```
#include <iostream>

int main() {
    while(++Procrastinators);
}
```

Compilaton error: Missing ‘slifer’

Be Done

Table of Contents

Data Structures

Ordered Set

```
//O(log ^ 2 N) updates and queries

#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/rope>
using namespace std;
using namespace __gnu_pbds;
using namespace __gnu_cxx;
template<class T> using Tree = tree<T, null_type, less<T>,
rb_tree_tag,tree_order_statistics_node_update>;
Tree<int> t[N];
void add(int idx, int v){
    for(int x = ++idx; x < N; x += x & -x){
        t[x].insert(v);
    }
}
void erase(int idx, int v){
    for(int x = ++idx; x < N; x += x & -x)
        t[x].erase(v);
}
int get(int idx,  int limit){
    int ret = 0;
    for(int x = ++idx; x; x -= x & -x)
        ret += (t[x].order_of_key(limit+1));
    return ret;
}
```

BIT Update Range

```
#include <bits/stdc++.h>
using namespace std;
const int N = (1 << 17);

typedef long long ll;
```

```
struct BIT {
	ll M[N], C[N];

	void init(int n) {
		n = (1 << (int) ceil(log2(n)));
		memset(M, 0, n * sizeof(M[0]));
		memset(C, 0, n * sizeof(C[0]));
	}

	void add(int idx, ll valM, ll valC) {
		++idx;
		while (idx <= N) {
			M[idx - 1] += valM;
			C[idx - 1] += valC;
			idx += (idx & (-idx));
		}
	}
	ll get(int idx) {
		ll valM = 0, valC = 0;
		int x = idx;
		++idx;
		while (idx) {
			valM += M[idx - 1];
			valC += C[idx - 1];
			idx -= (idx & (-idx));
		}
		return valM * x + valC;
	}

	void addrange(int s, int e, ll val) {
		add(s, val, (1 - s) * val);
		add(e + 1, -val, e * val);
	}

} bit;

int main() {

	int tc;
	scanf("%d", &tc);
	while (tc--) {
		int n, c;
		scanf("%d%d", &n, &c);
		bit.init(n);
		int t, x, y;
		ll v;
		while (c--) {
			scanf("%d", &t);
			if (t) {
				scanf("%d%d", &x, &y);
				printf("%lld\n", bit.get(y) - bit.get(x - 1));
			} else {
				scanf("%d%d%lld", &x, &y, &v);
				bit.addrange(x, y, v);
```

```
                    }
                }

            }
}
```

BIT Multi Set

```
struct BIT{
    int N = 1 << 16;
    int tree[1 << 16];
    BIT(){
        mm(tree,0);
        init();
    }
    int getSum(int i){
        if (i == -1)
            return -1;
        int res = 0;
        for (i++; i; i -= i & (-i))
            res += tree[i - 1];
        return res;
    }
    void add(int i, int v){
        for (i++; i <= N; i += i &(-i))
            tree[i-1] += v;
    }
    int find(int t){ // lower bound
        int s = 0;
        for (int sz = N >> 1; sz; sz >>=1)
            if (tree[(s + sz) - 1] < t)
                t -= tree[(s += sz) - 1];
        return s;
    }
    void init(){
        add(0, -1);
    }
    void insert(int x){
        add(x, 1);
    }
    int count(int x){
        return getSum(x) - getSum(x - 1);
    }
    int at(int i){
        return find(i);
    }
    void eraseAll(int x){
        add(x, -count(x));
    }
    void eraseNum(int x,int cnt){
        add(x, -min(cnt,count(x)));
    }
    void eraseOne(int x){
        if (count(x))
            add(x, -1);
```

```
    }
    int lower_bound(int v){
        return getSum(v - 1) + 1;
    }
    int upper_bound(int v){
        return lower_bound(v + 1);
    }
    int size(){
        return getSum(N - 1) + 1;
    }
};
```

BIT (Small)

```
const int N = 1e5 + 5;
vector<int> v;
long long tree[N];
void increase(int i, int delta){
    for (; i < N; i |= i + 1)
        tree[i] += delta;
}
int sum(int i){
      int ans = 0;
      for (; i >= 0; i = (i & (i + 1)) - 1)
            ans += tree[i];
      return ans;
}
int getsum(int left, int right){
    return sum(right) - sum(left-1);
}
void constuct(){
    for(int i = 0 ; i<N ; i++)
        increase(i, v[i]);
}
```

BIT 2D

```
struct BIT_2D{
      int N = 1 << 10;
      int tree[1 << 10][1 << 10];
      BIT_2D(){
            mm(tree,0);
      }
      int getSum(int i, int j) {
            ++i, ++j;
            int res = 0;
            for (; i; i -= i & (-i))
                  for (int jj = j; jj; jj -= jj & (-jj))
                        res += tree[i - 1][jj - 1];
            return res;
      }
      void add(int i, int j, int v) {
            ++i, ++j;
            for (; i <= N; i += i & (-i))
```

```
                    for (int jj = j; jj <= N; jj += jj & (-jj))
                          tree[i - 1][jj - 1] += v;
      }
      int getSum(int imin, int jmin, int imax, int jmax) {
            return getSum(imax, jmax) - getSum(imin - 1, jmax) -
getSum(imax, jmin - 1) + getSum(imin - 1, jmin - 1);
      }
};
```

Segment Tree With Beats Mod

```
ll sum[1<<18],mx[1<<18];
int n,q;
ll a[N];
void build(int ni=0,int ns=0,int ne=n-1){
    if(ns==ne){
        sum[ni]=mx[ni]=a[ns];
        re ;
    }
    int mid=(ns+(ne-ns)/2);
    build(ni*2+1,ns,mid);
    build(ni*2+2,mid+1,ne);
    sum[ni]=sum[ni*2+1]+sum[ni*2+2];
    mx[ni]=max(mx[ni*2+1],mx[ni*2+2]);
}
void update_mod(int qs,int qe,int m,int ni=0,int ns=0,int ne=n-1){
    if(qs>ne||qe<ns||mx[ni]<m)re ;
    if(ns==ne){
        sum[ni]=mx[ni]=sum[ni]%m;
        re ;
    }
    int mid=(ns+(ne-ns)/2);
    update_mod(qs,qe,m,ni*2+1,ns,mid);
    update_mod(qs,qe,m,ni*2+2,mid+1,ne);
    sum[ni]=sum[ni*2+1]+sum[ni*2+2];
    mx[ni]=max(mx[ni*2+1],mx[ni*2+2]);
}
void update_val(int qi,int v,int ni=0,int ns=0,int ne=n-1){
    if(qi>ne||qi<ns)re ;
    if(ns==ne){
        sum[ni]=mx[ni]=v;
        re ;
    }
    int mid=(ns+(ne-ns)/2);
    update_val(qi,v,ni*2+1,ns,mid);
    update_val(qi,v,ni*2+2,mid+1,ne);
    sum[ni]=sum[ni*2+1]+sum[ni*2+2];
    mx[ni]=max(mx[ni*2+1],mx[ni*2+2]);
}
ll query(int qs,int qe,int ni=0,int ns=0,int ne=n-1){
    if(qs>ne||qe<ns)re 0;
    if(qs<=ns&&ne<=qe)re sum[ni];
    int mid=(ns+(ne-ns)/2);
    re query(qs,qe,ni*2+1,ns,mid)+query(qs,qe,ni*2+2,mid+1,ne);
```

```
}
int x,y,t,m,v;
int main()
{
    scanf("%d%d",&n,&q);
    for(int i=0;i<n;++i)scanf("%lld",a+i);
    build();
    while(q--){
        scanf("%d",&t);
        if(t==1){
            scanf("%d%d",&x,&y);
            printf("%lld\n",query(--x,--y));
        }
        else if(t==2){
            scanf("%d%d%d",&x,&y,&m);
            update_mod(--x,--y,m);
        }
        else{
            scanf("%d%d",&x,&v);
            update_val(--x,v);
        }
    }
    re 0;
}
```

Merge Sort Tree

```
const int maxn=1e5+5;
const int N=1e3+5,M=25000+5;
int n,q,a[maxn];
pair<int,vector<int> > tree[1<<18];
void build(int ni=0,int ns=0,int ne=n-1){
      if(ns==ne){
            tree[ni].f=1;
            tree[ni].s.push_back(a[ns]);
            re ;
      }
      int mid=(ns+ne)/2;
      build(ni*2+1,ns,mid);
      build(ni*2+2,mid+1,ne);
      merge(all(tree[ni*2+1].s),all(tree[ni*2+2].s),inserter(tree[ni].s,tree[n
i].s.begin()));
      int cnt=0,Max=0,last=INT_MIN;
      for(auto i:tree[ni].s){
            if(i!=last){
                  Max=max(Max,cnt);
                  cnt=0;
            }
            last=i;
            ++cnt;
      }
      tree[ni].f=max(cnt,Max);
}
int query(int qs,int qe,int ni=0,int ns=0,int ne=n-1){
      if(qs>ne || qe<ns)re 0;
```

```
	if(ns>=qs&&ne<=qe)
		re tree[ni].f;
	int mid=(ns+ne)/2;
	re max(query(qs,qe,ni*2+1,ns,mid),query(qs,qe,ni*2+2,mid+1,ne));
}
}
void getPthParent(LCA){
const int maxn=1e5+5;
const int N=1e4+5,M=N*2;
int t,n,id;
string s;
int cost[N],cst[M];
int lev[N],in[N],out[N],sp[16][N];
int head[N],nxt[M],to[M],ne;
void init(){
	ne=0;
	memset(head,-1,n*sizeof head[0]);
}
void add_edge(int f,int t,int c){
	to[ne]=t;
	nxt[ne]=head[f];
	cst[ne]=c;
	head[f]=ne++;
}
void add_bi_edge(int a,int b,int c){
	add_edge(a,b,c);
	add_edge(b,a,c);
}
void dfs(int u,int par,int l,int c){
	in[u]=++id;
	lev[u]=l;
	cost[u]=c;
	sp[0][u]=par;
	for(int e=head[u];~e;e=nxt[e]){
		int v=to[e];
		if(v!=par)dfs(v,u,l+1,c+cst[e]);
	}
	out[u]=id;
}
void buildSparse(){
	for(int j=1;j<=15;++j)
		for(int i=0;i<n;++i)
			sp[j][i]=sp[j-1][sp[j-1][i]];
}
int getPth(int u,int p){
	for(int i=15;i>=0;--i)
		if(p&(1<<i))
			u=sp[i][u];
	re u;
}
int lca(int u,int v){
	if(lev[u]<lev[v])
		swap(u,v);
	u=getPth(u,lev[u]-lev[v]);
	if(u==v)re u;
```

```
	for(int i=15;i>=0;--i)
		if(sp[i][u]!=sp[i][v])
			u=sp[i][u],v=sp[i][v];
	re sp[0][u];
}
ll dis(int u,int v){
	int LCA=lca(u,v);
	re cost[u]+cost[v]-(2ll*cost[LCA]);
}
int getKth(int u,int v,int k){
	int LCA=lca(u,v);
	int Udis=lev[u]-lev[LCA];
	int Vdis=lev[v]-lev[LCA];
	int Alldis=Udis+Vdis;
	if(k<=Udis)re getPth(u,k);
	else re getPth(v,Alldis-k);
}
int main()
{
	IO;
	cin >> t;
	while(t--){
		cin >> n;
		init();
		int u,v,k,c;
		for(int i=1;i<n;++i){
			cin >> u >> v >> c;
			add_bi_edge(--u,--v,c);
		}
		id=-1;
		dfs(0,-1,0,0);
		buildSparse();
		while(true){
			cin >> s;
			if(s=="DONE")break;
			else if(s=="DIST"){
				cin >> u >> v;
				cout<<dis(--u,--v)<<"\n";
			}
			else{
				cin >> u >> v >> k;
				cout<<getKth(--u,--v,--k)+1<<"\n";
			}
		}
		cout<<"\n";
	}
	re 0;
}
```

DSU

```
const int N = 1e5 + 5;
int p[N] , w[N];
int n;
```

```
int f(int u){
    if(u == p[u])return u;
    return p[u] = f(p[u]);
}
bool is_connected(int a , int b){
    return f(a) == f(b);
}
void connect(int a , int b){
    a = f(a);
    b = f(b);
    if(a == b)return;
    if(w[a] < w[b])swap(a,b);
    w[a] += w[b];
    p[b] = a;
}
void init(){
    for(int i = 0 ; i < N ; i++)
        p[i] = i , w[i] = 1;
}
```

Gilbert Order

```
const int infinity = (int)1e9 + 42;
const int64_t llInfinity = (int64_t)1e18 + 256;
const int module = (int)1e9 + 7;
const long double eps = 1e-8;

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

struct Query {
	int l, r, idx;
	int64_t ord;

	inline void calcOrder() {
		ord = gilbertOrder(l, r, 21, 0);
	}
```

```
};

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

signed main() {
	#ifndef USE_FILE_IO
		ios_base::sync_with_stdio(false);
	#endif

	mt19937 rnd(42);

	int n, m, k; cin >> n >> m; k = rnd() % 1048576;
	vector<int> p(n+1);
	for (int i = 0; i < n; i++) {
		int val = rnd() % 1048576;
		p[i+1] = p[i] ^ val;
	}

	vector<Query> qry(m);
	for (int i = 0; i < m; i++) {
		int l = rnd() % n + 1, r = rnd() % n + 1;
		if (l > r) {
			swap(l, r);
		}
		qry[i].l = l; qry[i].r = r;
		qry[i].idx = i;
		qry[i].calcOrder();
	}

	int64_t ans = 0;
	vector<int64_t> res(m);
	vector<int64_t> cnt((int)2e6, 0);
	sort(qry.begin(), qry.end());
	int l = 0, r = 1;
	ans = (p[1] == k);
	cnt[p[0]]++; cnt[p[1]]++;

	for (Query q: qry) {
		q.l--;
		while (l > q.l) {
			l--;
			ans += cnt[p[l] ^ k];
			cnt[p[l]]++;
		}
		while (r < q.r) {
			r++;
			ans += cnt[p[r] ^ k];
			cnt[p[r]]++;
		}
		while (l < q.l) {
			cnt[p[l]]--;
			ans -= cnt[p[l] ^ k];
			l++;
```

```
            }
            while (r > q.r) {
                cnt[p[r]]--;
                ans -= cnt[p[r] ^ k];
                r--;
            }
            res[q.idx] = ans;
        }

        uint64_t rhsh = 0;
        for (int i = 0; i < m; i++) {
            rhsh *= (uint64_t)1e9 + 7;
            rhsh += (uint64_t)res[i];
        }
        cout << rhsh << "\n";

        return 0;
}
```

Median Tracker

```
struct median_tracker {
    priority_queue<long long> a;
    priority_queue<long long , vector<long long> , greater<long long> > b;

    long long get_mid() {
        return a.top();
    }

    void insert(long long x) {
        a.size() == b.size() ? b.push(x) : a.push(x);
        if (b.size() > a.size()) a.push(b.top()) , b.pop();
        if (a.size() - 1 > b.size()) b.push(a.top()) , a.pop();
    }
};
```

