TEMPLATE

```cpp
#include <bits/stdc++.h>
using namespace std;
//typedefs
typedef long long ll;
typedef unsigned long long ull;
typedef pair <int, int> pii;
typedef pair <int, pii> piii;
typedef vector <int> vi;
typedef vector <ll> vl;
typedef pair <ll, ll> pll;
const double PI = acos(-1);
//defines
#define MP make_pair
#define PB push_back
#define F first
#define S second
#define mem(a, b) memset(a, b, sizeof(a))
#define gcd(a,b) __gcd(a,b)
#define lcm(a,b) (a*(b/gcd(a,b)))
#define sqr(a) ((a)*(a))
#define inf 100000000
#define mod 1000000007
#define mod1 1000000007
#define mod2 1000000009
#define b1 43
#define b2 41
#define EPS 1e-9
//define harmonic(n) 0.57721566490153286l+log(n)
#define nl puts("")
#define odd(n) ((n)&1)
#define even(n) (!((n)&1))
#define vsort(v) sort(v.begin(), v.end())
#define lc (node<<1)
#define rc ((node<<1)|1)
//loop
#define rep(i, n) for(int i = 0; i < n; ++i)
#define REP(i, n) for(int i = 1; i <= n; ++i)
//input
#define si(a) scanf("%d", &a)
#define sii(a, b) scanf("%d%d", &a, &b)
#define siii(a, b, c) scanf("%d%d%d", &a, &b, &c)
#define sl(a) scanf("%lld", &a)
#define sll(a, b) scanf("%lld%lld", &a, &b)
#define slll(a, b, c) scanf("%lld%lld%lld", &a, &b, &c)
#define sd(a) scanf("%lf", &a)
#define sc(a) scanf("%c", &a)
#define sst(a) scanf("%s", a)
inline bool EQ(double a, double b) { return fabs(a-b) < 1e-9; }
//debug
#ifdef tahsin
template < typename F, typename S >
ostream& operator << ( ostream& os, const pair< F, S > & p ) {
     return os << "(" << p.first << ", " << p.second << ")";
}

template < typename T >
ostream &operator << ( ostream & os, const vector< T > &v ) {
     os << "{";
     for(auto it = v.begin(); it != v.end(); ++it) {
          if( it != v.begin() ) os << ", ";
          os << *it;
     }
    return os << "}";
}

template < typename T >
ostream &operator << ( ostream & os, const set< T > &v ) {
     os << "[";
     for(auto it = v.begin(); it != v.end(); ++it) {
          if( it != v.begin()) os << ", ";
          os << *it;
     }
    return os << "]";
}

template < typename F, typename S >
ostream &operator << ( ostream & os, const map< F, S > &v ) {
     os << "[";
     for(auto it = v.begin(); it != v.end(); ++it) {
          if( it != v.begin() ) os << ", ";
          os << it -> first << " = " << it -> second ;
     }
    return os << "]";
}

#define dbg(args...) do {cerr << #args << " : "; faltu(args); } while(0)

clock_t tStart = clock();
#define timeStamp dbg("Execution Time: ", (double)(clock() - tStart)/CLOCKS_PER_SEC)

void faltu () { cerr << endl; }

template <typename T>
void faltu( T a[], int n ) {
     for(int i = 0; i < n; ++i) cerr << a[i] << ' ';
     cerr << endl;
}

template <typename T, typename ... hello>
void faltu( T arg, const hello &... rest) { cerr << arg << ' '; faltu(rest...); }

#else
#define dbg(args...)
#endif
```

```
ll bigmod(ll a, ll b) {
     ll ret = 1;
     while(b) {
          if(b&1) ret = (ret*a)%mod;
          b >>= 1; a = (a*a)%mod;
     }
     return ret;
}

ll inverse(ll n) { return bigmod(n, mod-2); }

ll add(ll a, ll b) {
     ll ret = a+b;
     if(ret >= mod) ret %= mod;
     return ret;
}

ll subtract(ll a, ll b) {
     ll ret = a-b;
     if(ret < 0) ret += mod;
     return ret;
}

ll mult(ll a, ll b) {
     ll ret = a*b;
     if(ret >= mod) ret %= mod;
     return ret;
}
//Direction Array
//int fx[]={1, -1, 0, 0}; int fy[]={0, 0, 1, -1};
//int fx[]={0, 0, 1, -1, -1, 1, -1, 1}; int fy[]={-1, 1, 0, 0, 1, 1, -1,
-1};

//bit manipulation
bool checkBit(int n, int i) { return (n&(1<<i)); }
int setBit(int n, int i) { return (n|(1<<i)); }
int resetBit(int n, int i) { return (n&(~(1<<i))); }
//end of template

//#define MX

int main () {
#ifdef tahsin
     freopen("in", "r", stdin);
//   freopen("out", "w", stdout);
#endif


//   timeStamp;
     return 0;
}
```

VIM SETTINGS

```
set number
syntax on
set smartindent
set shiftwidth=4
set tabstop=4
colorscheme jellybeans
set ruler
au BufNewFile * silent! 0r ~/.vim/skeleton/template.%:e
filetype indent on
```

Suffix Array

```
char a[MX];
int p[32][MX], step, n;
pair <pii, int> L[MX];

void build_suffix_array() {
	rep(i, n) p[0][i] = a[i]-'a';

	step = 1;
	for(int cnt = 1; cnt>>1 < n; ++step, cnt <<= 1) {
		rep(i, n) {
			L[i].F.F = p[step-1][i];
			L[i].F.S = i+cnt < n ? p[step-1][i+cnt]:-1;
			L[i].S = i;
		}

		sort(L, L+n);

		rep(i, n)
			p[step][L[i].S] = (i && L[i].F == L[i-1].F)?p[step][L[i-
1].S]:i;
	}
	--step;
}

int lcp(int x, int y) {
	int ret = 0;
	for(int k = step; k >= 0 && x < n && y < n; --k) {
		if(p[k][x] == p[k][y]) {
			ret += 1<<k;
			x += 1<<k;
			y += 1<<k;
		}
	}
	return ret;
}
```

