> b1,b2;
        constexpr static int B = 31, M1 = 1e9+7, M2 = 1e9+9;
        str_hash(const string& s) {
                h1.resize(s.size()+1,0);
                h2.resize(s.size()+1,0);
                b1.reserve(s.size()+1), b2.reserve(s.size()+1);
                while(b1.size() <= s.size()) {
                        b1.push_back(1LL * b1.back() * B % M1);
                        b2.push_back(1LL * b2.back() * B % M2);
                }
                for(int i = 0; i < s.size(); ++i) {
                        h1[i+1] = (1LL * h1[i] * B + s[i]) % M1;
                        h2[i+1] = (1LL * h2[i] * B + s[i]) % M2;
                }
        }

        ll hash(int i, int j) const {
                ll a1 = (h1[j+1] - 1LL * h1[i] * b1[j-i+1]) % M1;
                ll a2 = (h2[j+1] - 1LL * h2[i] * b2[j-i+1]) % M2;
                if(a1 < 0) a1 += M1;
                if(a2 < 0) a2 += M2;
                return a1 ^ (a2<<32);
        }

        //for one time entire-string hashing
        static ll static_hash(const string& s) {
                ll a1=0,a2=0;
                for(char c : s) {
                        a1 = (a1 * B + c) % M1;
                        a2 = (a2 * B + c) % M2;
                }
                return a1 ^ (a2<<32);
        }
};
vector<int> str_hash::b1={1},str_hash::b2={1};
```

7.4 Z-Algorithm

```
//v[i] is LCP(s, suffix of s starting at i)
vector<int> z_algo(const string &s) {
        int n = s.size(), l = 0, r = 0;
        vector<int> Z(n);
        for(int i = 1; i < n; ++i) {
                if(i <= r && Z[i-l] < r-i+1)
                        Z[i] = Z[i-l];
                else {
                        l = i, r = max(r,i);
                        while(r < n && s[r-l] == s[r]) ++r;
                        Z[i] = r---l;
                }
        }
        Z[0] = n;
        return Z;
}
```

8 Transforms

8.1 FFT

```
//FFT1 for less precise
#include <bits/stdc++.h>
using namespace std;

//USAGE: FFT::fft(A, B) for vector<int> A and B of coefs
// double --> ld is about 30% slower
typedef long double ld;
namespace FFT {
        struct base {
                ld re,im;
                base(ld r=0, ld i=0):re(r),im(i){}
                inline base operator*(const base& b) const {
                        return base(re*b.re-im*b.im,re*b.im+b.re*im);
                }
                inline base operator-(const base& b) const {
                        return base(re-b.re,im-b.im);
                }
                inline base operator+(const base& b) {
                        return base(re+b.re,im+b.im);
                }
                inline void operator+=(const base& b) {
                        re += b.re, im += b.im;
                }
                inline void operator*=(const base& b) {
                        ld r = re;
                        re = re*b.re-im*b.im;
                        im = r*b.im+b.re*im;
                }
                inline void operator/=(int b) {
                        re /= b;
                        im /= b;
                }
                inline base operator/(int b) const {
                        return base(re/b,im/b);
                }

                inline base conj() const {
                        return base(re,-im);
                }
        };

        vector<int> rev;
        vector<base> wlen_pw;
        void fft(base a[], int n, bool invert) {
                static int last_n = 0;
                static const ld two_pi = acosl(0)*4;
                if(n != last_n) {
                        if(n > last_n) rev.resize(n), wlen_pw.resize(n);
                        last_n = n;
                        int log_n = 31-__builtin_clz(n);
                        for(int i = 0; i < n; ++i) {
                                rev[i] = 0;
                                for(int j = 0; j < log_n; ++j)
                                        if(i & (1<<j))
                                                rev[i] |= 1<<(log_n-1-j);
                        }
                }
                for(int i = 0; i < n; ++i)
                        if(i < rev[i])
                                swap(a[i], a[rev[i]]);
                for(int len=2; len<=n; len<<=1) {
                        ld ang = two_pi/len * (invert?-1:+1);
                        int len2 = len>>1;
                        base wlen(cos(ang), sin(ang));
                        wlen_pw[0] = base(1);
                        for(int i = 1; i < len2; ++i)
                                wlen_pw[i] = wlen_pw[i-1] * wlen;
                        for(int i = 0; i < n; i += len) {
                                base t, *pu = a+i, *pv = a+i+len2,
                                        *pu_end = a+i+len2, *pw = &wlen_pw[0];
                                for(; pu!=pu_end; ++pu, ++pv, ++pw)
                                        t = *pv * *pw, *pv = *pu - t, *pu += t;
                        }
                }
                if(invert)
                        for(int i=0; i<n; ++i)
                                a[i] /= n;
        }

        vector<ll> multiply (const vector<int> &a, const vector<int> &b) {
                vector<ll> res;
                vector<base> P(max(a.size(),b.size())),Q;
```