Monotonic Queue

```
struct monoqueue {
    deque<pair<int, int> > dq;
    int sz = 0;
    int mn() {
        return dq.size() ? dq.front().first : inf;
    }
    void push(int v) {
        sz++;
        int cnt = 1;
        while (dq.size() && v <= dq.back().first)
            cnt += dq.back().second , dq.pop_back();
        dq.push_back({v , cnt});
    }
    void pop() {
        sz--;
```

```
        if (dq.front().second == 1)
            return dq.pop_front();
        dq.front().second--;
    }
    int size() {
        return sz;
    }
};
```

Mo With Updates

```
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int N = 1e5 +5;
const int M = 2 * N;
const int blk = 2155;
const int mod = 1e9 + 7;
struct Query{
  int l, r, t, idx;
  Query(int a = 0,int b = 0,int c = 0,int d = 0){l=a,r=b,t=c,idx = d;}
  bool operator < (Query o){
    if(r / blk == o.r / blk && l / blk == o.l / blk)return t < o.t;
    if(r / blk == o.r / blk)return l < o.l;
    return r < o.r;
  }
} Q[N];

int a[N], b[N];
int cnt1[M], cnt2[N];
int L = 0, R = -1, K = -1;
void add(int x){
//  cout << x << '\n';
  cnt2[cnt1[x]]--;
  cnt1[x]++;
  cnt2[cnt1[x]]++;
}
void del(int x){
  cnt2[cnt1[x]]--;
  cnt1[x]--;
  cnt2[cnt1[x]]++;
}
map<int,int>id;
int cnt;
int ans[N];
int p[N], nxt[N];
int prv[N];
void upd(int idx){
  if(p[idx] >= L && p[idx] <= R)
```

```
    del(a[p[idx]]), add(nxt[idx]);
  a[p[idx]] = nxt[idx];
}
void err(int idx){
  if(p[idx] >= L && p[idx] <= R)
    del(a[p[idx]]), add(prv[idx]);
  a[p[idx]] = prv[idx];
}
int main(){

  int n, q, l, r, tp;

  scanf("%d%d", &n, &q);

  for(int i = 0; i < n; i++){
    scanf("%d", a + i);
    if(id.count(a[i]) == 0)
      id[a[i]] = cnt++;
    a[i] = id[a[i]];
    b[i] = a[i];
  }
  int qIdx = 0;
  int ord = 0;
  while(q--){

    scanf("%d", &tp);
    if(tp == 1){
      /// ADD Query
      scanf("%d%d", &l, &r);  --l, --r;
      Q[qIdx] = Query(l,r,ord-1,qIdx); qIdx++;
    } else{
      /// ADD Update
      scanf("%d%d",p + ord, nxt + ord); --p[ord];
      if(id.count(nxt[ord]) == 0)
        id[nxt[ord]] = cnt++;
      nxt[ord] = id[nxt[ord]];
      prv[ord] = b[p[ord]];
      b[p[ord]] = nxt[ord];
      ++ord;
    }

  }
  sort(Q,Q+qIdx);
  for(int i = 0; i < qIdx; i++){
    while(L < Q[i].l)del(a[L++]);
    while(L > Q[i].l)add(a[--L]);
    while(R < Q[i].r)add(a[++R]);
    while(R > Q[i].r)del(a[R--]);
    while(K < Q[i].t)upd(++K);
    while(K > Q[i].t)err(K--);
    ///Solve Query I
  }
  for(int i = 0; i < qIdx; i++)
    printf("%d\n", ans[i]);
```

```
  return 0;
}
```

Ordered Set

```
#include <iostream>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
using namespace std;

template<typename T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;


int main() {
       Tree<int> X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

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

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

       return 0;
}
```

Persistent Segment Tree (Lazy)

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

using namespace std;
#define ll long long
struct node
{
    node *l, *r;
    ll sum,lazy;
    node():sum(0),lazy(0),l(this),r(this){}
    node(ll sum,ll lazy,node *l,node *r):sum(sum),lazy(lazy),l(l),r(r){}
};
const int N=1e5+5;
node* root[N];
```

```
int n;
node* update(node *cur,int su,int eu,int d,int s=1,int e=n)
{
    if(s>eu||e<su) return cur;
    if(s>=su&&e<=eu)
        return new node(cur->sum+(e-s+1)*d,cur->lazy+d,cur->l,cur->r);
    int mid=s+(e-s)/2;
    node *l=update(cur->l,su,eu,d,s,mid);
    node *r=update(cur->r,su,eu,d,mid+1,e);
    return new node(l->sum+r->sum+(e-s+1)*cur->lazy,cur->lazy,l,r);
}
int intersect(int sq,int eq,int s,int e)
{
    int l=max(sq,s);
    int r=min(eq,e);
    if(l>r) return 0;
    return r-l+1;
}
ll query(node *cur,int sq,int eq,int s=1,int e=n)
{
    if(s>eq||e<sq) return 0;
    if(s>=sq&&e<=eq) return cur->sum;
    int mid=s+(e-s)/2;
    ll sumL=query(cur->l,sq,eq,s,mid)+intersect(sq,eq,s,mid)*cur->lazy;
    ll sumR=query(cur->r,sq,eq,mid+1,e)+intersect(sq,eq,mid+1,e)*cur->lazy;
    return sumL+sumR;

}
```

Persistent Segment Tree (Array)

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

using namespace std;

const int maxn = 1e5, maxk = 1e6 + 1;
int root[maxn], L[16 * maxn], R[16 * maxn], sum[16 * maxn];
int rt = 1, sz = 1;
int lpos[maxk];

int copy(int v, int &u)
{
    L[sz] = L[v];
    R[sz] = R[v];
    sum[sz] = sum[v];
    return u = sz++;
}

void make_root()
{
    copy(root[rt - 1], root[rt]);
    rt++;
}

void add(int pos, int x, int v = root[rt - 1], int l = 0, int r = maxn)
```

```
{
       sum[v] += x;
    if(r - l == 1)
        return;
    int m = (l + r) / 2;
    if(pos < m)
        add(pos, x, copy(L[v], L[v]), l, m);
    else
        add(pos, x, copy(R[v], R[v]), m, r);
}

int get(int a, int b, int v, int l = 0, int r = maxn)
{
    if(a <= l && r <= b)
        return sum[v];
    if(r <= a || b <= l)
        return 0;
    int m = (l + r) / 2;
    return get(a, b, L[v], l, m) + get(a, b, R[v], m, r);
}

int main()
 {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        int t;
        cin >> t;
        make_root();
        add(lpos[t], -1);
        lpos[t] = i;
        add(lpos[t], 1);
    }

    int q, l, r;
    cin >> q;
    while(q--)
    {
        cin >> l >> r;
        cout << get(l, r + 1, root[r]) << "\n";
    }
    return 0;
 }
```

Persistent Tree (Distinct Elements)

```
// number of distincit elements in range using persistent segment tree

#include <bits/stdc++.h>
//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>
```

```cpp
//using namespace __gnu_pbds;
using namespace std;

//typedef tree<int , null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
typedef vector<int> vi;
typedef long long ll;

#define pb push_back
#define inf 0x3f3f3f3f
#define all(v) (v).begin() , (v).end()
#define ones(n) __builtin_popcount(n)
#define watch(x) cout << (#x) << " is " << (x) << endl

int di[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dj[] = {1, -1, 0, 0, -1, 1, 1, -1};

const int N = 1e5 + 5;
int n, k;
int arr[N];
map<int, vi> adj;

struct node {
    int sum = 0;
    node *left, *right;

    node(int sum, node *left, node *right) : sum(sum), left(left),
right(right) {}

    node(int sum = 0) : sum(sum), left(this), right(this) {}
};

node *insert(node *cur, int v, int l = -inf, int r = inf) {
    if (v < l || v > r)
        return cur;
    if (l == r)
        return new node(cur->sum + 1);
    int mid = l + ((r - l) >> 1);
    node *left = insert(cur->left, v, l, mid);
    node *right = insert(cur->right, v, mid + 1, r);
    return new node(left->sum + right->sum, left, right);
}

node *roots[N];

int query(node *bs, node *e, int v, int l = -inf, int r = inf) {
    if (l == r)
        return 0;
    int mid = l + ((r - l) >> 1);
    int leftCnt = e->left->sum - bs->left->sum;
    if (v <= mid)
        return query(bs->left, e->left, v, l, mid);
    return leftCnt + query(bs->right, e->right, v, mid + 1, r);
}
```

```
int main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
#endif
    scanf("%d", &n);
    k = 1;
    roots[0] = new node;
    for (int i = 1; i <= n; i++) {
        scanf("%d", arr + i);
        int x = adj[arr[i]].size() >= k ? adj[arr[i]][adj[arr[i]].size() - k]
: 0;
        adj[arr[i]].pb(i);
        roots[i] = insert(roots[i - 1], x);
    }
    int q, l, r;
    scanf("%d", &q);
    while (q--) {
        scanf("%d%d", &l, &r);
        printf("%d\n", query(roots[l - 1], roots[r], l));
    }
}
```

Persistent Trie

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

using namespace std;

const int N = 1e5 + 5;

struct node {
    int sum = 0;
    node *left, *right;
    node(int sum, node *left, node *right) : sum(sum), left(left),
right(right) {}
    node(int sum = 0) : sum(sum), left(this), right(this) {}
};

node *insert(node *cur, int i , int v) {
    if (i == 0)
        return new node(cur->sum + 1);

    node *left = cur->left;
    node *right = cur->right;

    if (v & (1 << (i - 1)))
        left = insert(cur->left , i - 1, v);
    else
        right = insert(cur->right , i - 1, v);

    return new node(cur->sum + 1, left, right);
}

node *roots[N];
```

```
int query(node *bs, node *e , int i , int v) {
    if (i == 0)
        return 0;

    int leftCnt = e->left->sum - bs->left->sum;
    int rightCnt = e->right->sum - bs->right->sum;

    int j = i - 1;
    if (v & (1 << j)) {
        if (rightCnt) return (1 << j) + query(bs->right, e->right, i - 1, v);
        return query(bs->left , e->left , i - 1 , v);
    }

    if (leftCnt) return (1 << j) + query(bs->left , e->left , i - 1 , v);
    return query(bs->right, e->right, i - 1, v);
}


int main() {
    int T , n , q;
    scanf("%d" , &T);

    while (T--) {
        scanf("%d%d" , &n , &q);

        roots[0] = new node;
        int x;
        for (int i = 1 ;i <= n ;i++) {
            scanf("%d" , &x);
            roots[i] = insert(roots[i - 1] , 16 , x);
        }

        int l , r , a;
        while (q--) {
            scanf("%d%d%d" , &a , &l , &r);
            printf("%d\n" , query(roots[l - 1] , roots[r] ,  16 , a));
        }
    }
}
```

Segment Tree (Pointers and Lazy)

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

//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

//using namespace __gnu_pbds;
using namespace std;

//template<typename T>
//using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
```

```cpp
typedef vector<int> vi;
typedef long long ll;

#define pb push_back
#define inf 0x3f3f3f3f
#define all(v) (v).begin() , (v).end()
#define ones(n) __builtin_popcount(n)

int di[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dj[] = {1, -1, 0, 0, -1, 1, 1, -1};

struct node {
    ll sum , lazy;
    node *left , *right;
    node () : sum(0) , lazy(0) , left(0) , right(0) {};
    ~node() {delete left , delete right;}
};

using nodePtr = node*;

class segTree {
    const int LN = 0 , RN = 1e5 + 5; // can be negative
    nodePtr root;

    ll getSum(nodePtr &a) {
        return a ? a->sum : 0;
    }

    void addSum(nodePtr &a , ll v) {
        if (!a) a = new node();
        a->sum += v;
    }

    void addLazy(nodePtr &a , ll v) {
        if (!a) a = new node();
        a->lazy += v;
    }

    void pushDown(nodePtr &cur , ll l , ll r) {
        if (cur && cur->lazy) {
            cur->sum += cur->lazy * (r - l + 1);
            addLazy(cur->left , cur->lazy);
            addLazy(cur->right , cur->lazy);
            cur->lazy = 0;
        }
    }

    void in (ll i , ll v , nodePtr &cur , ll l , ll r) {
        if (i > r || i < l)
            return;

        addSum(cur , v);
        if (l == r)
            return;
```

```
        ll mid = l + (r - l) / 2;
        in(i , v  , cur->left, l , mid);
        in(i , v  ,cur->right , mid + 1 , r);
    }

    ll query (ll s , ll e , nodePtr &cur , ll l , ll r) {
        pushDown(cur , l , r);
        if (!cur || l > e || r < s)
            return 0;
        if (l >= s && r <= e)
            return cur->sum;
        ll mid = l + (r - l) / 2;
        return query(s , e , cur->left , l , mid) + query(s , e , cur->right ,
mid + 1 , r);
    }

    void update(ll s , ll e , ll v , nodePtr &cur , ll l , ll r) {
        pushDown(cur , l , r);
        if (l > e || r < s)
            return ;

        addSum(cur , 0); // to handle if cur = null

        if (l >= s && r <= e) {
            cur->sum += v * (r - l + 1);
            addLazy(cur->left , v);
            addLazy(cur->right , v);
            return;
        }

        ll mid = l + (r - l) / 2;
        update(s , e , v , cur->left , l , mid);
        update(s , e , v,  cur->right , mid + 1 , r);
        cur->sum = getSum(cur->left) + getSum(cur->right);
    }
public:
    segTree() : root(0) {}
    ~segTree() {delete root;}
    void in(ll i , ll v) {
        in(i , v , root , LN , RN);
    }

    ll query(ll s , ll e) {
        return query(s , e , root , LN , RN);
    }

    void update(ll s , ll e , ll v) {
        update(s , e , v , root , LN , RN);
    }
};


int main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
```

```
#endif
    int T , n , q;
    scanf("%d" , &T);

    while (T--) {
        scanf("%d%d" , &n , &q);
        segTree seg;
        int type , l , r , v;
        while (q--) {
            scanf("%d%d%d" , &type , &l , &r);
            if (!type) {
                scanf("%d" , &v);
                seg.update(l , r , v);
            } else {
                printf("%lld\n" , seg.query(l , r));
            }
        }
    }
}
```

Segment Tree Lazy (Error)

```
#define ll long long
const int N=200006;
ll seg[4*N],lazy[4*N],a[N];
// long long
int n;
void push(int id,int l,int r)
{
    seg[id]+=(r-l+1)*lazy[id];
    if(l==r){
        lazy[id]=0;
        return;
    }
    lazy[id*2]+=lazy[id];
    lazy[id*2+1]+=lazy[id];
    lazy[id]=0;
}
int query(int sq,int eq,int id=1,int s=0,int e=n-1)
{
    push(id,s,e);
    if(s>eq||e<sq)
        return 0;
    if(s>=sq&&e<=eq)
        return seg[id];
    int mid=(s+e)/2;
    return query(sq,eq,id*2,s,mid)+query(sq,eq,id*2+1,mid+1,e);
}
void update(int su,int eu,int v,int id=1,int s=0,int e=n-1)
{
    push(id,s,e);
    if(s>eu||e<su)
        return;
    if(s>=su&&e<=eu)
```

```
    {
        lazy[id]+=v;
        push(id,s,e);
        return;
    }
    int mid=(s+e)/2;
    update(su,eu,v,id*2,s,mid);
    update(su,eu,v,id*2+1,mid+1,e);
    seg[id]=seg[id*2]+seg[id*2+1];
}
```