Suffix Array nlogn

```
int n, step, cnt[MX], p[32][MX], L[MX], pre[MX], idx[MX];
char a[MX];

void build_suffix_array() {
	a[n++] = 'z'+1;

	rep(i, 129) cnt[i] = 0;
	rep(i, n) ++cnt[(int)a[i]];
	REP(i, 129) cnt[i] += cnt[i-1];
	rep(i, n) L[--cnt[(int)a[i]]] = i;

	p[0][L[0]] = 0;
	int classes = 0;
	for(int i = 1; i < n; ++i) {
		if(a[L[i]] != a[L[i-1]]) ++classes;
		p[0][L[i]] = classes;
	}

	step = 1;
	for(int h = 0; 1<<h < n; ++step, ++h) {
		rep(i, n) {
			pre[i] = L[i] - (1<<h);
			if(pre[i] < 0) pre[i] += n;
		}
		for(int i = 0; i <= classes; ++i) cnt[i] = 0;

		rep(i, n) ++cnt[p[step-1][pre[i]]];
		REP(i, classes) cnt[i] += cnt[i-1];

		for(int i = n-1; i >= 0; --i) L[--cnt[p[step-1][pre[i]]]] = pre[i];

		p[step][L[0]] = 0;
		classes = 0;
		for(int i = 1; i < n; ++i) {
			int mid1 = (L[i] + (1<<h))%n;
			int mid2 = (L[i-1] + (1<<h))%n;
			if(p[step-1][L[i]] != p[step-1][L[i-1]] || p[step-1][mid1] != p[step-1]
[mid2]) ++classes;

			p[step][L[i]] = classes;
		}
	}
	--n;
	--step;
}

int lcp(int x, int y) {
	int ret = 0;
	for(int k = step; k >= 0 && x < n && y < n; --k) {
		if(p[k][x] == p[k][y]) {
			ret += 1<<k;
			x += 1<<k;
			y += 1<<k;
		}
	}
	return ret;
}
```

Z-Algo

```
char a[MX];
int z[MX];

void z_algo() {
     int n = strlen(a);

     z[0] = 0;

     int l = 0, r = 0;
     for(int i = 1; a[i]; ++i) {
          if(i <= r) z[i] = min(z[i-l], r-i+1);
          while(i + z[i] < n && a[i+z[i]] == a[z[i]]) ++z[i];
          if(i+z[i]-1 > r) l = i, r = i+z[i]-1;
     }
}
```

Hashing

```
ll e1[MX], e2[MX];

void init() {
     e1[0] = e2[0] = 1;
     for(int i = 1; i < MX; ++i)
          e1[i] = (e1[i-1]*b1)%mod1, e2[i] = (e2[i-1]*b2)%mod2;
}

struct Hash {
     pll h[MX];
     int len;

     Hash () {
          len = 0;
          h[0] = {0, 0};
     }

     void insert(char ch) {
          ++len;
          h[len].F = (h[len-1].F*b1 + ch)%mod1;
          h[len].S = (h[len-1].S*b2 + ch)%mod2;
     }

     pll substr(int l, int r) {
          int L = r-l+1;

          pll ret;
          ret.F = ((h[r].F - h[l-1].F*e1[L])%mod1 + mod1)%mod1;
          ret.S = ((h[r].S - h[l-1].S*e2[L])%mod2 + mod2)%mod2;

          return ret;
     }
};
```

Aho-Corasick

```
int n, g[MX][26], f[MX], cnt[MX];
char text[1000010], pattern[505][505];
vi output[MX];

void build() {
     memset(g, -1, sizeof(g));
     memset(f, -1, sizeof(f));
     rep(i, MX) output[i].clear();
     rep(i, n) cnt[i] = 0;

     int root = 1;

     rep(i, n) {
          int curr = 0;
          for(int j = 0, k = strlen(pattern[i]); j < k; ++j) {
               int id = pattern[i][j]-'a';
               if(g[curr][id] == -1) g[curr][id] = root++;
               curr = g[curr][id];
          }
          output[curr].PB(i);
     }

     rep(i, 26) if(g[0][i] == -1) g[0][i] = 0;

     queue <int> q;
     rep(i, 26) if(g[0][i]) {
          q.push(g[0][i]);
          f[g[0][i]] = 0;
     }

     while(!q.empty()) {
          int curr = q.front();
          q.pop();

          rep(c, 26) if(g[curr][c] != -1) {
               int failure = f[curr];
               while(g[failure][c] == -1) failure = f[failure];
               failure = g[failure][c];

               f[g[curr][c]] = failure;

               rep(i, (int) output[failure].size())
                    output[g[curr][c]].PB(output[failure][i]);

               q.push(g[curr][c]);
          }
     }
}
```

```
int main () {
	si(n); sst(text);
	rep(i, n) sst(pattern[i]);

	build();
	int curr = 0;
	for(int i = 0; text[i]; ++i) {
		int id = text[i]-'a';
		while(g[curr][id] == -1) curr = f[curr];
		curr = g[curr][id];
		rep(j, (int) output[curr].size()) ++cnt[output[curr][j]];
	}
	rep(i, n) printf("%d\n", cnt[i]);

}
```

KMP

```
void buildPartialMatchTable(char* pattern) {
	table[0] = table[1] = 0;

	for(int i = 2; i <= n; ++i) {
		j = table[i - 1];
		while(true) {
			if(pattern[j] == pattern[i - 1]) {
				table[i] = j + 1;
				break;
			}
			if(j == 0) {
				table[i] = 0;
				break;
			}
			j = table[j];
		}
	}
}
int kmp(char* text, char* pattern) {
	int i = 0, j = 0, n = strlen(text), m = strlen(pattern);

	while(i < n) {
		if(text[i] == pattern[j]) {
			++i;
			++j;
			if(i == m) { // match found
				cnt ++;
				j = table[j];
			}
		}
		else if(j > 0) j = table[j];
		else ++i;
	}
	return cnt;
}
```

Manacher

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

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

	}
}
```

Matrix Exponantiation

```
struct mat {
	ll a[3][3];

	mat() { mem(a, 0); }

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

mat power(mat a, ll b) {
	mat ret;
	rep(i, 3) rep(j, 3) ret.a[i][i] = 1;

	while(b) {
		if(b&1) ret = ret*a;
		b >>= 1 ;
		a = a*a;
	}
	return ret;
}
```

Trie

```
struct node {
     int endmark;
     node *next[26];

     node() {
          endmark = 0;
          prefix = 0;
          for(int i = 0; i < 26; ++i) next[i] = NULL;
     }
} *root;

void insert() {
     node *curr = root;
     for(int i = 0, l = strlen(a); i < l; ++i) {
          int id = a[i]-'0';
          if(curr->next[id] == NULL) curr->next[id] = new node;
          curr = curr->next[id];
     }
     curr->endmark = 1;
}

void del(node *curr) {
     for(int i = 0; i < 10; ++i) if(curr->next[i]) del(curr->next[i]);
     delete curr;
}
```

Disjoint Set