```
		for(size_t i = 0; i < a.size(); ++i)
			P[i].re = a[i];
		for(size_t i = 0; i < b.size(); ++i)
			P[i].im = b[i];
		size_t n = 2;
		while ((n>>1) < P.size()) n <<= 1;
		P.resize(n), Q.resize(n);
		fft(&P[0], n, false);
		const base rot(0,-0.25);
		for(size_t i = 0; i != n; ++i) {
			base tmp = P[i?n-i:0].conj();
			Q[i] = (P[i]+tmp)*(P[i]-tmp)*rot;
		}
		fft(&Q[0], n, true);
		res.resize(n);
		for(size_t i = 0; i != n; ++i)
			res[i] = llrint(Q[i].re);
		return res;
	}
}

int main() {
	vector<int> A = {1, -3, 1}    // x^2 - 3x + 1
	vector<int> B = {-7, 2, 0, 2} // -2x^3 + 2x - 7
	vector<ll> C = FFT:multiply(A, B); // higher coefs might be zero
}
```

8.2 NTT

```
// store polynomials in a,b. set LA = len(a) and L = len(b)
// call calculate(), a now stores a*b.
const int maxn=4000000;
const int g=3,bigp=479*(1<<21)+1,x=21;
int LA,LB;
int a[maxn]={0},b[maxn]={0},w[maxn];
int C,N,L;
int powc(int a,int b) {
	if (!b) return 1;
	int d = powc(a,b/2);
	d = ll(d)*d%bigp;
	if(b%2) d = ll(d)*a%bigp;
	return d;
}
//K is the length of x[]
//v=0 : DFT, v=1 : IDFT
void FFT(int x[],int K,int v){
	w[0] = 1;
	int G = powc(g,(bigp-1)/K);
	for(int i = 0; i < K; ++i)
		w[i+1] = (ll)w[i]*G%bigp;
	for(int i=0,j=0; i<K; ++i) {
		if(i>j) swap(x[i],x[j]);
		for(int l=K>>1; (j^=l)<l; l>>=1);
	}
	for(int i = 2; i <= K; i<<=1)
		for(int j = 0; j < K; j += i)
			for(int l = 0; l < i>>1; l++) {
				int t = (ll)x[j+l+(i>>1)]*w[v?K-(K/i)*l:(K/i)*l]%bigp;
				x[j+l+(i>>1)] = ((ll)x[j+l]-t+bigp)%bigp;
				x[j+l] += t;
				x[j+l] %= bigp;
			}
	if(v) {
		int r = powc(N,bigp-2);
		for(int i = 0; i < N; ++i)
			a[i] = ll(a[i])*r%bigp;
	}
}
void calculate() {
	//LA,LB are length of a[] and b[] (include a[0] and b[0])
	N=1,C=LA+LB,L=0;
	while (N<=C) N*=2,++L;
	for(int i=LA; i<N; i++) a[i] = 0;
	for(int i=LB; i<N; i++) b[i] = 0;
	FFT(a,N,0);FFT(b,N,0);
	for (int i=0; i<N; i++)
		a[i]=(ll)a[i]*b[i]%bigp;
	FFT(a,N,1);
}
```

8.3 FHWT and Similar

```
/* matrices for bit operations and their inverses (fast-walsh-hadamard)
C_k = SUM over op(i,j) = k of A_i * B_i
the following operations are supported:

# XOR #
    |1  1|             |1  1|
T = |1 -1| and 2T^-1 = |1 -1|
note: there is a scale factor of 2
FORWARD TRANSFORM: a=a+b,b=a-2*b;
REVERSE TRANSFORM: same as first

# OR #
    |1  1|            |0  1|
T = |1  0| and T^-1 = |1 -1|
FORWARD TRANSFORM: a=a+b, b=a-b;
REVERSE TRANSFORM: b=a-b, a=a-b;


# AND #
    |0  1|            |-1 1|
T = |1  1| and T^-1 = |1  0|
FORWARD TRANSFORM: b=a+b, a=b-a;
REVERSE TRANSFORM: a=b-a, b=b-a;
*/


// fast walsh-hadamard-like transform
// n should be a power of 2
void FWHT(ll* d, int n) {
	for(int g = 1; g*2 <= n; g *= 2)
		for(int i = 0; i < n; i += 2*g)
			for(int j = i; j < i+g; ++j) {
				ll &a = d[j], &b = d[j+g];
				//PASTE RELEVANT LINE HERE
			}
}


const int N = 8;
int main() {
	ios::sync_with_stdio(0); cin.tie(0);
	//find P^77 under XOR tranform
	ll h[N] = {0,1,2,3,4,5,6,7};
	FWHT(h,N);
	for(int i = 0; i < N; ++i) h[i] = pw(h[8],77);
	FWHT(h,N); //needs seperate inverse if not xor
	for(int i = 0; i < N; ++i) h[i] /= N //only need if xor, might need mod arith

	return 0;
}
```

9 Trees

9.1 Centroid Decomposition

```
vvi T;

const int N = 1e5+5;
int cdt_sz[N]={};
int cdt_parent[N];
vi cdt_children[N];

int cdt_fsz(int u, int v) {
	if(cdt_sz[v] == -1) return 0;
	cdt_sz[v] = 1;
	for(int w : T[v])
		if(w != u)
			cdt_sz[v] += cdt_fsz(v,w);
	return cdt_sz[v];
}