Segment Tree (Lazy)

```
const int M = 1e5 + 5;
const int N = 4*M;
int s[N] , a[M] , lazy[N];
int n;
void build(int id = 1,int l = 0,int r = n){
      if(r - l < 2){      //     l + 1 == r
            s[id] = a[l];
            return ;
      }
      int mid = (l+r)/2;
      build(id * 2, l, mid);
      build(id * 2 + 1, mid, r);
      s[id] = s[id * 2] + s[id * 2 + 1];
}
void upd(int id,int l,int r,int x){
      lazy[id] += x;
      s[id] += (r - l) * x;
}
void shift(int id,int l,int r){//pass update information to the children
      int mid = (l+r)/2;
      upd(id * 2, l, mid, lazy[id]);
      upd(id * 2 + 1, mid, r, lazy[id]);
      lazy[id] = 0;// passing is done
}
void increase(int x,int y,int v,int id = 1,int l = 0,int r = n){
      if(x >= r or l >= y)      return ;
      if(x <= l && r <= y){
            upd(id, l, r, v);
            return ;
      }
      shift(id, l, r);
      int mid = (l+r)/2;
      increase(x, y, v, id * 2, l, mid);
      increase(x, y, v, id*2+1, mid, r);
      s[id] = s[id * 2] + s[id * 2 + 1];
}
int sum(int x,int y,int id = 1,int l = 0,int r = n){
      if(x >= r or l >= y)      return 0;
      if(x <= l && r <= y)      return s[id];
      shift(id, l, r);
      int mid = (l+r)/2;
      return sum(x, y, id * 2, l, mid) +
```

```
            sum(x, y, id * 2 + 1, mid, r);
}
```

Segment Tree

```
const int M = 1e5 + 5;
const int N = 4*M;
int s[N] , a[M];
int n;
void build(int id = 1,int l = 0,int r = n){
      if(r - l < 2){      //     l + 1 == r
            s[id] = a[l];
            return ;
      }
      int mid = (l+r)/2;
      build(id * 2, l, mid);
      build(id * 2 + 1, mid, r);
      s[id] = s[id * 2] + s[id * 2 + 1];
}
void modify(int p,int x,int id = 1,int l = 0,int r = n){
      s[id] += x - a[p];
      if(r - l < 2){      //     l = r - 1 = p
            a[p] = x;
            return ;
      }
      int mid = (l + r)/2;
      if(p < mid)
            modify(p, x, id * 2, l, mid);
      else
            modify(p, x, id * 2 + 1, mid, r);
}
int sum(int x,int y,int id = 1,int l = 0,int r = n){
      if(x >= r or l >= y)      return 0;
      if(x <= l && r <= y)      return s[id];
      int mid = (l+r)/2;
      return sum(x, y, id * 2, l, mid) +
            sum(x, y, id * 2 + 1, mid, r);
}
```

Segment Tree OOP

```
class SegmentTree {                                 // OOP style
private:
    vl A;                                             // the underlying array
    int n;                                            // n = (int)A.size()
    vl st;                                            // segment tree array
    vl lazy;                                          // lazy propagation array

    int l(int p) { return  p<<1; }                    // go to left child
    int r(int p) { return (p<<1)+1; }                 // go to right child

    ll conquer(ll a, ll b) {
```

```
        if (a == -1) return b;                          // corner case
        if (b == -1) return a;
        return a <= b ? a : b;                  // RMQ
    }

    void build(int p, int L, int R) {               // O(n)
        if (L == R)
            st[p] = L;                                  // base case
        else {
            int m = (L+R)/2;
            build(l(p), L  , m);
            build(r(p), m+1, R);
            st[p] = conquer(st[l(p)], st[r(p)]);
        }
    }

    void propagate(int p, int L, int R) {
            st[p] += lazy[p];                                   // [L..R] has
same value
            if (L != R){
                lazy[l(p)] +=  lazy[p];       // propagate downwards
                lazy[r(p)] += lazy[p];
            }                                               // time to update
this
            lazy[p] = 0;                                // erase lazy flag
    }

    void update(int p, int L, int R, int i, int j, ll val) { // O(log n)
        propagate(p, L, R);                             // lazy propagation
        if (i > j) return;
        if ((L >= i) && (R <= j)) {                 // found the segment
            lazy[p] += val;                             // update this
            propagate(p, L, R);                       // lazy propagation
        }
        else {
            int m = (L + R) / 2;
            update(l(p), L, m, i, min(m, j), val);
            update(r(p), m + 1, R, max(i, m + 1), j, val);
            st[p] = conquer(st[l(p)], st[r(p)]);
        }
    }

    ll RMQ(int p, int L, int R, int i, int j) {   // O(log n)
        propagate(p, L, R);                             // lazy propagation
        if (i > j) return -1;                           // infeasible
        if ((L >= i) && (R <= j)) return st[p];     // found the segment
        int m = (L + R) / 2;
        return conquer(RMQ(l(p), L, m, i, min(m, j)),
                       RMQ(r(p), m + 1, R, max(i, m + 1), j));
    }

public:
    SegmentTree(int sz) : n(sz + 1), st(4*n), lazy(4*n) {}

    SegmentTree(const vl &_A) : SegmentTree((int)_A.size()) {
```

```
        A = _A;
        build(1, 1, n);
    }

    void update(int i, int j, ll val) { update(1, 1, n, i, j, val); }

    ll RMQ(int i, int j) { return RMQ(1, 1, n, i, j); }
};
```

Sparse Table 1D

```
int LG[100100];
int arr[100100];
int sTable[100100][17];
void build() {
  LG[0] = -1;
  for (int i = 0; i < n; i++) {
    LG[i + 1] = LG[i] + !(i & (i + 1));
    sTable[i][0] = i;
  }
  for (int j = 1; (1 << j) <= n; j++)
    for (int i = 0; i + (1 << j) <= n; i++) {
      int a = sTable[i][j - 1];
      int b = sTable[i + (1 << (j - 1))][j - 1];
      sTable[i][j] = (arr[a] < arr[b] ? a : b);
    }
}
int findMinIdx(int s, int e) {
  int len = e - s + 1;
  int lg = LG[len];
  int a = sTable[s][lg];
  int b = sTable[e - (1 << lg) + 1][lg];
  return (arr[a] < arr[b] ? a : b);
}
```

Sparse Table 2D

```
short get(int x1, int y1, int x2, int y2) {
    int lenx = x2 - x1 + 1;
    int kx = sv[lenx];
    int leny = y2 - y1 + 1;
    int ky = sv[leny];
    short ret = 0;
    ret = max(ret, max(tb[kx][x1][ky][y1], tb[kx][x2 - (1 << kx) +
1][ky][y1]));
    ret = max(ret, max(tb[kx][x1][ky][y2 - (1 << ky) + 1], tb[kx][x2 - (1 <<
kx) + 1][ky][y2 - (1 << ky) + 1]));
    return ret;
}
void build() {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
```

```
            tb[0][i][0][j] = a[i][j];
        }
    for (int j = 1; (1 << j) <= m; j++)
        for (int ii = 1; ii <= n; ii++)
            for (int jj = 1; jj + (1 << (j - 1)) <= m; jj++)
                tb[0][ii][j][jj] = max(tb[0][ii][j - 1][jj], tb[0][ii][j -
1][jj + (1 << (j - 1))]);
    for (int i = 1; (1 << i) <= n; i++)
        for (int j = 0; (1 << j) <= m; j++) {

            for (int ii = 1; ii + (1 << (i - 1)) <= n; ii++)
                for (int jj = 1; jj + (1 << (j - 1)) <= m; jj++)
                    tb[i][ii][j][jj] = max(
                            tb[i - 1][ii][j][jj],
                            tb[i - 1][ii + (1 << (i - 1))][j][jj]
                    );
        }
}
```

Splay Tree Implicit

```
// http://w...content-available-to-author-only...j.com/problems/CERC07S/
#include <bits/stdc++.h>

using namespace std;
typedef int vt;

struct node;

node *empty;

struct node {
    node *ch[2], *p;
    vt v;
    int sz;
    bool isrev;

    node() :  p(this), v(0), sz(0), isrev(0) {
        ch[0] = ch[1] = this;
    }

    node(vt v) : v(v), sz(1), p(empty), isrev(0) {
        ch[0] = ch[1] = empty;
    }
};

node *root;

inline void setlazy(node *nd) {

    if (nd == empty) return;
    nd->isrev ^= 1;
    swap(nd->ch[0], nd->ch[1]);
}
```

```
inline void push_up(node *nd) {
    nd->sz = 1 + nd->ch[0]->sz + nd->ch[1]->sz;
}

inline void push_down(node *nd) {
    if (nd->isrev) {
        setlazy(nd->ch[0]);
        setlazy(nd->ch[1]);
        nd->isrev = 0;
    }
}

inline void setchild(node *p, node *c, int idx) {
    if (p != empty) {
        p->ch[idx] = c;
    }
    c->p = p;
}

inline int getdir(node *p, node *c) {
    return p->ch[1] == c;
}
//     nd
//    / \
//   ch   c
//  / \
// a   b

//   ch
//  / \
// a   nd
//    / \
//   b   c
inline void rotate(node *nd, int idx) {
    node *ch = nd->ch[idx], *p = nd->p;
    setchild(nd, ch->ch[!idx], idx);
    setchild(ch, nd, !idx);

    setchild(p, ch, getdir(p, nd));

    push_up(nd);
    push_up(ch);
}

//the path from nd to root MUST be without lazy
inline void splay(node *nd) {
    while (nd->p != empty) {
        node *p = nd->p, *g = p->p;
        int pi = getdir(p, nd), gi = getdir(g, p);
        if (g == empty) {
            rotate(p, pi);
        } else if (pi == gi) { // TODO why log N?
            rotate(g, gi);
            rotate(p, gi);
```

```
        } else {
            rotate(p, pi);
            rotate(g, gi);
        }
    }
    root = nd;
}

inline node *getNodeAt(node *t, int idx) {
    idx = min(idx, t->sz - 1);
    while (push_down(t), t->ch[0]->sz != idx) {
        if (t->ch[0]->sz < idx) {
            idx -= t->ch[0]->sz + 1;
            t = t->ch[1];
        } else {
            t = t->ch[0];
        }
    }
    splay(t);
    return t;
}

inline void split(node *t, int presz, node *&pre, node *&suf) {
    if (presz == 0) {
        pre = empty;
        suf = t;
    } else {
        pre = getNodeAt(t, presz - 1);
        suf = pre->ch[1];
        pre->ch[1] = suf->p = empty;
        push_up(pre);
    }
}

inline void merge(node *pre, node *suf, node *&t) {
    if (pre == empty) {
        t = suf;
    } else {
        t = getNodeAt(pre, pre->sz - 1);
        setchild(t, suf, 1);
        push_up(t);
    }

}

inline void push_down_to_root(node *nd) {
    if (nd->p != empty) {
        push_down_to_root(nd->p);
    }
    push_down(nd);
}

inline void reverse(node *&t, int st, int en) {
    node *before, *mid, *after;
    split(t, en + 1, before, after);
```

```
    split(before, st, before, mid);
    setlazy(mid);
    merge(before, mid, before);
    merge(before, after, t);
}

const int N = 100005;

int n;
pair<int, node *> a[N];

int main() {
//    freopen("in.txt", "r", stdin);
    empty = new node();
    while (scanf("%d", &n), n) {
        root = empty;
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i].first);
            a[i].second = new node(a[i].first);
            merge(root, a[i].second, root);
        }
        stable_sort(a, a + n, [](const pair<int, node *> &x, const pair<int,
node *> &y) {
            return x.first < y.first;
        });
        for (int i = 0; i < n; i++) {
            push_down_to_root(a[i].second);
            splay(a[i].second);
            printf("%d%c", root->ch[0]->sz + 1, " \n"[i == n - 1]);
            reverse(root, i, root->ch[0]->sz);
        }
    }
    return 0;
}
```

Splay Tree

```
//http://w...content-available-to-author-only...j.com/problems/KPMATRIX/
//#pragma GCC optimize ("O3")
#include <bits/stdc++.h>
using namespace std;
struct node {
    node *ch[2];
    int  freq, sz;
    long long v;
    node() : v(0), ch({this, this}), freq(0), sz(0) {}
};
node *empty = new node();
void pushup(node *nd) {
    nd->sz = nd->ch[0]->sz + nd->ch[1]->sz + nd->freq;
}
//    p
//   / \
//  q   c
```

```
// / \
//a   b

//    q
//   / \
//  a   p
//     / \
//    b   c
void rotate(node *&nd, int idx) {
    node *p = nd, *q = nd->ch[idx];
    p->ch[idx] = q->ch[!idx];
    q->ch[!idx] = p;
    nd = q;
    pushup(p);
    pushup(q);
}
inline int getidx(node *nd, long long v) {
    return (v > nd->v);
}
void splay(node *&nd, long long v) {
    if (nd == empty) return;
    if (nd->v == v) return;
    for (int i = 0; i < 2; ++i) {
        if (nd->ch[i]->v == v && nd->ch[i] != empty) {
            rotate(nd, i);
            return;
        }
    }


    int c1 = getidx(nd, v);
    if (nd->ch[c1] == empty) return;
    int c2 = getidx(nd->ch[c1], v);
    splay(nd->ch[c1]->ch[c2], v);
    if (nd->ch[c1]->ch[c2] == empty) {
        rotate(nd, c1);
        return;
    }
    if (c1 == c2) {
        rotate(nd, c1);
        rotate(nd, c1);
    } else {
        rotate(nd->ch[c1], c2);
        rotate(nd, c1);
    }
}
void split(node *t, long long v, node *&le, node *&g) {
    if (t == empty) {
        le = g = empty;
        return;
    }
    splay(t, v);
    if (t->v <= v) {
        le = t;
        g = t->ch[1];
```

```
        t->ch[1] = empty;
        pushup(t);
    } else {
        g = t;
        le = t->ch[0];
        t->ch[0] = empty;
        pushup(t);
    }
}
long long getmax(node *t) {
    return t->ch[1] == empty ? t->v : getmax(t->ch[1]);
}
void merge(node *le, node *g, node *&t) {
    if (le == empty) t = g;
    else {
        splay(le, getmax(le));
        le->ch[1] = g;
        pushup(le);
        t = le;
    }
}
void insert(node *&t, long long v) {

    splay(t, v);
    if (t != empty && t->v == v) {
        t->freq++;
        t->sz++;
        return;
    }
    node *l, *g;
    split(t, v, l, g);
    t = new node();
    t->v = v;
    t->freq = 1;
    t->ch[0] = l;
    t->ch[1] = g;
    pushup(t);
}
int lower_bound(node *&t, long long v) {
    splay(t, v);
    int ret = t->ch[0]->sz;
    if (t->v < v) ret += t->freq;
    return ret;
}
int upper_bound(node *&t, long long v) {
    return lower_bound(t, v + 1);
}
const int N = 255;
long long mat[N][N];
int main() {
   // freopen("in.txt", "rt", stdin);
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
```

```
            scanf("%lld", &mat[i][j]);
            mat[i][j] += mat[i][j - 1];
        }
    }
    long long a, b, res = 0;
    scanf("%lld %lld", &a, &b);
    for (int s = 0; s < m; ++s) {
        for (int e = s + 1; e <= m; ++e) {
            node *root = empty;
            insert(root, 0);
            long long sum = 0;
            for (int i = 1; i <= n; ++i) {
                sum += mat[i][e] - mat[i][s];
                res += upper_bound(root, sum - a) - lower_bound(root, sum -
b);

                insert(root, sum);
            }
        }
    }
    printf("%lld\n", res);
    return 0;
}
```