```
#include<stdio.h>

int find_set(int x) { return p[x] = (p[x] == x ? x : find_set( p[x] )); }

void merge_sets(int x, int y) {
     int px = find_set(x);
     int py = find_set(y);

     if(px == py)return ;

     if(rnk[px] > rnk[py]) {
          p[py] = px;
          tot[px] += tot[py];
     }
     else {
          p[px] = py;
          tot[py] += tot[px];
     }
     if(rnk[px] == rnk[py]) rnk[py]++;
}

void init() {
     for(i=1; i<=n; i++) {
          p[i]=i;//root of the set
          tot[i]=1;//size of the set
          rnk[i]=0;//upper bound of height of the three
     }
}
```

HLD

```
int n, lev, lv[MX], par[MX][LOGMX], edgeToNode[MX];
pair<int, int> sp[MX];

struct data {
     int nd, cs, in;
     data(){}
     data(int a, int b, int c) {
          nd=a;
          cs=b;
          in=c;
     }
};

vector<data> edge[MX];

int dfs(int i, int pre) {
     lv[i] = ++lev;
     int sz = 1, k, subsz, mx = -1;

     for(k = 0; k < edge[i].size(); k ++) {
          if(edge[i][k].nd == pre)continue;

          subsz = dfs(edge[i][k].nd, i);

          par[edge[i][k].nd][0] = i;

          edgeToNode[edge[i][k].in] = edge[i][k].nd;

          if(subsz > mx) {
               mx = subsz;
               sp[i] = make_pair(edge[i][k].nd, edge[i][k].cs);
          }
          sz += subsz;
     }

     lev --;
     return sz;
}

int chainNo, chainHead[MX], chainInd[MX], posInAr[MX], Ar[MX], ind;

void hld(int i, int pre, int cs) {
     chainInd[i] = chainNo;
     posInAr[i] = ind;
```

```
     Ar[ind++] = cs;
     if(chainHead[chainNo] == -1) chainHead[chainNo] = i;

     if(sp[i].FR != -1) hld(sp[i].FR, i, sp[i].SC);

     for(int k = 0; k < edge[i].size(); k ++) {
          if(edge[i][k].nd == pre)continue;
          if(edge[i][k].nd != sp[i].FR) {
               chainNo ++;
               hld(edge[i][k].nd, i, edge[i][k].cs);
          }
     }
}

void build_lca() {
     int i, j;
     for(j = 1; (1 << j) < n; j ++)
          for(i = 2; i <= n; i ++)
               if(par[i][j-1] != -1)
                    par[i][j] = par[par[i][j-1]][j-1];
}

int query_lca(int u, int v) {
     int i, log;

     if(lv[u]<lv[v]) u^=v^=u^=v;

     for(log = 1; (1 << log) <= lv[u]; log ++); log--;

     for(i = log; i >= 0; i --) if(lv[u] - (1 << i) >= lv[v]) u = par[u]
[i];

     if(u == v)return u;

     for(i = log; i >= 0; i --) if(par[u][i] != -1 && (par[u][i] != par[v]
[i])) u = par[u][i], v = par[v][i];

     return par[u][0];
}

int t[4*MX];

void build_seg(int nd, int tl, int tr) {
     if(tl==tr) {
          t[nd]=Ar[tl];
          return ;
     }

     int tm=(tl+tr)>>1;
     int lc=nd<<1;
     int rc=lc|1;

     build_seg(lc, tl, tm);
     build_seg(rc, tm+1, tr);
     t[nd]=max(t[lc], t[rc]);
}

void update_seg(int nd, int tl, int tr, int pos, int num) {
     if(tl==tr) {
          t[nd]=num;
          return ;
     }

     int tm=(tl+tr)>>1;
     int lc=nd<<1;
     int rc=lc|1;

     if(pos<=tm)update_seg(lc, tl, tm, pos, num);
     else update_seg(rc, tm+1, tr, pos, num);

     t[nd]=max(t[lc], t[rc]);
}

int query_seg(int nd, int tl, int tr, int l, int r) {
     if(l > r)return -1;

     if(tl >= l && tr <= r) return t[nd];

     int tm = (tl + tr) >> 1;
     int lc = nd << 1;
     int rc = lc | 1;

     if(r <= tm) return query_seg(lc, tl, tm, l, r);
     else if(l > tm) return query_seg(rc, tm+1, tr, l, r);
     else return max(query_seg(lc, tl, tm, l, r), query_seg(rc, tm+1, tr, l, r));
}

int query_up(int u, int v) {
     if(u == v)return 0;
     int ans = -1;
     while(chainInd[u] != chainInd[v]) {
          ans = max(ans, query_seg(1, 0, n-1, posInAr[chainHead[chainInd[u]]],
posInAr[u]));
          u = par[chainHead[chainInd[u]]][0];
     }
     ans = max(ans, query_seg(1, 0, n-1, posInAr[v]+1, posInAr[u]));
     return ans;
}

int query(int u, int v) {
     if(u==v)return 0;
     int lca=query_lca(u, v);
     return max(query_up(u, lca), query_up(v, lca));
}

int main() {
     int tc, i, j, u, v, w;
```

```
char s[100];
scanf("%d", &tc);
while(tc--) {
	scanf("%d", &n);

	for(i=0; i<=n; i++) {
		chainHead[i]=-1;
		sp[i]=make_pair(-1, -1);
		edge[i].clear();
		for(j=0; (1<<j)<n; j++) par[i][j]=-1;
	}

	for(i=1; i<n; i++) {
		scanf("%d %d %d", &u, &v, &w);
		edge[u].PB(data(v, w, i));
		edge[v].PB(data(u, w, i));
	}

	lev=-1;
	dfs(1, -1);

	ind=chainNo=0;
	hld(1, -1, -1);

	build_seg(1, 0, n-1);
	build_lca();

		while(scanf("%s", s)) {
			if(!strcmp(s, "DONE"))break;
			if(!strcmp(s, "QUERY")) {
				scanf("%d %d", &u, &v);
				printf("%d\n", query(u, v));
			}
			else if(!strcmp(s, "CHANGE")) {
				scanf("%d %d", &u, &v);
				update_seg(1, 0, n-1, posInAr[edgeToNode[u]], v);
			}
		}
	}
	return 0;
}
```

BIT