//call cdt_build(0, tree_size)
//cdt_parent of the centroid root is -1
int cdt_build(int v, int n, bool rc = 1) {
	if(rc) cdt_fsz(-1,v);
	int p = -1;
	while(1) {
		int x = -1;
		for(int w : T[v])
			if(x == -1 || cdt_sz[w] > cdt_sz[x])
				x = w;
		if(cdt_sz[x] * 2 <= n) break;
		cdt_sz[v] = n-cdt_sz[x];
		p = v, v = x;
	}
	cdt_sz[v] = cdt_parent[v] = -1;
```

```
	for(int w : T[v]) {
		if(cdt_sz[w] == -1) continue;
		int x = cdt_build(w,cdt_sz[w],w == p);
		//found cdt edge (v -> x)
		cdt_children[v].push_back(x);
		cdt_parent[x] = v;
	}
	return v;
}
```

9.2 Heavy-Light Decomposition

```
vvi T;

// heavy-light decomposition for general range query structures
// for edge update (u,v), simply update MAX(hld_ind[u],hld_ind[v])
int hld_sz[N], hld_ind[N], hld_parent[N], hld_cstart[N], hld_pr[N], hld_ctr;

int hld_fsz(int v, int u = -1) {
	hld_sz[v] = 1;
	for(int w : T[v])
		if(w != u)
			hld_sz[v] += hld_fsz(w,v);
	return hld_sz[v];
}

void hld(int v = 0, int u = -1, int head = 0, int d = 0) {
	if(u == -1) hld_ctr=0,hld_fsz(v);
	else hld_parent[hld_ctr] = hld_ind[u];
	hld_cstart[hld_ctr] = head;
	hld_pr[hld_ctr] = d;
	hld_ind[v] = hld_ctr++;
	int lg = -1;
	for(int w : T[v])
		if(w != u && (lg == -1 || hld_sz[w] > hld_sz[lg]))
			lg = w;
	if(lg != -1) hld(lg,v,head,d);
	for(int w : T[v]) {
		if(w == u || w == lg) continue;
		hld(w,v,hld_ctr,d+1);
	}
}

//remember to make queries inclusive
void path_query(int u, int v) {
	u = hld_ind[u], v = hld_ind[v];
	while(hld_cstart[u] != hld_cstart[v]) {
		if(hld_pr[u] < hld_pr[v]) swap(u,v);
		// combine : QUERY [hld_cstart[u], u]
		u = hld_parent[hld_cstart[u]];
	}
	if(u > v) swap(u,v);
	//combine (vert query): QUERY [u,v]
	//combine (edge query): QUERY [u+1,v]
	return mx;
}
```

9.3 LCA

```
struct lca {
	void init(int n) { T.resize(N = n); }
	lca(){}
	lca(int n){init(n);}
	int N,ctr=0;
	vector<int> idx,depth;
	vector<vector<int>> table,T;
	static inline int lg(int a) {return 31-__builtin_clz(a);}
	void dfs(int u, int v, int d) {
		idx[v] = ctr;
		depth[v] = d, table[0][ctr++] = v;
		for(int i = 0; i < (int)T[v].size(); ++i)
			if(T[v][i] != u)
				dfs(v,T[v][i],d+1), table[0][ctr++] = v;
	}

	void add_edge(int u, int v) {
		T[u].push_back(v);
		T[v].push_back(u);
	}

	//call build after edges are added
	void build(int root = 0) {
		idx.resize(N), depth.resize(N), table.assign(lg(2*N-1)+1,vector<int>(2*N-1));
		dfs(-1,root,0);
		for(int k = 0; k+1 < (int)table.size(); ++k) {
			for(int i = 0; i < (int)table[k].size(); ++i) {
				int j = min(i+(1<<k),(int)table[k].size()-1);
				table[k+1][i] = (depth[table[k][i]] < depth[table[k][j]])?table[k][i]:
					table[k][j];
			}
		}
	}

	//lca(u,v)
	inline int query(int u, int v) const {
		u = idx[u], v = idx[v];
		if(v < u) swap(u,v);
		int g = lg(v-u+1);
		u = table[g][u], v = table[g][v+1-(1<<g)];
		return (depth[u] < depth[v])?u:v;
	}

	//unweighted distance between u and v
	inline int dist(int u, int v) {
		return depth[u] + depth[v] - 2*depth[query(u,v)];
	}
};
```