Treap

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

//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

//using namespace __gnu_pbds;
using namespace std;

//template<typename T>
//using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

typedef vector<int> vi;
typedef long long ll;

#define pb push_back
#define inf 0x3f3f3f3f
#define all(v) (v).begin() , (v).end()
#define ones(n) __builtin_popcount(n)
#define ONES(n) __builtin_popcountll(n)

int di[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dj[] = {1, -1, 0, 0, -1, 1, 1, -1};




struct node {
```

```
    int key , prior , cnt ;
    ll s , v , lazy;
    node * left , *right;
    node (int _key , int _prior = rand()) : v(0) , key(_key) , prior(_prior) ,
left(0) , right(0) , lazy(0) {}
    ~node() { delete left , delete right;}
};

using nodePtr = node*;

class treap {
    nodePtr root;

    ll sum(nodePtr a) {
        pushDown(a);
        return a ? a->s: 0;
    }

    int cnt(nodePtr a) {
        return a ? a->cnt : 0;
    }

    void update(nodePtr a) {
        if (a) {
            a->s = a->v + sum(a->left) + sum(a->right);
            a->cnt = 1 + cnt(a->left) + cnt(a->right);
        }
    }

    void pushDown(nodePtr &a) {
        if (a) {
            a->s += a->lazy * a->cnt , a->v += a->lazy;
            if (a->left) a->left->lazy += a->lazy;
            if (a->right) a->right->lazy += a->lazy;
            a->lazy = 0;
        }
    }

    void insert(nodePtr &cur , nodePtr nw) {
        pushDown(cur);
        if (!cur)
            cur = nw;
        else if (cur->prior > nw->prior)
            insert(nw->key <= cur->key ? cur->left : cur->right , nw);
        else
            split(cur , nw->left , nw->right , nw->key) , cur = nw;

        update(cur);
    }

    void split (nodePtr cur , nodePtr &l , nodePtr &r , int key) {
        pushDown(cur);
        if (!cur)
            l = r = 0;
        else if (key >= cur->key)
```

```
            split(cur->right , cur->right , r , key) , l = cur;
        else
            split(cur->left , l , cur->left , key) , r = cur;

        update(cur);
    }

    void merge (nodePtr &cur , nodePtr l , nodePtr r) {
        pushDown(cur);
        if (!l || !r)
            cur = l ? l : r;
        else if (l->prior > r->prior)
            merge(l->right , l->right , r) , cur = l;
        else
            merge(r->left , l , r->left) , cur = r;
        update(cur);
    }

public:
    treap() : root(0) {}
    ~treap() {delete root;}

    ll query(int l , int r) {
        nodePtr a , b , c , d;
        split(root , a , b , l - 1);
        split(b , c , d , r);
        ll ans = sum(c);
        merge(b , c , d);
        merge(root , a , b);
        return ans;
    }

    void update(int l , int r , int val) {
        nodePtr a , b , c , d;
        split(root , a , b , l - 1);
        split(b , c , d , r);
        c->lazy += val;
        pushDown(c);
        merge(b , c , d);
        merge(root , a , b);
    }

    void insert(int key) {
        insert(root , new node(key));
    }

};

int main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
#endif
    int T , n , q;
    scanf("%d" , &T);
    while (T--) {
```

```
        scanf("%d%d" , &n , &q);

        treap t;
        for (int i = 1 ;i <= n ;i++)
            t.insert(i);

        int type , l , r , val;
        while (q--) {
            scanf("%d%d%d" , &type , &l , &r);
            if (!type) {
                scanf("%d" , &val);
                t.update(l , r , val);
            } else {
                printf("%lld\n" , t.query(l , r));
            }
        }
    }
}
```

Treap Implicit Key

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

//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

//using namespace __gnu_pbds;
using namespace std;

//template<typename T>
//using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

typedef vector<int> vi;
typedef long long ll;

#define pb push_back
#define inf 0x3f3f3f3f
#define all(v) (v).begin() , (v).end()
#define ones(n) __builtin_popcount(n)
#define ONES(n) __builtin_popcountll(n)

int di[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dj[] = {1, -1, 0, 0, -1, 1, 1, -1};

struct node {
    int prior , cnt , val;
    int lazy;
    node * left , *right;
    node (int _val) : val(_val) , prior(rand()) , cnt(1) , left(0) , right(0)
, lazy(0) {}
    ~node() { delete left , delete right;}
};
```

```
using nodePtr = node*;

class Treap {
    nodePtr root;

    int cnt(nodePtr a) {
        return a ? a->cnt : 0;
    }

    void update(nodePtr a) {
        if (a) a->cnt = 1 + cnt(a->left) + cnt(a->right);
    }

    void pushDown(nodePtr &a) {
        if (a && a->lazy) {
            swap(a->left , a->right);
            if (a->left) a->left->lazy ^= 1;
            if (a->right) a->right->lazy ^= 1;
            a->lazy = 0;
        }
    }


    void split (nodePtr cur , nodePtr &l , nodePtr &r , int key , int curKey =
0) {
        pushDown(cur);
        if (!cur)
            return void(l = r = 0);

        curKey += cnt(cur->left) + 1;
        if (key >= curKey)
            split(cur->right , cur->right , r , key , curKey) , l = cur;
        else
            split(cur->left , l , cur->left , key , curKey - cnt(cur->left) -
1) , r = cur;

        update(cur);
    }

    void merge (nodePtr &cur , nodePtr l , nodePtr r) {
        pushDown(l) , pushDown(r);
        if (!l || !r)
            cur = l ? l : r;
        else if (l->prior > r->prior)
            merge(l->right , l->right , r) , cur = l;
        else
            merge(r->left , l , r->left) , cur = r;
        update(cur);
    }

public:
    Treap() : root(0) {}
    ~Treap() {delete root;}

    void insert (int pos , int v) {
```

```
        nodePtr a , b , c = new node(v);
        split(root , a , b , pos - 1);
        merge(a , a , c);
        merge(root , a , b);
    }

    void cyclic_shift(int l , int r) {
        nodePtr a , b , c , d;
        split(root , a , b , r);
        split(a , a , c , r - 1);
        split(a , d , a , l - 1);
        merge(a , c , a);
        merge(a , a , b);
        merge(root , d , a);
    }

    void reverse(int l , int r) {
        nodePtr a , b , c;
        split(root , a , b , r);
        split(a , c , a , l - 1);
        a->lazy ^= 1;
        merge(a , c , a);
        merge(root , a , b);
    }

    int at(int i) {
        nodePtr cur = root;
        int curKey = cnt(cur->left) + 1;
        while (curKey != i) {
            if (i > curKey)
                cur = cur->right;
            else {
                curKey -= cnt(cur->left) + 1;
                cur = cur->left;
            }
            pushDown(cur);
            curKey += cnt(cur->left) + 1;
        }
        return cur->val;
    }

};

int main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
#endif
    int n , q , m , x;
    scanf("%d%d%d" , &n , &q , &m);

    Treap t;
    for (int i = 1 ;i <= n ;i++) {
        scanf("%d" , &x);
        t.insert(i , x);
    }
```

```
    int type , l , r;
    while (q--) {
        scanf("%d%d%d" , &type , &l , &r);
        if (type == 1)
            t.cyclic_shift(l , r);
        else
            t.reverse(l , r);

    }

    for (int i = 0 ;i < m ;i++) {
        scanf("%d" , &x);
        printf("%d " , t.at(x));
    }
}
```

Wavelet Tree

```
// number of elements greater than or equal k in range [l , r]

#include <bits/stdc++.h>
//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>

//using namespace __gnu_pbds;
using namespace std;

//typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
typedef vector<int> vi;
typedef long long ll;

#define pb push_back
#define inf 0x3f3f3f3f
#define all(v) (v).begin() , (v).end()
#define ones(n) __builtin_popcount(n)
#define watch(x) cout << (#x) << " is " << (x) << endl

int di[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dj[] = {1, -1, 0, 0, -1, 1, 1, -1};

const int N = 3e4 + 5;

struct node {
    vi arr, freq;
    int mn, mx , md;
    node *left, *right;

    node() {
        arr = {0};
    }

    void go() {
        mn = *min_element(arr.begin() + 1 , arr.end());
```

```
        mx = *max_element(arr.begin() + 1 , arr.end());
        if (mn == mx)
            return;

        freq = vi(arr.size(), 0);
        left = new node;
        right = new node;

        md = mn + (mx - mn) / 2;
        for (int i = 1; i < arr.size(); i++) {
            if (arr[i] <= md) {
                left->arr.pb(arr[i]);
                freq[i] = freq[i - 1] + 1;
            } else {
                right->arr.pb(arr[i]);
                freq[i] = freq[i - 1];
            }
        }

        left->go();
        right->go();
    }

    int query(int s, int e, int k) {
        if (mn == mx)
            return mn > k ? e - s + 1: 0;

        int cnt = freq[e] - freq[s - 1];

        int ret = 0;
        if(k <= md) {
            ret = (e - s + 1) - cnt;
            if(cnt)
                ret += left->query(freq[s - 1] + 1 , freq[e] , k);
        } else if((e - s + 1) - cnt)
            ret = right->query(s - freq[s - 1] , e - freq[e] , k);

        return ret;
    }
};

int main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
#endif
    int n, x;
    scanf("%d", &n);

    node root;
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        root.arr.pb(x);
    }

    root.go();
```

```
    int q, s, e, k;
    scanf("%d", &q);
    while (q--) {
        scanf("%d %d %d", &s, &e, &k);
        printf("%d\n", root.query(s, e, k));
    }
}
```

Sqrt Decomposition

```
struct SqareRootDecompo {
    struct block {
        int lazy = 0;
        vector<pair<int , int> > v;

        void update() {
            // update the whole block
        }

        void update(int l , int r) {
            // update a range in the block
        }

        int get() {
            // query about the whole block
        }

        int get(int l , int r) {
            // query about a range in the block
        }
    };
    int n , bsz , bcnt;
    vector<block> blocks;
    SqareRootDecompo(vector<int> &a) {
        n = a.size();
        bsz = sqrt(n) + 1;
        bcnt = (a.size() - 1) / bsz + 1;
        blocks = vector<block>(bcnt);

        for (int i = 0 ;i < n ;i++)
            blocks[i / bsz].v.push_back({a[i] , i});

        for (int i = 0 ;i < bcnt ;i++)
            sort(blocks[i].v.begin() , blocks[i].v.end());
    }
    void update(int l , int r) {
        for (int i = l ;i <= r ;) {
            if (i % bsz == 0 && min(i + bsz - 1 , n - 1) <= r) {
                blocks[i / bsz].update();
                i += bsz;
            } else {
                int bid = i / bsz;
                blocks[bid].update(i , min(r , (bid + 1) * bsz - 1));
```

```
                i = (bid + 1) * bsz;
            }
        }
    }
    int query(int l , int r) {
        int ret = 0;
        for (int i = l ;i <= r ;) {
            if (i % bsz == 0 && min(i + bsz - 1 , n - 1) <= r) {
                ret += blocks[i / bsz].get();
                i += bsz;
            } else {
                int bid = i / bsz;
                ret += blocks[bid].get(i , min(r , (bid + 1) * bsz - 1));
                i = (bid + 1) * bsz;
            }
        }
        return ret;
    }
};
```

SQRT Decomposition MO

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
const int SQRT = sqrt(N) + 1;//can be calculated for each test case
pair<int,pair<int,int > > q[N];
int v[N] , cur[N] , a[N];
int n , m , ans;
void add(int x){
    if(x >= N)return;
    if(cur[x] == x)ans--;
    cur[x]++;
    if(cur[x] == x)ans++;
}
void del(int x){
    if(x >= N)return;
    if(cur[x] == x)ans--;
    cur[x]--;
    if(cur[x] == x)ans++;
}
bool cmp(pair<int,pair<int,int> > a , pair<int,pair<int,int > > b){
    if(a.first / SQRT != b.first / SQRT)return a.first  < b.first ;
    return a.second.first < b.second.first;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i = 0 ; i < n; i++){
        scanf("%d",&v[i]);
    }
    for(int i = 0 ; i < m ; i++){
        scanf("%d%d",&q[i].first,&q[i].second);
        q[i].first-- , q[i].second.first--;
        q[i].second.second = i;
```

```
    }
    sort(q,q+m,cmp);
    int s = 0 , e = -1;
    for(int i = 0 ; i < m ; i++){
        while(e < q[i].second.first){
            add(v[++e]);
        }
       while(s > q[i].first){
            add(v[--s]);
        }
        while(e > q[i].second.first){
            del(v[e--]);
        }
        while(s < q[i].first){
            del(v[s++]);
        }
        a[q[i].second.second] = ans;
    }
    for(int i = 0 ; i < m ; i++)
        printf("%d\n",a[i]);
    return 0;
}
```

Dynamic Programming

Convex Hull Trick (log N)

```
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for
maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
```

```
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

Divide and Conquer

```
int n;
long long C(int i, int j);
vector<long long> dp_before(n), dp_cur(n);
// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr)
{
    if (l > r)
        return;
    int mid = (l + r) >> 1;
    pair<long long, int> best = {INF, -1};
    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {dp_before[k] + C(k, mid), k});
    }
    dp_cur[mid] = best.first;
    int opt = best.second;
    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}
```

Knuth optmization

```
for(int s = 0 ; s <= n+1 ; s++){
    for(int l = 0 ; l+s <= n+1 ; l++){
        int r = l + s;
        if(s < 2){
            dp[l][r] = 0;
            mid[l][r] = l;
            continue;
        }
        int mleft = mid[l][r-1] , mright = mid[l+1][r];
        dp[l][r] = INF;
        for(int i = mleft ; i <= mright ; i++){
            int cost = dp[l][i] + dp[i][r] + cut[r] - cut[l];
            if(dp[l][r] > cost){
                dp[l][r] = cost;
                mid[l][r] = i;
            }
        }
    }
}
```

LIS N log N

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

```
using namespace std;
int get(vector<int> &v, int l, int r, int key) {
    while (r - l > 1) {
        int m = l + (r - l) / 2;
        if (v[m] >= key)
            r = m;
        else
            l = m;
    }
    return r;
}
int lis(vector<int> &v) {
    if (v.size() == 0)
        return 0;
    vector<int> t(v.size(), 0);
    int length = 1;
    t[0] = v[0];
    for (int i = 1; i < v.size(); i++) {
        if (v[i] < t[0])
            t[0] = v[i];
        else if (v[i] > t[length - 1])
            t[length++] = v[i];
        else
            t[get(t, -1, length - 1, v[i])] = v[i];
    }
    return length;
}
```