```
int n, a[MX], tree[MX]; //tree is 1-indexed

void update(int idx, int val) {//add val to index idx
	while(idx <= n) tree[idx] += val;
		idx += (idx & -idx);
	}
}
```

```
int query(int idx) {//returns sum from index 1 to index idx
	int sum = 0;
	while(idx) {
		sum += tree[idx];
		idx -= (idx & -idx);
	}
	return sum;
}

int query(int idi, int idj) {//return sum from index idi to index idj
	int sum = 0;
	while(idi <= idj) {
		sum += tree[idj];
		idj -= (idj & -idj);
	}

	idi--;
	while(idi != idj) {
		sum -= tree[idi];
		idi -= (idi & -idi);
	}
	return sum;
}

int bS(int sum) {//normal binary search, return the index for which element 1-n sum
equals this parameter 'sum'
	int idx = 0;
	while(bitMask != 0) {
		int mid = idx + bitMask;
		if(mid <= n) {
			if(tree[mid] == sum) {
				return mid;
			}
			else if(tree[mid] < sum) {
				sum -= tree[mid];
				idx = mid;
			}
		}
		bitMask >>= 1;
	}
	if(sum != 0) return -1; //not found
	return idx;
}

int bS(int sum) {//returns the greatest index which equals sum
	int idx = 0;
	while(bitMask != 0) {

		int mid = idx + bitMask;

		if(mid <= n && (tree[mid] <= sum)) {
			sum -= tree[mid];
			idx = mid;
		}
```

```
          bitMask >>= 1;
     }
     if(sum != 0) return -1; //not found
     return idx;
}
```

2D-BIT

```
#define MX 1027

int n, tree[MX][MX];

void update(int idx, int idy, int val) {
     int y = idy;
     while(idx <= n) {
          idy = y;
          while(idy <= n) {
               tree[idx][idy] += val;
               idy += (idy & -idy);
          }
          idx += (idx & -idx);
     }
}

int query(int idxi, int idyi, int idxj, int idyj) {
     int sum = 0, yi = idyi, yj = idyj;

     while(idxi <= idxj) {
          idyi = yi, idyj = yj;
          while(idyi <= idyj) {
               sum += tree[idxj][idyj];
               idyj -= (idyj & -idyj);
          }

          idyi--;
          while(idyi != idyj) {
               sum -= tree[idxj][idyi];
               idyi -= (idyi & -idyi);
          }

          idxj -= (idxj & -idxj);
     }

     idxi--;
     while(idxi != idxj) {
          idyi = yi, idyj = yj;
          while(idyi <= idyj) {
               sum -= tree[idxi][idyj];
               idyj -= (idyj & -idyj);
          }

          idyi--;
          while(idyi != idyj) {
               sum += tree[idxi][idyi];
               idyi -= (idyi & -idyi);
          }

          idxi -= (idxi & -idxi);
     }
     return sum;
}

int query(int idx, int idy) {
     int sum = 0, y = idy;
     while(idx) {
          idy = y;
          while(idy) {
               sum += tree[idx][idy];
               idy -= (idy & -idy);
          }
          idx -= (idx & -idx);
     }
     return sum;
}
```

MO-s Algorithm

```
int sqr;
struct data {
     int f, s, in;
     data(){}
     data(int f, int s, int in) { this->f = f;
          this->s = s;
          this->in = in;
     }
     bool operator<(const data& a)const {
          return f/sqr == a.f/sqr ? s < a.s : f/sqr < a.f/sqr;
     }
};

data dt[MX3];
int a[MX1], cnt[MX2], ans[MX3], ans_now, l, r;

void rem(int pos) {
     cnt[a[pos]] --;
     if(cnt[a[pos]] == 0) ans_now --;
}

void add(int pos) {
     cnt[a[pos]] ++;
     if(cnt[a[pos]] == 1) ans_now ++;
}

int main() {
     int n, q, i, f, s, cur_r, cur_l;
```

```
	scanf("%d", &n);
	sqr = (int) sqrt(double (n) + 1e-7);
	for(i = 0; i < n; i ++) scanf("%d", &a[i]);

	scanf("%d", &q);
	for(i = 0; i < q; i ++) {
		scanf("%d %d", &f, &s);
		dt[i] = data(f-1, s-1, i);
	}

	sort(dt, dt+q);

	cur_l = cur_r = 0; // cur_l denotes the start of the current range,
cur_r denotes the point after the end of the current range
	//cnt[a[cur_l]] = ans_now = 1;

	for(i = 0; i < q; i ++) {
		l = dt[i].f;
		r = dt[i].s;

		while(cur_l < l) {
			rem(cur_l);
			cur_l ++;
		}
		while(cur_l > l) {
			add(cur_l - 1);
			cur_l --;
		}
		while(cur_r <= r) {
			add(cur_r);
			cur_r ++;
		}
		while(cur_r > r + 1) {
			rem(cur_r - 1);
			cur_r --;
		}
		ans[dt[i].in] = ans_now;
	}

	for(i = 0; i < q; i ++) printf("%d\n", ans[i]);
	return 0;
}
```

Given sum find n

```
int main() {
    int n, m, i, sqr, a, b;
    scanf("%d %d", &n, &m);
    m%=n*(n+1)/2;
    sqr=(sqrt(m*8.0+1)-1)/2;
    sqr=sqr*(sqr+1)/2;
    printf("%d\n", m-sqr);
}
```

Bitwise-Sieve

```
#define MX 1000000000
int marked[MX/64+2];

#define mark(x) marked[x>>6] |= (1<<((x&63)>>1))
#define check(x) (marked[x>>6] & (1<<((x&63)>>1)))

bool isPrime(int x) { return (x>1) && ((x==2) || ((x&1) && (!(check(x))))); }

void seive(int n) {
    int i, j;
    for(i=3; i*i<=n; i+=2)
        if(!check(i))
            for(j=i*i; j<=n; j+=i<<1) mark(j);
}
```

Euler Totient

```
int EulerT(int n) {
    if(n==1)return 0;
    int cnt, toi=n, i;
    for(i=0; i<primes.size() && (primes[i]*primes[i])<=n; i++) {
        if(n%primes[i]==0) {
            while(n%primes[i]==0) n/=primes[i];
            toi-=toi/primes[i];
        }
    }
    if(n>1) toi-=toi/n;
    return toi;
}
```

Divisors

```
struct data {
    int base, pwr;
    data(){}
    data(int a, int b){base=a, pwr=b;}   {
        printf("%d %d\n", divisors[i].base, divisors[i].pwr);
    }
}
};

vector<data> divisors;

void divs(int n) {
    int cnt, tot=1, i;
    for(i=0; i<primes.size() && (primes[i]*primes[i])<=n; i++) {
        if(n%primes[i]==0) {
            cnt=1;
            while(n%primes[i]==0) {
```

```
                n/=primes[i];
                cnt++;
            }
            divisors.push_back(data(primes[i], cnt-1));
            tot*=cnt;
        }
    }
    if(n>1) {
        tot*=2;
        divisors.push_back(data(n, 1));
    }
    printf("Number of divisors %d\n", tot);
    for(i=0; i<divisors.size(); i++)
          printf("%d %d\n", divisors[i].base, divisors[i].pwr);
}
```

Lucas Theorem

```
long long nCr(long long n, long long r) { return
(((fact[n]*inverse(fact[r]))%MOD)*inverse(fact[n-r]))%MOD; }

long long lucas(long long n, long long r) {
    if(!n && !r)return 1;
    long long ni=n%MOD;
    long long ri=r%MOD;
    if(ri>ni)return 0;
    return (lucas(n/MOD, r/MOD)*nCr(ni, ri))%MOD;
}
```

Persistant Segment Tree

```
int en[MX], de[MX], par[MX][20], lv[MX], lev, n;
pair<int, int> w[MX];
vector<int> edge[MX];

struct node {
     int cnt;
     node *l, *r;
     node() : cnt(0), l(NULL), r(NULL) {}
     node(int cnt, node *l, node *r): cnt(cnt), l(l), r(r) {}
     node* update(int tl, int tr, int add) {
          if(tl == tr) return new node(this -> cnt + 1, NULL, NULL);

          int tm = (tl + tr) >> 1;

          if(add <= tm) return new node(this -> cnt + 1, this -> l ->
update(tl, tm, add), this -> r);
          return new node(this -> cnt + 1, this -> l, this -> r ->
update(tm + 1, tr, add));
     }
} *head[MX];

node *root = new node();
```

```
int query(node *a, node *b, node *c, node *d, int tl, int tr, int k)
{
     if(tl == tr) return tl;

     int tm = (tl + tr) >> 1;
     int cnt = a -> l -> cnt + b -> l -> cnt - c -> l -> cnt - d -> l -> cnt;

     if(cnt >= k) return query(a -> l, b -> l, c -> l, d -> l, tl, tm, k);
     return query(a -> r, b -> r, c -> r, d -> r, tm + 1, tr, k - cnt);
}

void dfs(int i, int pre) {
     par[i][0] = pre;
     lv[i] = ++ lev;

     head[i] = ((pre == -1) ? root : head[pre]) -> update(0, n - 1, en[i]);

     for(int k = 0; k < edge[i].size(); k ++) {
          if(edge[i][k] == pre) continue;
          dfs(edge[i][k], i);
     }
     -- lev;
}

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

int query_lca(int u, int v) {
     int logg, i;
     if(lv[u] < lv[v]) swap(u, v);

     for(logg = 1; (1 << logg) <= lv[u]; logg ++); logg --;

     for(i = logg; ~i; i --)
          if(lv[u] - (1 << i) >= lv[v])
               u = par[u][i];

     if(u == v) return v;

     for(i = logg; ~i; i --)
          if(par[u][i] != -1 && par[u][i] != par[v][i])
               u = par[u][i], v = par[v][i];

     return par[u][0];
}

int main() {
     int m, i, j, u, v, k;
     scanf("%d %d", &n, &m);
```

```
for(i = 0; i < n; i ++) {
     scanf("%d", &w[i].FR);
     w[i].SC = i + 1;
}

sort(w, w + n);
for(i = 0; i < n; i ++) en[w[i].SC] = i, de[i] = w[i].FR;

for(i = 1; ; i ++) {
     for(j = 0; (1 << j) < n; j ++) par[i][j] = -1;
     if(i == n) break;
     scanf("%d %d", &u, &v);
     edge[u].PB(v);
     edge[v].PB(u);
}

root -> l = root -> r = root;

lev = 0;
dfs(1, -1);
lca();

while(m --) {
     scanf("%d %d %d", &u, &v, &k);

     int lc = query_lca(u, v);
          printf("%d\n", de[query(head[u], head[v], head[lc], (par[lc][0]
== -1 ? root : head[par[lc][0]]), 0, n - 1, k)]);
     }
     return 0;
}

int en[MX], de[MX];
pair<int, int> w[MX];

struct node {
     node *l, *r;
     int cnt;
     node() : l(NULL), r(NULL), cnt(0) {}
     node(node * l, node *r, int cnt) : l(l), r(r), cnt(cnt) {}
     node* update(int tl, int tr, int add) {
          if(tl == tr) return new node(NULL, NULL, this -> cnt + 1);

          int tm = (tl + tr) >> 1;

          if(add <= tm) return new node(this -> l -> update(tl, tm, add),
this -> r, this -> cnt + 1);
          return new node(this -> l, this -> r -> update(tm + 1, tr, add),
this -> cnt + 1);
     }
} *head[MX];
```

```
int query(node *a, node *b, int tl, int tr, int k) {
     if(tl == tr) return tl;

     int tm = (tl + tr) >> 1;
     int cnt = a -> l -> cnt - b -> l -> cnt;

     if(cnt >= k) return query(a -> l, b -> l, tl, tm, k);
     return query(a -> r, b -> r, tm + 1, tr, k - cnt);
}

int main() {
     int n, m, i, j, k;
     scanf("%d %d", &n, &m);
     for(i = 0; i <  n; i ++) {
          scanf("%d", &w[i].FR);
          w[i].SC = i;
     }
     sort(w, w + n);
     for(i = 0; i < n; i ++) en[w[i].SC] = i, de[i] = w[i].FR;

     node *root = new node();
     root -> l = root -> r = root;

     for(i = 0; i < n; i ++) {
          head[i] = (i ? head[i-1] : root) -> update(0, n - 1, en[i]);
     }
     while(m --) {
          scanf("%d %d %d", &i, &j, &k);
          i --; j --;
          printf("%d\n", de[query(head[j], (i ? head[i-1] : root), 0, n - 1, k)]);
     }
        return 0;
}
```

Knight-Distance