SOS

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
      dp[mask][-1] = A[mask];   //handle base case separately (leaf states)
      for(int i = 0;i < N; ++i){
            if(mask & (1<<i))
                  dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
            else
                  dp[mask][i] = dp[mask][i-1];
      }
      F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
      F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
      if(mask & (1<<i))
            F[mask] += F[mask^(1<<i)];
}
```

Graphs

Dijkstra

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

```cpp
const int N = 1e5 + 5;
struct edge {
    int to;
    long long cost;
};
struct path {
    int node;
    long long cost;
    bool operator < (const path &other) const {
        return cost > other.cost;
    }
};
bool known[N];
long long dist[N];
vector<edge> adj[N];
void Dijkstra(int s) {
    priority_queue<path> pq;
    memset(known , 0 , sizeof known);
    memset(dist , 63 , sizeof dist);
    pq.push({s , 0});
    dist[s] = 0;
    while (pq.size()) {
        int node = pq.top().node;
        long long cost = pq.top().cost;
        pq.pop();

        if (known[node]) continue;
        known[node] = 1;

        for (auto &c : adj[node]) {
            if (dist[c.to] > dist[node] + c.cost) {
                dist[c.to] = dist[node] + c.cost;
                pq.push({c.to , dist[c.to]});
            }
        }
    }
}
```

Floyd

```cpp
int dist[N][N];
int next[N][N];
void path(int fr, int to) {
    if(next[fr][to] == -1) {
        cout << fr << " "; // beytalla3 kolo ma 3ada a5er node
        return;
    }
    path(fr, next[fr][to]);
    path(next[fr][to], to);
}
void floyd() {
    // dist[i][j] contains the weight of edge (i, j)
    // or INF (1B) if there is no such edge
    for(int k = 0; k < n; k++) {
        for(int i = 0; i < n; i++) {
```

```
            for(int j = 0; j < n; j++) {
                if(dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    next[i][j] = k;
                }
            }
        }
    }
}
```

Bellman Optimized

```
const int N = 1e3+1, M = 3e3+2, OO = 0x3f3f3f3f;

int n, ne, head[N], nxt[M], to[M], cost[M];

void init(){
	memset(head, -1, n*sizeof head[0]);
	ne = 0;
}

void addEdge(int f, int t, int c){
	to[ne] = t;
	cost[ne] = c;
	nxt[ne] = head[f];
	head[f] = ne++;
}


int dist[N];
int inQ[N], vid;
bool bellman(int src){
	memset(dist, OO, n*sizeof dist[0]);
	dist[src] = 0;
	queue<int> q;
	q.push(src);
	inQ[src] = ++vid;
	int x = n;
	int s;
	while(x-- && (s = q.size())){
		while(s--){
			int u = q.front();
			q.pop();
			inQ[u] = 0;
			for(int e = head[u] ; ~e ; e = nxt[e]){
				int v = to[e];
				int d = dist[u] + cost[e];
				if(dist[v] > d){
					dist[v] = d;
					if(inQ[v] != vid){
						q.push(v);
						inQ[v] = vid;
					}
					if(!x)  return 0;
				}
```

```
                    }
                }
        }
        return 1;
}

int t, m, u, v, c;

int main(){
        scanf("%d", &t);
        while(t--){
                scanf("%d %d", &n, &m);
                ++n;
                init();
                for(int i = 0 ; i < m ; ++i){
                        scanf("%d %d %d", &u, &v, &c);
                        addEdge(u, v, c);
                }
                for(int i = 0 ; i < n-1 ; ++i){
                        addEdge(n-1, i, 0);
                }
                puts(bellman(n-1) ? "not possible" : "possible");
        }

        return 0;
}
```

SPFA

```
void spfa() {
    memset(dist , '?' , sizeof dist);
    queue<int> q;
    q.push(src);
    dist[src] = 0;
    inq[src] = 1;
    for (int i = 0 ;i < n && q.size() ;i++) {
        int sz = q.size();
        while (sz--) {
            int cur = q.front();
            inq[cur] = 0;
            q.pop();
            for (auto &e : adj[cur]) {
                if (dist[e.to] > dist[cur] + e.cost) {
                    dist[e.to] = dist[cur] + e.cost;
                    par[e.to] = cur;
                    if (!inq[e.to]) {
                        inq[e.to] = 1;
                        q.push(e.to);
                    }
                }
            }
        }
    }
}
```

SCC Contract Graph

```
const int maxn=1e2+5;
const int N=1e4+5,M=N*20;
int *head;
int headG[N],nxt[M],to[M],ne,n;
void init(){
    ne=0;
    head=headG;
    memset(head,-1,n*sizeof head[0]);
}
void addEdge(int f, int t){
    to[ne]=t;
    nxt[ne]=head[f];
    head[f]=ne++;
}
int headC[N],visT[N],lo[N],stk[N],stkSz,compID[N],comps,curT;
void DFSTarjan(int u){
    visT[u]=lo[u]=curT++;
    stk[stkSz++]=u;
    for(int e=head[u];~e;e=nxt[e]){
        int v=to[e];
        if(visT[v]==-1){     //White
            DFSTarjan(v);
            lo[u]=min(lo[u],lo[v]);
        }
        else if(compID[v]==-1)     //Gray
            lo[u]=min(lo[u],lo[v]);
    }
    if(visT[u]==lo[u]){
        do{
            compID[stk[--stkSz]]=comps;
        }
        while(stk[stkSz]!=u);
        ++comps;
    }
}
void tarjan(){
    memset(compID,-1,n*sizeof compID[0]);
    memset(visT,-1,n*sizeof visT[0]);
    comps=curT=0;
    for(int i=0;i<n;++i)
        if(visT[i]==-1)
            DFSTarjan(i);
}
int notSrc[N],notSnk[N],vid,srcCnt,snkCnt,u,v;
void contractGraph(){
    head=headC;
    memset(headC,-1,comps*sizeof headC[0]);
    ++vid;
    srcCnt=snkCnt=comps;
    for(int u=0;u<n;++u){
        for(int e=headG[u];~e;e=nxt[e]){
            int v=to[e];
```

```
            int uu=compID[u],vv=compID[v];
            if(uu==vv)
                continue;
            addEdge(uu,vv);
            if(notSrc[vv]!=vid){
                notSrc[vv]=vid;
                --srcCnt;
            }
            if(notSnk[uu]!=vid){
                notSnk[uu]=vid;
                --snkCnt;
            }
        }
    }
}
int vis[N],vid2;
void dfs(int u){
    vis[u]=vid2;
    for(int e=head[u];~e;e=nxt[e]){
        int v=to[e];
        if(vis[v]!=vid2)
            dfs(v);
    }
}
int tc;
int main()
{
    int t;
    scanf("%d",&t);
    while(t--){
        int m;
        scanf("%d%d",&n,&m);
        init();
        while(m--){
            scanf("%d%d",&u,&v);
            addEdge(--u,--v);
        }
        tarjan();
        contractGraph();
        ++vid2;
        int cnt=0;
        for(int i=0;i<comps;++i)
            if(notSrc[i]!=vid2)
                dfs(i),++cnt;
        printf("Case %d: %d\n",++tc,cnt);
    }
    re 0;
}
```

Min Cut Ford

```
const int maxn=2e5+5;
const int N=1400+5,M=N*N;
int head[N],nxt[M],to[M],cap[M];
int n,ne;
```

```
void init()
{
	memset(head,-1,n*sizeof head[0]);
	ne=0;
}

void addEdge(int f, int t, int cp)
{
	to[ne]=t;
	nxt[ne]=head[f];
	cap[ne]=cp;
	head[f]=ne++;
}

void addAugEdge(int u, int v, int cp, int rev=0)
{
	addEdge(u,v,cp);
	addEdge(v,u,rev);
}

int src,snk;
int vis[N],vid;

int dfs(int u, int mn)
{
	if(u==snk)   return mn;
	if(!mn || vis[u]==vid)    return 0;
	vis[u]=vid;
	for(int e=head[u] ; ~e ; e=nxt[e])
	{
		int v=to[e];
		int f=dfs(v,min(mn,cap[e]));
		if(f){
			cap[e]-=f;
			cap[e^1]+=f;
			return f;
		}
	}
	return 0;
}

int MaxFlow()
{
	int flow=0,f;
	do
	{
		++vid;
		f=dfs(src,oo);
		flow+=f;
	}while(f);
	return flow;
}
vector<int> BFS(){
	queue<pair<int,int> > q;
	vector<int> lev(n,0);
```

```
	lev[0]=1;
	q.push({0,1});
	int Max=0;
	while(q.size()){
		int u=q.front().f;
		int l=q.front().s;
		Max=max(Max,l);
		q.pop();
		for(int e=head[u];~e;e=nxt[e]){
			int v=to[e];
			if(!lev[v]){
				lev[v]=lev[u]+1;
				q.push({v,lev[v]});
			}
		}
	}
	vector<int> ret;
	for(int i=0;i<n;++i)
		if(lev[i]==Max)
			ret.push_back(i);
	re ret;
}

int main()
{
	cin >> n;
	init();
	cin.ignore();
	vector<int> in(n,0);
	for(int i=0;i<n;++i){
		string s;
		getline(cin,s);
		istringstream iss(s);
		int x;
		while(iss >> x){
			addAugEdge(i,x,1);
			++in[x];
		}
	}
	vector<int> sinks=BFS();
	src=n;
	snk=n+1;
	addAugEdge(n,0,oo);
	for(auto i:sinks)
		addAugEdge(i,snk,oo);
	int Min=oo;
	for(int i=0;i<n;++i)
		Min=min(Min,in[i]);
	cout<<min(Min,MaxFlow())<<endl;
	re 0;
}
```

Get Pth Parent

```
const int maxn=1e5+5;
```

```
const int N=1e4+5,M=N*2;
int t,n,id;
string s;
int cost[N],cst[M];
int lev[N],in[N],out[N],sp[16][N];
int head[N],nxt[M],to[M],ne;
void init(){
	ne=0;
	memset(head,-1,n*sizeof head[0]);
}
void add_edge(int f,int t,int c){
	to[ne]=t;
	nxt[ne]=head[f];
	cst[ne]=c;
	head[f]=ne++;
}
void add_bi_edge(int a,int b,int c){
	add_edge(a,b,c);
	add_edge(b,a,c);
}
void dfs(int u,int par,int l,int c){
	in[u]=++id;
	lev[u]=l;
	cost[u]=c;
	sp[0][u]=par;
	for(int e=head[u];~e;e=nxt[e]){
		int v=to[e];
		if(v!=par)dfs(v,u,l+1,c+cst[e]);
	}
	out[u]=id;
}
void buildSparse(){
	for(int j=1;j<=15;++j)
		for(int i=0;i<n;++i)
			sp[j][i]=sp[j-1][sp[j-1][i]];
}
int getPth(int u,int p){
	for(int i=15;i>=0;--i)
		if(p&(1<<i))
			u=sp[i][u];
	re u;
}
int lca(int u,int v){
	if(lev[u]<lev[v])
		swap(u,v);
	u=getPth(u,lev[u]-lev[v]);
	if(u==v)re u;
	for(int i=15;i>=0;--i)
		if(sp[i][u]!=sp[i][v])
			u=sp[i][u],v=sp[i][v];
	re sp[0][u];
}
ll dis(int u,int v){
	int LCA=lca(u,v);
	re cost[u]+cost[v]-(2ll*cost[LCA]);
```

```
}
int getKth(int u,int v,int k){
	int LCA=lca(u,v);
	int Udis=lev[u]-lev[LCA];
	int Vdis=lev[v]-lev[LCA];
	int Alldis=Udis+Vdis;
	if(k<=Udis)re getPth(u,k);
	else re getPth(v,Alldis-k);
}
int main()
{
	IO;
	cin >> t;
	while(t--){
		cin >> n;
		init();
		int u,v,k,c;
		for(int i=1;i<n;++i){
			cin >> u >> v >> c;
			add_bi_edge(--u,--v,c);
		}
		id=-1;
		dfs(0,-1,0,0);
		buildSparse();
		while(true){
			cin >> s;
			if(s=="DONE")break;
			else if(s=="DIST"){
				cin >> u >> v;
				cout<<dis(--u,--v)<<"\n";
			}
			else{
				cin >> u >> v >> k;
				cout<<getKth(--u,--v,--k)+1<<"\n";
			}
		}
		cout<<"\n";
	}
	re 0;
}
```

Path Cover

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

using namespace std;
const int N = 1e3 + 3, M = 1e3 + 3, E = 6e5 + 3;
int head[N], work[N], to[E], nxt[E], ne;
int n, m;

void init() {
	memset(head, -1, n * sizeof(head[0]));
	ne = 0;
}
```

```
void add_edge(int f, int t) {
	to[ne] = t;
	nxt[ne] = head[f];
	head[f] = ne++;
}

int rID[N], lID[M], dist[N];

bool dfs(int lf) {
	for (int &e = work[lf]; ~e; e = nxt[e]) {
		int rt = to[e];
		int nwLf = lID[rt];
		if (nwLf == -1 || dist[nwLf] == dist[lf] + 1 && dfs(nwLf)) {
			rID[lf] = rt;
			lID[rt] = lf;
			return true;
		}
	}
	return false;
}

int q[N], qsz;

bool bfs() {
	qsz = 0;
	for (int i = 0; i < n; ++i)
		if (rID[i] == -1)
			q[qsz++] = i, dist[i] = 0;
		else
			dist[i] = 1e9;
	bool kammel = true;
	int frnt = 0;
	while (frnt < qsz && kammel) {
		int s = qsz;
		while (frnt < s) {
			int lf = q[frnt++];
			for (int e = head[lf]; ~e; e = nxt[e]) {
				int rt = to[e];
				int nwLf = lID[rt];
				if (nwLf == -1) {
					kammel = false;
				} else if (dist[nwLf] > dist[lf] + 1) {
					dist[nwLf] = dist[lf] + 1;
					q[qsz++] = nwLf;
				}
			}
		}
	}
	return !kammel;
}

int match() {
	memset(rID, -1, n * sizeof(rID[0]));
	memset(lID, -1, m * sizeof(lID[0]));
	int res = 0;
```

```
	while (bfs()) {
		memcpy(work, head, n * sizeof head[0]);
		for (int i = 0; i < n; ++i)
			if (rID[i] == -1 && dfs(i))
				++res;
	}
	return res;
}

char s[1001];
int main() {
#ifndef ONLINE_JUDGE
	freopen("in.txt", "r", stdin);
//	freopen("stall4.in", "r", stdin);
//	freopen("stall4.out", "w", stdout);
#endif
	int test = 1;
	while (scanf("%s", s) && s[0] != 'e') {
		n = strlen(s);
		init();
		m = n;
		for (int i = 0; i < n; ++i)
			for (int j = i + 1; j < m; ++j)
				if (s[j] <= s[i])
					add_edge(i, j);
		printf("Case %d: %d\n", test++, n - match());
	}
	return 0;
}
```