```
long long dist(long long x1, long long y1, long long x2, long long y2) {
     long long dx = abs(x2-x1);
     long long dy = abs(y2-y1);
     long long lb=(dx+1)/2;
     lb = max(lb, (dy+1)/2);
     lb = max(lb, (dx+dy+2)/3);
     while ((lb&1) != ((dx+dy)&1)) lb++;
     if (dx==1 && dy==0) return 3;
     if (dy==1 && dx==0) return 3;
     if (dx==2 && dy==2) return 4;
     return lb;
}

int n;
long long dp[(1<<15)+2], d[20][20];
pair <long long, long long> a[20], b[20];

long long f(int idx, int mask) {
```

```
     if(idx == n) return 0LL;

     long long &ret = dp[mask];
     if(ret != -1) return ret;

     ret = 10000000000000000LL;
     for(int i = 0; i < n; ++i) if(checkBit(mask, i) == 0) ret = min(ret,
d[idx][i] + f(idx+1, setBit(mask, i)));
     return ret;
}

int main () {
     int cs = 0;

     while(scanf("%d", &n) && n) {
          if(n == 0) break;
          for(int i = 0; i < n; ++i) scanf("%lld %lld", &a[i].first,
&a[i].second);
          for(int i = 0; i < n; ++i) scanf("%lld %lld", &b[i].first,
&b[i].second);

          for(int i = 0; i < n; ++i) for(int j = 0; j < n; ++j) d[i][j] =
dist(a[i].first, a[i].second, b[j].first, b[j].second);

          for(int i = 0, j = 1<<n; i < j; ++i) dp[i] = -1;
          printf("%d. %lld\n", ++cs, f(0, 0));
     }

     return 0;
}
```

SCC

```
void dfs1(int u) {
     vis[u] = 1;

     rep(i, (int) e[u].size()) if(vis[e[u][i]] == 0) dfs1(e[u][i]);
     order.PB(u);
}

void dfs2(int u) {
     vis[u] = 1;
     comp[u] = cnt;

     rep(i, (int) r[u].size()) if(vis[r[u][i]] == 0) dfs2(r[u][i]);
}

void scc() {
     rep(i, m*2) vis[i] = 0;
     rep(i, 2*m) if(vis[i] == 0) dfs1(i);
     rep(i, m*2) vis[i] = 0;

     cnt = 0;
     reverse(order.begin(), order.end());

     rep(i, 2*m) if(vis[order[i]] == 0) {
          ++cnt;
          dfs2(order[i]);
     }
}
```

2-SAT

```
int main () {
#ifdef tahsin
     freopen("in", "r", stdin);
#endif
     int t, n, u, v, nu, nv;

     si(t);

     REP(cs, t) {
          sii(n, m);

          order.clear();
          rep(i, m*2) e[i].clear(), r[i].clear();

          rep(i, n) {
               sii(u, v);
               int p = (u>0);
               int q = (v>0);

               u = abs(u)-1;
               v = abs(v)-1;

               if(p) u = u*2, nu = u+1;
               else u = u*2+1, nu = u-1;

               if(q) v = v*2, nv = v+1;
               else v = v*2+1, nv = v-1;

               e[u].PB(nv);
               r[nv].PB(u);

               e[v].PB(nu);
               r[nu].PB(v);
          }

          scc();

          int flag = 0;
          for(int i = 0; i < m<<1; i += 2) flag |= comp[i] == comp[i+1];

          printf("Case %d: ", cs);
          if(flag) printf("No\n");
```

```
          else {
               vi ans;
               for(int i = 0, k = 1; i < m<<1; i += 2, ++k) if(comp[i] <
comp[i+1]) ans.PB(k);

               printf("Yes\n%d", (int) ans.size());
               rep(i, (int) ans.size()) printf(" %d", ans[i]);
               nl;
          }
     }

     return 0;
}
```

Bridge/Articulation Point

```
#define MX 100000
int timer, n, vis[MX], fup[MX], tin[MX];
vi e[MX];
vector <pii> bridge;

void dfs(int u, int p) {
     vis[u] = 1;
     fup[u] = tin[u] = timer++;

     rep(i, (int) e[u].size()) if(e[u][i] != p) {
          int v = e[u][i];
          if(vis[v]) fup[u] = min(fup[u], tin[v]);
          else {
               dfs(v, u);
               fup[u] = min(fup[u], fup[v]);
               if(fup[v] > tin[u]) bridge.PB(MP(min(u, v), max(u, v)));
          }
     }
}

void find_bridge() {
     timer = 0;
     rep(i, n) vis[i] = 0;
     rep(i, n) if(vis[i] == 0) dfs(i, -1);

     printf("%d critical links\n", (int) bridge.size());

     sort(bridge.begin(), bridge.end());
     rep(i, (int) bridge.size()) printf("%d - %d\n", bridge[i].F,
bridge[i].S);

     bridge.clear();
}

int main () {
#ifdef tahsin
     freopen("in", "r", stdin);
#endif
     int t, m, u, v;

     si(t);

     REP(cs, t) {
          si(n);
          rep(i, n) {
               scanf("%d (%d)", &u, &m);

               while(m--) {
                    si(v);
                    e[u].PB(v);
                    e[v].PB(u);
               }
          }

          printf("Case %d:\n", cs);
          find_bridge();

          rep(i, n) e[i].clear();
     }

     return 0;
}
```

Maximum Bipartite Matching

```
bool kuhn(int u) {
     if(vis[u]) return 0;
     vis[u] = 1;

     rep(i, (int) e[u].size()) {
          int v = e[u][i]-n;

          if(pairs_of_right[v] == -1 || kuhn(pairs_of_right[v])) {
               pairs_of_right[v] = u;
               pairs_of_left[u] = v;
               return 1;
          }
     }
     return 0;
}

int max_matching() {
     rep(i, n) pairs_of_left[i] = -1;
     rep(i, m) pairs_of_right[i] = -1;

     int path_found = -1;
     do {
          path_found = 0;
          rep(i, n) vis[i] = 0;
```

```
        rep(i, n) if(pairs_of_left[i] < 0 && vis[i] == 0) path_found |=
kuhn(i);
     } while(path_found);

     int ret = 0;
     rep(i, n) if(pairs_of_left[i] != -1) ++ret;
     return ret;
}
```

MST Kruscal

```
int root[MAX], rank[MAX];

struct data {
    int u, v, w;
    bool operator<(const data& p)const {
        return w>p.w;
    }
};

int find(int i) {
    if(root[i]==i)return i;
    return root[i]=find(root[i]);
}

void merge(int x, int y) {
    int rx=find(x);
    int ry=find(y);
    if(rx!=ry) {
        if(rank[rx]<rank[ry]) root[rx]=ry;
        else root[ry]=rx;
        if(rank[rx]==rank[ry])rank[rx]++;
    }
}

int main() {
    pq<data> q;
    data get;
    int n, e, i, count=0, mst=0;
    scanf("%d %d", &n, &e);//n=nodes e=edges
    for(i=1; i<=n; i++) {
        root[i]=i;
        rank[i]=0;
    }
    for(i=1; i<=e; i++) {
        scanf("%d %d %d", &get.u, &get.v, &get.w);
        q.push(get);
    }
    for(i=1; i<=q.size(); i++) {
        data now=q.top();
        q.pop();
        if(find(now.u)!=find(now.v)) {
            count++;
            mst+=now.w;
            merge(now.u, now.v);
            if(count==n-1)break;
        }
    }
    cout<<mst<<endl;
    return 0;
}
```

Bellman Ford

```
struct edge
{ int u, v, w; };

edge data[MAX];
int key[MAX];

int main() {
    int n, m, i, j, cost;
    scanf("%d %d", &n, &m);
    for(i=1; i<=m; i++) scanf("%d %d %d", &data[i].u, &data[i].v, &data[i].w);
    for(i=1; i<=n; i++) key[i]=INF;
    key[1]=0;
    for(i=1; i<n; i++)
    {
        for(j=1; j<=m; j++) {
            cost=key[data[j].u]+data[j].w;
            if(key[data[j].v]>cost) key[data[j].v]=cost;
        }
    }
    for(j=1; j<=m; j++) {
        cost=key[data[j].u]+data[j].w;
        if(key[data[j].v]>cost) break;
    }
    if(j>m)printf("no negative cycle\n");
    else printf("negative cycle\n");
    return 0;
}
```

Maximum Subarray Sum (kadane)

```
int main() {
    int n, cum, cursi, curei, mx, mxsi, mxei, num, mxnum, in;
    scanf("%d", &n);
    mx=mxnum=-INF;//mx=max sum, mxnum=max element
    cum=0;//cumulative sum=ekhon porjonto shob element er sum
    cursi=1;//current start index
    for(curei=1; curei<=n; curei++)//curei=current end index
    {
        scanf("%d", &num);
        if(num>mxnum) {
            mxnum=num;
```

```
            in=curei;//in=position of max element
        }
        cum+=num;
        if(cum<0) {
            cum=0;//cumulative sum jodi 0 er cheye kom hoy then cum=0
            cursi=curei+1;//and then current start index er pore chole jabe
        }
        if(cum>mx) {
            mx=cum;
            mxsi=cursi;//mxsi=max sum array er start index
            mxei=curei;//mxei=max sum array er end index
        }
    }
    if(!mx) {//jodi max sum still 0 hoy then? that means if shob elelment
negative number hoy
        mx=mxnum;
        mxsi=mxei=in;
    }
    printf("%d %d %d\n", mx, mxsi, mxei);
    return 0;
}
```

Dinitz

```
int source, sink, NN, n, m;
int Q[MXN], fin[MXN], pro[MXN], dist[MXN], nEdge;
int flow[MXE], cap[MXE], nex[MXE], to[MXE];

void init() {
     for(int i = 0; i <= NN; i ++) fin[i] = -1;
     nEdge = 0;
}

void add(int u, int v, int c) {
     to[ nEdge ] = v; cap[ nEdge ] = c; flow[ nEdge ] = 0; nex[ nEdge ] =
fin[ u ]; fin[ u ] = nEdge ++;
     to[ nEdge ] = u; cap[ nEdge ] = 0; flow[ nEdge ] = 0; nex[ nEdge ] =
fin[ v ]; fin[ v ] = nEdge ++;
}

bool bfs() {
     int st, en;

     for(int i = 0; i <= NN; i ++) dist[i] = -1;

     dist[source] = st = en = 0;

     Q[ en ++ ] = source;

     while(st < en) {

          int u = Q[ st ++ ];

          for(int i = fin[u]; i >= 0; i = nex[ i ]) {
               int v = to[ i ];
               if(flow[i] < cap[i] && dist[v] == -1) {

                    dist[v] = dist[u] + 1;
                    Q[ en ++ ] = v;
               }
          }
     }

     return dist[sink] != -1;
}

int dfs(int u, int fl) {
     if(u == sink) return fl;

     for(int &i = pro[ u ]; i >= 0; i = nex[ i ]) {

          int v = to[ i ];

          if(flow[i] < cap[i] && dist[v] == dist[u] + 1) {

               int df = dfs(v, min(cap[ i ] - flow[ i ], fl) );

               if(df > 0) {
                    flow[i] += df;
                    flow[i ^ 1] -= df;
                    return df;
               }
          }
     }
     return 0;
}

int dinic() {
     int ret = 0;

     for(int i = 0; i < nEdge; i ++) flow[i] = 0;

     while( bfs() ) {
          for(int i = 0; i <= NN; i ++) pro[i] = fin[i];

          while(true) {

               int df = dfs(source, INF);
               if(df) ret += df;
               else break;
          }
     }
     return ret;
}
```

Edmond-Karp

```
vi e[MX];
int source, sink, n, vis[MX], parent[MX], cap[MX][MX];

int find_path() {
     mem(vis, 0);
     mem(parent, -1);

     queue <int> q;
     q.push(source);
     vis[source] = 1;

     while(!q.empty()) {
          int where = q.front(); q.pop();

          if(where == sink) break;

          rep(next, tot) {
               if(vis[next] == 0 && cap[where][next] > 0) {
                    q.push(next);
                    vis[next] = 1;
                    parent[next] = where;
               }
          }
     }

     int where = sink, path_cap = inf;

     while(parent[where] > -1) {
          int prev = parent[where];
          path_cap = min(path_cap, cap[prev][where]);
          where = prev;
     }

     where = sink;
     while(parent[where] > -1) {
          int prev = parent[where];
          cap[prev][where] -= path_cap;
          cap[where][prev] += path_cap;
          where = prev;
     }

     if(path_cap == inf) return 0;
     return path_cap;
}

int max_flow() {
     int ret = 0;
     while(1) {
          int path_cap = find_path();
          if(path_cap == 0) break;
          ret += path_cap;
     }
     return ret;
}
```

Edit-Distance