SCC Tarjan

```
#include <bits/stdc++.h>
using namespace std;
vector<vector<int> > SCCs;
#define comps SCCs
vector<int> compIndex,ind, lowLink;
stack<int> st;
vector<bool> inst;
vector<vector<int> > adj;
int idx = 0;
void tarjanSCC(int i){
    lowLink[i] = ind[i] = idx++;
    st.push(i);
    inst[i] = true;
    for(int j = 0; j < adj[i].size(); j++){
        int k = adj[i][j];
        if(ind[k] == -1) {
            tarjanSCC(k);
            lowLink[i] = min(lowLink[i], lowLink[k]);
        }else if(inst[k]){
            lowLink[i] = min(lowLink[i], lowLink[k]);
        }
```

```
    }
    if(lowLink[i] == ind[i]) {
    vector<int> comp;
    int n = -1;
    while(n != i){
        n = st.top();
        st.pop();
        comp.push_back(n);
        inst[n] = 0;
        compIndex[n] = comps.size();
    }
    comps.push_back(comp);
    }
}
void SCC(){
    comps.clear();
    compIndex.resize(adj.size());
    ind.clear();
    ind.resize(adj.size(), -1);
    lowLink.resize(adj.size());
    inst.resize(adj.size());
    idx = 0;
    for(int i = 0; i <adj.size(); i++)
        if(ind[i] == -1)
            tarjanSCC(i);
}
int cntSrc , cntSnk;
vector<vector<int> > cmpAdj;
vector<int> inDeg, outDeg;
void computeNewGraph(){
    outDeg.clear();
    outDeg.resize(comps.size());
    inDeg.clear();
    inDeg.resize(comps.size());
    cntSrc = cntSnk = comps.size();
    cmpAdj.clear();
    cmpAdj.resize(comps.size());
    for(int i = 0; i <adj.size(); i++) {
        for(int j = 0; j < adj[i].size(); j++) {
            int k = adj[i][j];
            if(compIndex[k] != compIndex[i]) {
                cmpAdj[compIndex[i]].push_back(compIndex[k]);
                if(!(inDeg[compIndex[k]]++))
                    cntSrc--;

                if(!(outDeg[compIndex[i]]++))
                    cntSnk--;
            }
        }
    }
}
```

Tarjan (SCC) 2

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

```cpp
using namespace std;
const int N = 1e5 + 5; // number of nodes
int vis[N] , low[N] , dfsTime[N] , comp_id[N] , Time;
vector<int> adj[N];
stack<int> st;
vector<vector<int> > comps;
int dfs(int node) { // add par if undirected
    if(vis[node])
        return vis[node] == 1 ? low[node] : 1e9;

    dfsTime[node] = low[node] = Time++;
    vis[node] = 1;
    st.push(node);

    for (int child : adj[node]) // in case of undirected continue if par
        low[node] = min(low[node] , dfs(child));

    if(low[node] == dfsTime[node]) {
        comps.push_back(vector<int>());
        do {
            vis[st.top()] = 2;
            comp_id[st.top()] = comps.size() - 1;
            comps.back().push_back(st.top());
            st.pop();
        } while(comps.back().back() != node);
    }

    return low[node];
}
vector<int> tree[N]; // tree or dag
void tarjan(int n) {
    for (int i = 1 ;i <= n ;i++) // one based
        if(!vis[i])
            dfs(i);

    for (int i = 1 ;i <= n ;i++) // one based
        for (int child : adj[i])
            if (comp_id[i] != comp_id[child])
                tree[comp_id[i]].push_back(comp_id[child]);
}
void init(int n) {
    fill(adj , adj + (n + 1) , vector<int>()); // n + 1 for one based
    fill(tree , tree + (n + 1) , vector<int>());
    memset(vis , 0 , (n + 1) * sizeof vis[0]);
    comps.clear();
}
```

Tarjan (Bridges)

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int n; // number of nodes
vector<int> adj[N]; // adjacency list of graph
```

```
bool vis[N];
int tin[N], low[N];
int timer;
vector<pair<int,int> > bridges
void add_BRIDGE(int v, int to) {
    bridges.push_back({v,to});
    return;
}
void dfs(int v, int p = -1) {
    vis[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (vis[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                add_BRIDGE(v, to);
        }
    }
}
void find_bridges() {
    timer = 0;
    memset(vis, 0, sizeof vis);
    for (int i = 0; i < n; ++i) {
        if (!vis[i])
            dfs(i);
    }
}
```

Tarjan (Articulation Points)

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int n; // number of nodes
vector<int> adj[N]; // adjacency list of graph
bool vis[N];
int tin[N],low[N];
int timer;
void IS_CUTPOINT(int v){return;}
void dfs(int v, int p = -1) {
    vis[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (vis[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
```

```
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}
void find_cutpoints() {
    timer = 0;
    memset(vis,0,sizeof vis);
    for (int i = 0; i < n; ++i) {
        if (!vis[i])
            dfs (i);
    }
}
```

Max Matching

```
const int MX = 405;
vector<int> adj[MX];
int r[MX], l[MX], vis[MX];
int vis_id, numR;
bool match(int u) {
    if (vis[u] == vis_id)return false;
    vis[u] = vis_id;
    for (int nxt : adj[u]) {
        if (l[nxt] == -1 || match(l[nxt])) {
            l[nxt] = u;
            r[u] = nxt;
            return true;
        }
    }
    return false;
}
int runMatching() {
    int cc = 0;
    memset(r, -1, sizeof r);
    memset(l, -1, sizeof l);
    for (int i = 0; i < numR; i++) {
        vis_id++;
        if (match(i))
            cc++;
    }
    return cc;
}
```

Max Flow

```
const int N  = 5e3 + 3;
const int M = 6e4 + 4;
int n,m;
int head[N], work[N], nxt[M], to[M], cap[M], dis[N], que[N];
int edge_cnt , src, snk , back , front , qsz;
```

```
//src and snk is different according to the problem
void add_edge(int f, int t, int c) {
  nxt[edge_cnt] = head[f];
  head[f] = edge_cnt;
  to[edge_cnt] = t;
  cap[edge_cnt] = c;
  edge_cnt++;
}
void add_bi_edge(int f, int t, int c) {
  add_edge(f, t, c);
  add_edge(t, f, c); // 0 if directed edge
}
void init() {
    memset(head, -1, sizeof head);
    edge_cnt = 0;
}
bool bfs() {
    back = front = qsz = 0;
    memset(dis, -1, sizeof dis);
    dis[src] = 0;
    que[qsz++, back++] = src;
    while (qsz) {
        int cur = que[qsz--,front++];
        for (int i = head[cur]; ~i; i = nxt[i]){
                int t = to[i];
                if (!cap[i] || dis[t] != -1)continue;
                dis[t] = dis[cur] + 1;
                if (t == snk)return 1;
                que[qsz++, back++] = t;
        }
    }
    return 0;
}
int dfs(int u, int flow = 1e9 + 3) {// make sure that no edge has cap
greater than 1e9
    if(!flow)return flow;
    if (u == snk)return flow;
    for (int &i = work[u]; i != -1; i = nxt[i]){
        int v = to[i];
        if (!cap[i] || dis[u] + 1 != dis[v])continue;
        int f = dfs(v, min(flow, cap[i]));
        if (f){
          cap[i] -= f;
          cap[i ^ 1] += f;
          return f;
        }
    }
    return 0;
}
long long  max_flow(){
    long long ret = 0, flow = 0;
    while (bfs()) {
        memcpy(work, head, sizeof head);
        while (flow = dfs(src))ret += flow;
  }
```

```
  return ret;
}
```

Max Flow Min Cost

```
const int MAXN = 102;
const int MAXE = 5202;
int head[MAXN], nxt[MAXE], to[MAXE], from[MAXE], edgeCnt;
int cap[MAXE], cost[MAXE];
int vis[MAXN], visID = 1, src, snk;
int parE[MAXN];
int flow[MAXN], dist[MAXN];
int que[MAXN], front, back, qSiz;
int n;
inline void addEdge(int f, int t, int cst, int flw) {
  nxt[edgeCnt] = head[f];
  head[f] = edgeCnt;
  to[edgeCnt] = t;
  from[edgeCnt] = f;
  cost[edgeCnt] = cst;
  cap[edgeCnt] = flw;
  edgeCnt++;
}
inline void addBiEdge(int f, int t, int cst, int flw) {
  addEdge(f, t, cst, flw);
  addEdge(t, f, -cst, 0);
}
inline void init() {
  memset(head, -1,sizeof head);
  edgeCnt = 0;
}
inline int bellman() {
  memset(dist, 0x3f, n * sizeof dist[0]);
  flow[snk] = 0;
  flow[src] = 1e9;
  dist[src] = 0;
  front = back = qSiz = 0;
  que[qSiz++, ++back] = src;
  vis[src] = ++visID;
  int m = n;
  while (qSiz && m--) {
    int s = qSiz;
    while (s--) {
      int cur = que[qSiz--, front = ++front % MAXN];
      vis[cur] = 0;
      for (int i = head[cur]; i != -1; i = nxt[i]) {
        if (cap[i] == 0)
          continue;
        int cst = dist[cur] + cost[i];
        int t = to[i];
        if (dist[t] > cst) {
          dist[t] = cst;
          parE[t] = i;
          flow[t] = min(flow[cur], cap[i]);
          if (vis[t] != visID) {
```

```
            vis[t] = visID;
            que[qSiz++, back = ++back % MAXN] = t;
          }
        }
      }
    }
  }
  return flow[snk];
}
inline void minCostMaxFlow(int &cst, int &flw) {
  cst = 0, flw = 0;
  while (bellman()) {
    for (int i = snk, p; i != src; i = from[p]) {
      p = parE[i];
      cap[p] -= flow[snk];
      cap[p ^ 1] += flow[snk];
    }
    flw += flow[snk];
    cst += flow[snk] * dist[snk];
  }
}
```

2 SAT

```
struct _2Sat {

    vector <vector<int>> g, rg;
    vector<bool> vis;
    vector<int> cmp, tr;  ///true variables
    int nodes;

    _2Sat(int n = 0) {
        nodes = n;
        g.resize(n + n + 5);
        rg.resize(n + n + 5);
        vis.assign(n + n + 5, 0);
        cmp.assign(n + n + 5, 0);
    }

    void addEdge(int u, int v) {
        g[u].pb(v);
        rg[v].pb(u);
    }

    void addOR(int u, int v) {
        ///this means either U or V is true
        addEdge(u ^ 1, v);
        addEdge(v ^ 1, u);
    }

    void dfs1(int u) {
        if (vis[u])
            return;
        vis[u] = true;
```

```
        for (auto v : g[u])
            dfs1(v);
        stk.push(u);
    }

    void dfs2(int u, int cnt) {
        if (cmp[u])
            return;
        cmp[u] = cnt;
        for (auto v : rg[u])
            dfs2(v, cnt);
    }

    bool killSat() {
        for (int i = 0; i < nodes + nodes; i++)
            dfs1(i);
        int cnt = 0;
        while (stk.size()) {
            auto x = stk.top();
            stk.pop();
            if (!cmp[x])
                dfs2(x, ++cnt);
        }
        for (int i = 0; i < nodes; i++)
            if (cmp[i << 1] == cmp[i << 1 | 1])
                return false;
            else if (cmp[i << 1] > cmp[i << 1 | 1])
                tr.pb(i);
        return true;
    }
};
```

Trees

DSU On Trees

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

const int N = 1e5 + 5;
int sz[N] , in[N];
vector<int> adj[N] , e;

int dfs(int u , int p) { // call first as a pre process
    sz[u] = 1;
    in[u] = e.size();
    e.push_back(u);
    for (int v : adj[u])
        if (v != p)
            sz[u] += dfs(v , u);
    return sz[u];
}

void update (int node) {
}
```

```
void dsu(int u , int p , bool keep) { //
    int mx = -1 , bigChild = -1;
    for (int v : adj[u])
        if (v != p && sz[v] > mx)
            mx = sz[v] , bigChild = v;

    for (int v : adj[u])
        if (v != p && v != bigChild)
            dsu(v , u , 0); // update and do not keep

    if (bigChild != -1)
        dsu(bigChild , u , 1); // update the big child and keep it

    for (int v : adj[u])
        if (v != p && v != bigChild)
            for (int i = in[v] ;i < in[v] + sz[v] ;i++)
                update(e[i]);

    update(u); // update the current node

    // answer queries for the current node

    if (!keep)
        for (int i = in[u] ;i < in[u] + sz[u] ;i++)
            update(e[i]);
}
```

LCA O(1)

```
#include <bits/stdc++.h>
//#include <ext/pb_ds/assoc_container.hpp>
//#include <ext/pb_ds/tree_policy.hpp>
//using namespace __gnu_pbds;
using namespace std;
//template <typename T>
//using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
typedef vector<int> vi;
typedef long long ll;
#define pb push_back
#define inf 0x3f3f3f3f
#define all(v) (v).begin() , (v).end()
#define ones(n) __builtin_popcount(n)
#define watch(x) cout << (#x) << " is " << (x) << endl
int di[] = {0, 0, 1, -1, 1, 1, -1, -1};
int dj[] = {1, -1, 0, 0, -1, 1, 1, -1};
/* blackBox
   update N and M acording to the problem
   adj is the undirected tree
   don't forget to clear it every test case
*/
const int N = 1e5 + 5, M = 18; // update according to the problem
int n;
```

```
vi adj[N];
int occ[N], E[2 * N], L[2 * N], table[M][2 * N], LG[2 * N], sz = 0;
void dfs(int node = 1, int par = -1, int level = 0) {
    table[0][sz] = sz;
    occ[node] = sz;
    E[sz] = node;
    L[sz++] = level;
    for (int child : adj[node]) {
        if (child == par) continue;
        dfs(child, node, level + 1);
        E[sz] = node;
        table[0][sz] = sz;
        L[sz++] = level;
    }
}
// build the sparse table to get the position of the minmum value
void build() {
    sz = 0;
    dfs();

    for (int j = 1; j <= M; j++) {
        for (int i = 0; i + (1 << j) - 1 < sz; i++) {
            int f = table[j - 1][i];
            int s = table[j - 1][i + (1 << (j - 1))];
            table[j][i] = L[f] < L[s] ? f : s;
        }
    }
}
int RMQ(int l, int r) {
    if (r < l)
        swap(l, r);
    int lg = LG[r - l + 1];
    int f = table[lg][l];
    int s = table[lg][r - (1 << lg) + 1];
    return L[f] < L[s] ? E[f] : E[s];
}
int LCA(int a, int b) {
    return RMQ(occ[a], occ[b]);
}
void init() {
    LG[0] = -1;
    for (int i = 1; i < 2 * N; i++)
        LG[i] = LG[i - 1] + !(i - (i & (-i)));
}
int main() {
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
#endif
    init(); // only once through all test cases
    build(); // every test case
}
```

LCA 2

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