```
int main() {
     dp[0][0] = 0;
     for(int i = 1; i <= 100 ++i) dp[i][0] = dp[0][i] = i;

     scanf("%s%s", a, b);

     int n = strlen(a);
     int m = strlen(b);

     for(int i = 1; i <= n; ++i)
          for(int j = 1; j <= m; ++j)
               dp[i][j] = min(min(dp[i-1][j], dp[i][j-1])+1, dp[i-1][j-1] + (a[i-1]!
=b[j-1]));

     printf("%d\n", dp[n][m]);
}
```

LIS nlogn

```
int n;
int a[10010], b[10010];

int bs(int h, int n) {
     int l = 1, ans = 0;
     while(l <= h) {
          int m = (l+h)>>1;
          if(n > a[b[m]]) { ans = m; l = m+1; }
          else h = m-1;
     }
     return ans+1;
}

int lis() {
     int len = 0;
     int p[n];

     for(int i = 0; i < n; ++i) {
          int tmp = bs(len, a[i]);
          b[tmp] = i;
          p[i] = b[tmp-1];
          len = max(tmp, len);
     }
     int k = b[len];
     int s[len];
     for(int i = len-1; i >= 0; --i) {
          s[i] = a[k];
          k = p[k];
     }
```

```
     for(int i = 0; i < len; ++i) cout << s[i] << ' ';
     cout << endl;
     return len;
}
```

Segment Tree (Lazy propagation)

```
void update_node(int nd, int tl, int tr, long long add) {
    lz[nd]+=add;
    t[nd]+=add*(tr-tl+1);
}
void push(int nd, int tl, int tr){
    int tm=(tl+tr)/2;
    update_node(nd*2, tl, tm, lz[nd]);
    update_node(nd*2+1, tm+1, tr, lz[nd]);
    lz[nd]=0;
}
void build(int nd, int tl, int tr){
    t[nd]=lz[nd]=0;
    if(tl==tr)return ;

    int tm=(tl+tr)/2;
    build(nd*2, tl, tm);
    build(nd*2+1, tm+1, tr);
}
void update(int nd, int tl, int tr, long long add){
    if(tr<l || tl>r) return;
    if(tl>=l && tr<=r) {
        update_node(nd, tl, tr, add);
        return ;
    }
    if(lz[nd]) push(nd, tl, tr);

    int tm=(tl+tr)/2;

    update(nd*2, tl, tm, add);
    update(nd*2+1, tm+1, tr, add);

    t[nd]=t[nd*2]+t[nd*2+1];
}

long long query(int nd, int tl, int tr) {
    if(tr<l || tl>r)return 0;

    if(tl>=l && tr<=r) return t[nd];

    if(lz[nd]) push(nd, tl, tr);

    int tm=(tl+tr)/2;

    return query(nd*2, tl, tm)+query(nd*2+1, tm+1, tr);
}
```

GEO

```
struct point_i {
	int x, y;

	point_i () { x = y = 0.0; }
	point_i (int _x, int _y) { x = _x, y = _y; }

	int normSq() {
		return sqr(x) + sqr(y);
	}
};

struct point {
	double x, y;

	point () { x = y = 0.0; }
	point (double _x, double _y) { x = _x, y = _y; }

	double normSq() { //same as dot product A.A
		return x*x + y*y;
	}

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

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

struct vec {
	double x, y;
	vec () { x = y = 0.0; }
	vec (double _x, double _y) { x = _x, y = _y; }
	vec (point a, point b) { x = b.x-a.x, y = b.y-a.y; }

	vec operator + (const point &rhs) {
		vec tmp;
		tmp.x = x+rhs.x;
		tmp.y = y+rhs.y;

		return tmp;
	}

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

		return tmp;
	}

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

		return tmp;
	}

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

		return tmp;
	}

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

//distance between two points
double dist (point a, point b) { return hypot(a.x - b.x, a.y - b.y);}


//rotate the point CCW
point rotate (point p, double theta) {
	double rad = theta*PI/180;    //degree to radian
	return point(p.x * cos(rad) - p.y * sin(rad), p.x * sin(rad) + p.y * cos(rad));
}

point rotate (point p, point c, double rad) {
	p.x -= c.x;
	p.y -= c.y;
	return point(p.x * cos(rad) - p.y * sin(rad) + c.x, p.x * sin(rad) + p.y *
cos(rad) + c.y);
}

struct line {
	double a, b, c;

	line () { a = b = c = 0.0; }

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

```
          c = -(double) (a * p1.x) - p1.y;
     }
};

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

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

bool lineIntersect (line l1, line l2, point &p) { //not segments
     if(areParallel(l1, l2)) return 0;

     p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
     if(fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
     else p.y = -(l2.a * p.x + l2.c);

     return 1;
}

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

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

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

double distToLine (point p, point a, point b, point &c) {
     //formula c = a + u*ab;

     vec ap(a, p), ab(a, b);
     double u = (ap*ab) / (ab*ab);
     c = translate(a, scale(ab, u));
     return dist(p, c);
}

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

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

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

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

bool segmentIntersect(point a, point b, point c, point d) //return true if two segments
intersect
{
     //two lines are AB and CD
     int d1 = direction(c, d, a); //direction of A, with respect to line CD;
     int d2 = direction(c, d, b);
     int d3 = direction(a, b, c);
     int d4 = direction(a, b, d);

     if(d1*d2 < 0 && d3*d4 < 0) return 1; //if they intersect;
     if(d1 == 0 && onSegment(c, d, a)) return 1;
     if(d2 == 0 && onSegment(c, d, b)) return 1;
     if(d3 == 0 && onSegment(a, b, c)) return 1;
     if(d4 == 0 && onSegment(a, b, d)) return 1;

     return 0;
}


double area2Dpolygon(int n, point a[]) {
     double area = 0;

     for(int i = 0; i+1 < n; ++i) {
          area += a[i].x*a[i+1].y;
          area -= a[i].y*a[i+1].x;
     }
     area += a[2].x*a[0].y;
     area -= a[2].y*a[0].x;

     return fabs(area)/2.0;
}

//circle
double perimeterTriangle(double a, double b, double c) { //perimeter of a triangle
```

```
	return a+b+c;
}

double areaTriangle(double a, double b, double c) { //area of a triangle
	double s = 0.5*perimeterTriangle(a, b, c);
	return sqrt ( s * (s-a) * (s-b) * (s-c) );
}

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

double rCircumCircle(double ab, double bc, double ca) {
	return ab * bc * ca / (4.0 * areaTriangle(ab, bc, ca));
}

double rCircumCircle(point a, point b, point c) {
	return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));
}

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