```cpp
using namespace std;


const int N = 100010;
const int LG = 30;
vector <int> adj[N];
int depth[N], dp[N], anc[N][LG];
int tin[N], tout[N], timer;
int n;

void dfs(int u, int par = 0, int dep = 0) {
    depth[u] = dep;
    dp[u] = 1;
    tin[u] = timer++;
    anc[u][0] = par;

    for (int i = 1; i < LG; i++){
        anc[u][i] = anc[anc[u][i - 1]][i - 1];
    }

    for (int i = 0; i < adj[u].size(); i++){
        int to = adj[u][i];
        if (to != par) {
            dfs(to, u, dep + 1);
            dp[u] += dp[to];
        }
    }

    tout[u] = timer++;
}

bool ancestor(int a, int b){
    return tin[a] <= tin[b] && tout[b] <= tout[a];
}

int go_up(int a, int b) {
    for (int i = LG - 1; i >= 0; i--) {
        if (!ancestor(anc[a][i], b)){
            a = anc[a][i];
        }
    }
    return a;
}

int lca(int a, int b) {
    int result = -1;
    if (ancestor(a, b)){
        result = a;
    } else if (ancestor(b, a)){
        result = b;
    } else {
        result = anc[go_up(a, b)][0];
    }

    return result;
```

```
}

int main(){

    return 0;
}


--------------------------------------------------------------------------------
--------------
const int N = 10001;
vector<int> adj[N];
int lvl[N], anc[N][25];
int n;

void dfs(int u = 0 , int par = 0 , int l = 0){
    lvl[u] = l;
    anc[u][0] = par;
    for(auto nxt : adj[u])
       if(nxt != par)
            dfs(nxt , u , l+1);
}

void buildLCA() {
    int lg = ceil(log2(n));
    for(int j = 1 ; j < lg ; j++)
        for(int i = 0 ; i < n ; i++)
            anc[i][j] = anc[anc[i][j - 1]][j - 1];
}

int LCA(int i, int j){
    int lg = ceil(log2(n));
    int st = lg;
    if (lvl[i] > lvl[j])swap(i, j);
    int cur = lvl[j];
    for (; st >= 0; st--)
        if (cur - (1 << st) >= lvl[i])
            cur -= (1 << st), j = anc[j][st];
    if (i == j)return 2 * i - j;
    cur = lvl[i];
    for (st = lg; st >= 0; st--)
        if (anc[i][st] != anc[j][st])
            cur -= (1 << st), i = anc[i][st], j = anc[j][st];
    return anc[i][0];
}
```

Tree Diameter

```
// the first of the pair is the number of nodes on the diemeter
pair<int, int> get_diameter(int node, int par = -1) {
    int d = 0, mx[3] = {};
    for (auto child : adj2[node]) {
        if (child == par) continue;
        auto p = get_diameter(child, node);
```

```
        mx[0] = p.second;
        sort(mx, mx + 3);
        d = max(d, p.first);
    }
    return {max(d, mx[2] + mx[1] + 1), mx[2] + 1};
}
```

Diameter And Tree Centers

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1005; // number of nodes in the tree
vector<int> adj[N]; // adj must be undirected tree
int Deg[N];
pair<int , vector<int>> getDimeter(vector<int> &nodes) { // all nodes in one
connected component
    vector<int> centers;
    queue<int> leafs;
    for (int x : nodes) {
        Deg[x] = adj[x].size();
        if(Deg[x] == 1) // because undirected
            leafs.push(x);
    }

    if (nodes.size() <= 2)
        return {nodes.size() - 1 , nodes};

    int d = 0;
    int cnt = nodes.size();
    while(cnt > 2) {
        int sz = leafs.size();
        d += 2;
        cnt -= sz;
        while(sz--) {
            int cur = leafs.front();
            leafs.pop();
            for (int par : adj[cur]) {
                Deg[par]--;
                if(Deg[par] == 1)
                    leafs.push(par);
            }
        }
    }


    while(leafs.size())
        centers.push_back(leafs.front()) , leafs.pop();

    return {d + (centers.size() == 2) , centers};  // returns dimeter and
centers
}
int main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
#endif
```

```
}
```

Centroid

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100005;
struct edge {
    int u, v;
    long long cost;

    bool operator<(const edge &other) const {
        return cost > other.cost;
    }
};
vector<edge> edges;
vector<int> AdjList[MAXN];
bool cut[MAXN];
int subTreeSize[MAXN];

int dfs(int u, int par) {
    subTreeSize[u] = 1;
    for (int e : AdjList[u]) {
        int v = edges[e].u + edges[e].v - u;
        if (v != par && !cut[v])
            subTreeSize[u] += dfs(v, u);
    }
    return subTreeSize[u];
}
int pick(int u, int par, int size) {
    int mx = 0;
    for (int e : AdjList[u]) {
        int v = edges[e].u + edges[e].v - u;
        if (v != par && !cut[v])
            mx = max(mx, subTreeSize[v]);
    }
    if (mx <= size / 2)
        return u;
    for (int e : AdjList[u]) {
        int v = edges[e].u + edges[e].v - u;
        if (!cut[v] && v != par && subTreeSize[v] == mx) {
            return pick(v, u, size);
        }
    }

}
int getCenter(int u) {
    return pick(u, -1, dfs(u, -1));
}
// solve
void centroid(int u) {
// calculate center
    int center = getCenter(u);
```

```
// solve
    solve(center);

// recurse
    cut[center] = 1;
    for (int e : AdjList[center]) {
        int v = edges[e].u + edges[e].v - center;
        if (!cut[v])
            centroid(v);
    }
}
void pre(int n) {
    // don't miss pre of solve
    edges.clear();
    for (int i = 0; i <= n; ++i) {
        AdjList[i].clear();
        cut[i] = 0;
    }
}
```

Heavy Light Decomposition

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

using namespace std;
const int N=1e5+5;
vector<int>adj[N];
int head[N],pos[N],parent[N],heavy[N],depth[N];
int cur_pos;
int dfs(int u) {
    int sz = 1;
    int mx = 0;
    for (int v : adj[u]) {
        if (v != parent[u]) {
            parent[v] = u, depth[v] = depth[u] + 1;
            int c = dfs(v);
            sz += c;
            if (c > mx)
                mx = c, heavy[u] = v;
        }
    }
    return sz;
}
void decompose(int u, int h) {
    head[u] = h;
    pos[u] = cur_pos++;
    if (heavy[u] != -1)
        decompose(heavy[u], h);
    for (int v : adj[u]) {
        if (v != parent[u] && v != heavy[u])
            decompose(v, v);
    }
}

int query(int a, int b) {
```

```
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        //int cur_heavy_path_max = segment_tree_query(pos[head[b]], pos[b]);
        //res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
   // int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
    //res = max(res, last_heavy_path_max);
    return res;
}
void init()
{
    cur_pos=0;
    // zbt el root fel depth we parent
    memset(heavy,-1,sizeof heavy);

}
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    memset(heavy,-1,sizeof heavy);
    int n,q;
    cin>>n>>q;
    for(int i=1;i<n;i++)
    {
        int u,v;
        cin>>u>>v;
        u--;v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    return 0;
}
```

Maths

Extended GCD

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

Modular Power

```
long long power(long long a , long long b) {
    if (!b) return 1;
    long long r = power(a , b / 2);
    r = r * r % MOD;
    if (b&1) return r * a % MOD;
    return r;
}
```

Modular Inverse

```
inv[1] = 1;
for(int i = 2; i < m; ++i)
    inv[i] = (m - (m/i) * inv[m%i] % m) % m;
const long long MOD = 1e9 + 7; // have to be a prime
long long mul(long long a , long long b) {
    return a * b % MOD;
}
long long power(long long a , long long b) {
    if (!b) return 1;
    long long r = power(a , b / 2);
    r = mul(r , r);
    if (b&1) return mul(r , a);
    return r;
}
long long mod_inv(long long x) {
    return power(x , MOD - 2);
}
```

nCr (Small, Dp)

```
long long nCr(int n, int r) {
    long long ret = 1;
    r = max(r , n - r);
    for (int i = 1; i <= n - r; ++i) {
        ret *= r + i;
        ret /= i;
    }
    return ret;
}

ll C[N][N];
ll nCr(int n ,int r){
    if(n == r  r == 0)
        return 1;
    ll &ret = C[n][r];
    if(ret != -1)
        return ret;
    return ret = nCr(n-1 , r) + nCr(n-1 , r-1);
}
```

nCr Mod

```
#include <bits/stdc++.h>
using namespace std;
const long long MOD = 1e9 + 7;
const int N = 1e5+5;
long long f[N];
long long power(long long a , long long b){
    long long x = 1 , y = a;
    while(b > 0){
        if(b & 1){
            x = (x * y);
            if(x > MOD)
                x %= MOD;
        }

        y = y*y;
        if(y > MOD)
            y %= MOD;

        b >>= 1;
    }
    return x;
}
long long nCr(long long n , long long r){
    return (f[n] * power(f[r] * f[n-r] %MOD , MOD - 2))% MOD;
}
int main(){
    f[0] = 1;
    for(int i = 1 ; i < N ;i++)
        f[i] = (f[i-1] * i) % MOD;


    int t;
    cin >> t;
    while(t--){
        int n , m;
        cin >> n >> m;
        cout << nCr(n+m , n) << endl;
    }
}
```

Miller Rabin Primality Test

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
// n < 4,759,123,141                            3 :  2, 7, 61
// n < 1,122,004,669,633                   4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383                   6 :  pirmes <= 13
// n < 3,825,123,056,546,413,051 9 :  primes <= 23
int testPrimes[] = {2,3,5,7,11,13,17,19,23};
struct MillerRabin{
  ///change K according to n
  const int K = 9;
```

```
  ll mult(ll s, ll m, ll mod){
    if(!m) return 0;
    ll ret = mult(s, m/2, mod);
    ret = (ret + ret) % mod;
    if(m & 1) ret = (ret + s) % mod;
    return ret;
  }
  ll power(ll x, ll p, ll mod){
    ll s = 1, m = x;
    while(p){
      if(p&1) s = mult(s, m, mod);
      p >>= 1;
      m = mult(m, m, mod);
    }
    return s;
  }
  bool witness(ll a,ll n,ll u,int t){
    ll x = power(a, u, n), nx;
    for(int i = 0; i < t; i++){
      nx = mult(x, x, n);
      if(nx == 1 and x != 1 and x != n-1) return 1;
      x = nx;
    }
    return x != 1;
  }
  bool isPrime(ll n){     // return 1 if prime, 0 otherwise
    if(n < 2) return 0;
    if(!(n&1)) return n == 2;
    for(int i = 0; i < K; i++)if(n == testPrimes[i])return 1;
    ll u = n-1; int t = 0;

    while(u&1) u >>= 1, t++;     // n-1 = u*2^t

    for(int i = 0; i < K; i++) if(witness(testPrimes[i], n, u, t)) return 0;
    return 1;
  }
}tester;
```

Matrix Power

```
typedef vector< long long > vl;
typedef vector<vl> matrix;
long long  md=1000000007 ;
matrix initial(int n,int m){
return matrix(n,vl(m,0));
}

matrix identity(int n){
     matrix ret= initial(n,n);
     for(int i=0;i<n;i++)ret[i][i]=1;
     return ret;

}

matrix multiply(const matrix &x,const matrix &y){
```

```
    matrix ret=initial(x.size(),y[0].size());
    for(int i=0;i<x.size();i++){
        for(int k=0;k<x[0].size();k++){
            for(int j=0;j<y[0].size();j++){
                ret[i][j]+=(((x[i][k]*y[k][j])%md)+md)%md;
                ret[i][j]=((ret[i][j]%md)+md)%md;
            }
        }
    }
    return ret;
}

matrix pow(const matrix &x,long long k){
    if(k==0)return identity(x.size());
    if(k&1)return multiply(x,pow(x,k-1));

    return pow(multiply(x,x),k/2);
}

long long fib(int n){
    if(n<3)return 1;
    matrix init=initial(1,2),t=initial(2,2);
    t[0][1]=t[1][1]=t[1][0]=1;
    init[0][0]=init[0][1]=1;
    matrix moves=pow(t,n-2);
    matrix ans=multiply(init,moves);
    return ans[0][1];

}
```

Matrix Power (Short)

```
using row = vector<long long>;
using mat = vector<row> ;

const int MOD = 1e9 + 7;

mat operator * (mat a , mat b) {
    mat ret = mat(a.size() , row(b[0].size()));
    for (int i = 0 ;i < ret.size() ;i++)
        for (int j = 0 ;j < ret[i].size() ;j++)
            for (int k = 0 ;k < ret[i].size() ;k++)
                ret[i][j] = (ret[i][j] + a[i][k] * b[k][j]) % MOD;
    return ret;
}

mat operator ^ (mat b , int p) {
    if (p == 1) return b;
    if (p&1) return b * (b ^ (p - 1));
    return (b * b) ^ (p / 2);
}
```

Sieve

```
const int N = 1e7;
bool prime[N];
void sieve() {
        memset(prime , 1 , sizeof prime);
        prime[0] = prime[1] = 0;
        for (int i = 2 ;i * i < N ;i++)
            if (prime[i])
                for (int j = i * i ;j < N ;j += i)
                    prime[j] = 0;
}

// get unique prime factors of numbers from 1 - N
const int N = 2e5 + 5;
vector<int> pf[N];
void sieve() {
    for (int i = 2 ;i < N ;i++)
        if (!pf[i].size())
            for (int j = i ;j < N ;j += i)
                pf[j].push_back(i);
}
```

Phi

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            while(n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if(n > 1)
        result -= result / n;
    return result;
}
```

Quadratic Formula

```
pair<long long , long long> Quadratic_Formula(long long a, long long b, long
long c) {
    long long sqt = sqrt(b * b - 4 * a * c);
    return {(-b + sqt) / (2 * a), (-b - sqt) / (2 * a)};
}
```

Nth Root

```
bool overflow(long long a, long long b) {
    long double res = a * b;
    if (a == 0 || b == 0 || res / b == a)
        return false;
    return true;
```

```
}
long long nth_root(long long x , int n) {
    long long l = 1 , r = x , theOne = -1;
    while (l <= r) {
        long long mid = (l + r) / 2 , ans = 1;
        bool ovf = 0;
        for (int i = 0 ;i < n ;i++) {
            ovf |= overflow(ans , mid);
            ans *= mid;
        }

        if (ans == x) {
            theOne = mid;
            break;
        }

        if (ovf || ans >= x)
            r = mid - 1;
        else
            l = mid + 1;
    }

    return theOne;
}
```

Kadane 1D

```
// C++ program to print largest contiguous array sum
#include<iostream>
#include<climits>
using namespace std;

int maxSubArraySum(int a[], int size)
{
    int max_so_far = INT_MIN, max_ending_here = 0;

    for (int i = 0; i < size; i++)
    {
        max_ending_here = max_ending_here + a[i];
        if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;

        if (max_ending_here < 0)
            max_ending_here = 0;
    }
    return max_so_far;
}

/*Driver program to test maxSubArraySum*/
int main()
{
    int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};
    int n = sizeof(a)/sizeof(a[0]);
    int max_sum = maxSubArraySum(a, n);
    cout << "Maximum contiguous sum is " << max_sum;
```

```
    return 0;
}
```

Kadane 2D

```
// Program to find maximum sum subarray
// in a given 2D array
#include<bits/stdc++.h>
using namespace std;

#define ROW 4
#define COL 5

// Implementation of Kadane's algorithm for
// 1D array. The function returns the maximum
// sum and stores starting and ending indexes
// of the maximum sum subarray at addresses
// pointed by start and finish pointers
// respectively.
int kadane(int* arr, int* start,
           int* finish, int n)
{
    // initialize sum, maxSum and
    int sum = 0, maxSum = INT_MIN, i;

    // Just some initial value to check
    // for all negative values case
    *finish = -1;

    // local variable
    int local_start = 0;

    for (i = 0; i < n; ++i)
    {
        sum += arr[i];
        if (sum < 0)
        {
            sum = 0;
            local_start = i + 1;
        }
        else if (sum > maxSum)
        {
            maxSum = sum;
            *start = local_start;
            *finish = i;
        }
    }

    // There is at-least one
    // non-negative number
    if (*finish != -1)
        return maxSum;

    // Special Case: When all numbers
```

```
    // in arr[] are negative
    maxSum = arr[0];
    *start = *finish = 0;

    // Find the maximum element in array
    for (i = 1; i < n; i++)
    {
        if (arr[i] > maxSum)
        {
            maxSum = arr[i];
            *start = *finish = i;
        }
    }
    return maxSum;
}

// The main function that finds
// maximum sum rectangle in M[][]
void findMaxSum(int M[][COL])
{
    // Variables to store the final output
    int maxSum = INT_MIN, finalLeft, finalRight,
                          finalTop, finalBottom;

    int left, right, i;
    int temp[ROW], sum, start, finish;

    // Set the left column
    for (left = 0; left < COL; ++left)
    {
        // Initialize all elements of temp as 0
        memset(temp, 0, sizeof(temp));

        // Set the right column for the left
        // column set by outer loop
        for (right = left; right < COL; ++right)
        {

            // Calculate sum between current left
            // and right for every row 'i'
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

            // Find the maximum sum subarray in temp[].
            // The kadane() function also sets values
            // of start and finish. So 'sum' is sum of
            // rectangle between (start, left) and
            // (finish, right) which is the maximum sum
            // with boundary columns strictly as left
            // and right.
            sum = kadane(temp, &start, &finish, ROW);

            // Compare sum with maximum sum so far.
            // If sum is more, then update maxSum and
            // other output values
```

```
            if (sum > maxSum)
            {
                maxSum = sum;
                finalLeft = left;
                finalRight = right;
                finalTop = start;
                finalBottom = finish;
            }
        }
    }

    // Print final values
    cout << "(Top, Left) (" << finalTop
         << ", " << finalLeft << ")" << endl;
    cout << "(Bottom, Right) (" << finalBottom
         << ", " << finalRight << ")" << endl;
    cout << "Max sum is: " << maxSum << endl;
}

// Driver Code
int main()
{
    int M[ROW][COL] = {{1, 2, -1, -4, -20},
                       {-8, -3, 4, 2, 1},
                       {3, 8, 10, 1, 3},
                       {-4, -1, 1, 7, -6}};

    findMaxSum(M);

    return 0;
}
```

FFT

```
using cd = complex<double>;
const double PI = acos(-1);
int reverse(int num, int lg_n) {
    int res = 0;
    for (int i = 0; i < lg_n; i++) {
        if (num & (1 << i))
            res |= 1 << (lg_n - 1 - i);
    }
    return res;
}
void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    int lg_n = 0;
    while ((1 << lg_n) < n)
        lg_n++;

    for (int i = 0; i < n; i++) {
        if (i < reverse(i, lg_n))
            swap(a[i], a[reverse(i, lg_n)]);
    }
```

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

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

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

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

EGCD - LDE - Phi

```
typedef long long ll;
const int N = 2e5+5, M = 1e6+6, OO = 0x3f3f3f3f;
//bool is_prime[N];
bitset<N> is_prime;
void sieve(){
      is_prime.set();
      is_prime[0] = is_prime[1] = 0;
      for(ll p = 2 ; p <= N/p ; ++p)
            if(is_prime[p])
                  for(ll m = p*p ; m < N ; m += p)
                        is_prime[m] = 0;
}
```

```
int gcd(int a, int b){
	if(b)   return gcd(b, a%b);
	else return a;
}

void move1step(int& a, int& b, const int& q){
	int c = a - q*b;
	a = b;
	b = c;
}

int gcdI(int a, int b){
	while(b)    move1step(a, b, a/b);
	return a;
}

int eGCD(int r0, int r1, int& x0, int& y0){
	int x1 = y0 = 0, y1 = x0 = 1;
	while(r1){
		int q = r0/r1;
		move1step(r0, r1, q);
		move1step(x0, x1, q);
		move1step(y0, y1, q);
	}
	return r0;
}

bool solveLDE(int a, int b, int c, int& x, int& y, int& g){
	g = eGCD(a, b, x, y);
	int m = c/g;
	x *= m;
	y *= m;
	return (c%g == 0);
}


ll phi[N];
void phiSieve(){
  iota(phi, phi+N, 0);
  for(ll i = 2 ; i < N ; i += 1 + (i&1))
	if(phi[i] == i)
	  for(ll j = i ; j < N ; j+=i)
		phi[j] -= phi[j]/i;
}
```

Solving Inequalities

```
const int maxn=200005;
const int N=128,M=205,N1=N-1;
int n,m,cost[M],dis[N],ne;
int head[N],nxt[M],to[M];
void init(){
	ne=0;
	memset(head,-1,n*sizeof head[0]);
}
```

```
void add_edge(int f,int t,int c){
	to[ne]=t;
	nxt[ne]=head[f];
	cost[ne]=c;
	head[f]=ne++;
}
int inQ[N],vid;
int Q[N],szq,frnt,bck;
bool bellman(int src){
	Q[0]=src;
	szq=1;
	frnt=N1;
	bck=0;
	memset(dis,'?',n*sizeof dis[0]);
	dis[src]=0;
	int cnt=n;
	inQ[src]=++vid;
	while(szq && cnt--){
		int s=szq;
		while(s--){
			int u=Q[szq--,frnt=(++frnt & N1)];
			inQ[u]=0;
			for(int e=head[u];~e;e=nxt[e]){
				int v=to[e];
				int c=cost[e];
				int d=dis[u]+c;
				if(dis[v]>d){
					if(!cnt)re false;
					dis[v]=d;
					if(inQ[v]!=vid){
						inQ[v]=vid;
						Q[szq++,bck=(++bck & N1)]=v;
					}
				}
			}
		}
	}
	re true;
}
int a,b,c;
char s[2];
int main()
{
	while(true){
		scanf("%d",&n);
		if(!n)re 0;
		n+=2;
		init();
		scanf("%d",&m);
		while(m--){
			scanf("%d %d %s %d",&a,&b,&s,&c);
			if(s[0]=='g')add_edge(a+b,a-1,-c-1);
			else add_edge(a-1,a+b,c-1);
		}
		for(int i=0;i<n-1;++i)
```

```
                    add_edge(n-1,i,0);
            if(bellman(n-1))
                    puts("lamentable kingdom");
            else puts("successful conspiracy");
        }
        re 0;
}
```

Number Of Solutions in linear equation

```
int n,rhs,cof[maxn];
int countSol(){
    int dp[rhs + 1];
    memset(dp, 0, sizeof(dp));
    dp[0] = 1;
    for (int i = 0; i < n; i++)
        for (int j = cof[i]; j <= rhs; j++)
            dp[j] += dp[j - cof[i]];
    return dp[rhs];
}
```

nCr

```
long long mod(1e9+7);

long long f[100005];
long long pow(long long a, long long b, long long MOD)
{
    long long x=1,y=a;
    while(b > 0)
    {
        if(b%2 == 1)
        {
            x=(x*y);
            if(x>MOD) x%=MOD;
        }
        y = (y*y);
        if(y>MOD) y%=MOD;
        b /= 2;
    }
    return x;
}
inline long long InverseEuler(long long n, long long MOD)
{
    return pow(n,MOD-2,MOD);
}

inline long long C(long long n, long long r, long long MOD)
{
    return (f[n]*((InverseEuler(f[r], MOD) * InverseEuler(f[n-r], MOD)) %
MOD)) % MOD;
}


f[0]=1;
```

```
    for(ll i=1;i<=100000;++i)
        f[i]=(f[i-1]*i)%MOD;
```

Strings

Rabin Karp

```
typedef long long ll;

#define R first.first
#define C first.second
#define S second

const int N = 1e5+5, M = 4e4+7;
const double EPS = 1e-6;
const int OO = 0x3f3f3f3f;

ll fixMod(ll a, ll b){
	return (a%b + b)%b;
}

void pushBack(ll& h, ll base, ll mod, char c){
	h = ((h*base)%mod + c)%mod;
}

void pushFront(ll& h, ll p, ll mod, char c){
	h = (h + (c*p)%mod)%mod;
}

void popFront(ll& h, ll p, ll mod, char c){
	h = fixMod((h - (p*c)%mod), mod);
}

void popBack(ll& h, ll inv, ll mod, char c){
	h = (fixMod(h-c, mod)*inv)%mod;
}

ll power(ll a, ll b, ll m){
	if(!b)  return 1;
	ll t = power(a, b>>1, m);
	return ((t*t)%m * (b&1 ? a : 1))%m;
}

int t, n, k;
char s[N];

int main(){
	scanf("%d", &t);
	while(t--){
		scanf("%d %d %s", &n, &k, s);
		ll hw1 = 0, base1 = 129, mod1 = 1e9+7, p1 = 1;
		ll hw2 = 0, base2 = 131, mod2 = 1e9+9, p2 = 1;
		for(int i = 0 ; i < k ; ++i){
```

```
                    pushBack(hw1, base1, mod1, s[i]);
                    pushBack(hw2, base2, mod2, s[i]);
                    if(i) p1 = (p1*base1)%mod1;
                    if(i) p2 = (p2*base2)%mod2;
            }
            set<pair<int, int> > st;
            st.insert({hw1, hw2});
            for(int i = k ; i < n ; ++i){
                    popFront(hw1, p1, mod1, s[i-k]);
                    popFront(hw2, p2, mod2, s[i-k]);
                    pushBack(hw1, base1, mod1, s[i]);
                    pushBack(hw2, base2, mod2, s[i]);
                    st.insert({hw1, hw2});
            }
            printf("%d\n", st.size());
      }
      return 0;
}
```

Hashing

```
#include <bits/stdc++.h>
using namespace std;
const int B1 = 256;
const int B2 = 128;
const int MOD1 = 2000000011;
const int MOD2 = 1000000007;
const int N = 1e5 + 5;
char s[N];
int mul(int a , int b , int m) {
    return a * 1ll * b % m;
}
int add(long long a , int b , int m) {
    a += b;
    while (a >= m)a -= m;
    while (a < 0) a += m;
    return a;
}
pair<int, int> prefix[N];
void hash_prefix() {
    int h1, h2;
    h1 = h2 = 0;
    for (int i = 0; s[i]; i++) {
        h1 = ((1LL * h1 * B1) % MOD1 + s[i]) % MOD1;
        h2 = ((1LL * h2 * B2) % MOD2 + s[i]) % MOD2;
        prefix[i] = make_pair(h1, h2);
    }
}
pair<int, int> suffix[N];
void hash_suffix(){
    int h1 , h2 , pw1 , pw2;
    h1 = h2 = 0;
    pw1 = pw2 = 1;
    int sz = strlen(s);
    for(int i = sz-1 ; i>=0;i--){
```

```
        h1 = (h1 + (1LL * pw1 * s[i])%MOD1 )%MOD1;
        h2 = (h2 + (1LL * pw2 * s[i])%MOD2 )%MOD2;
        pw1 = (1LL*pw1*B1)%MOD1;
        pw2 = (1LL*pw2*B2)%MOD2;
        suffix[i] = {h1 , h2};
    }
}
int p[N] , pp[N];
void pre() { // one time for all test cases
    p[0] = pp[0] = 1;
    for (int i = 1 ;i < N ;i++) {
        p[i] = mul(p[i - 1] , B1 , MOD1);
        pp[i] = mul(pp[i - 1] , B2 , MOD2);
    }
}
pair<int , int> get(int l , int r) {
    if (!l) return prefix[r];
    int len = r - l + 1;
    return {add(prefix[r].first , -mul(p[len] , prefix[l - 1].first , MOD1)
, MOD1)
            , add(prefix[r].second, -mul(pp[len] , prefix[l - 1].second ,
MOD2) , MOD2)};
}
```

KMP

```
const int N = 1e7;
char pat[N], str[N];
int f[N];

int PF(int len, char c) {
  while (len && pat[len] != c)
    len = f[len - 1];
  if (pat[len] == c)
    len++;
  return len;
}

void computeF() {
  f[0] = 0;
  int len = 0;
  if (*pat) {
    for (int i = 1; pat[i]; i++) {
      len = PF(len, pat[i]);
      f[i] = len;
    }
  }
}

vector<int> match() {
  vector<int> ind;
  int len = 0;
  for (int i = 0; str[i]; i++) {
    len = PF(len, str[i]);
```

```
    if (!pat[len])
      ind.push_back(i - len + 1), len = f[len - 1];
  }
  return ind;
}
```

Z-Algorithm

```
vector<int> Z(string s) {
    vector<int> z(s.size() , 0);
    int l = 0 , r = 0;
    for (int i = 1 ;i < s.size() ;i++) {
        if (i <= r)
            z[i] = min(z[i - l] , r - i + 1);

        while (i + z[i] < s.size() && s[i + z[i]] == s[z[i]])
            z[i]++;

        if (i + z[i] - 1 > r)
            l = i , r = i + z[i] - 1;
    }

    return z;
}
int main() {
    string patt = "aab";
    string str = "abbaaabbab";
    vector<int> x = Z(patt + "$" + str); // $ removes unnecessary
comparisons
}
```

Trie

```
struct Hash{
  int operator()(const pair<int,char>&p)const{
    return p.first*128+p.second;
  }
};

unordered_map<pair<int,char>,int,Hash> trie;
vector<bool> isEnd;

int addNode(){
  isEnd.push_back(false);
  return sz(isEnd)-1;
}

void init(){
  trie.clear();
  isEnd.clear();
  addNode();
}

bool insert(const char* s){
  int cur=0;
```

```
  for(;*s;s++){
    int nxt=trie.insert({{cur,*s},-1}).first->second;
    if(nxt==-1)nxt=trie[{cur,*s}]=addNode();
    cur=nxt;
    if(isEnd[cur])return true;
  }
  isEnd[cur]=true;
  return false;
}
--------------------------------------------------------------------------------
---------
int trie[100100][128];
bool isEnd[100100];
int nodeCnt;

int addNode(){
  memset(trie[nodeCnt],-1,sizeof trie[nodeCnt]);
  isEnd[nodeCnt]=false;
  return nodeCnt++;
}

void init(){
  nodeCnt=0;
  addNode();
}

void insert(const char* s){
  int cur=0;
  for(;*s;s++){
    int &nxt=trie[cur][(int)*s];
    if(nxt==-1)nxt=addNode();
    cur=nxt;
  }
  isEnd[cur]=true;
}
--------------------------------------------------------------------------------
--------------------
int addNode(){
  trie.push_back(vector<int>(128,-1));
  isEnd.push_back(false);
  return sz(isEnd)-1;
}

void init(){
  trie.clear();
  isEnd.clear();
  addNode();
}

void insert(const char* s){
  int cur=0;
  for(;*s;s++){
    int nxt=trie[cur][(int)*s];
    if(nxt==-1)nxt=trie[cur][(int)*s]=addNode();
    cur=nxt;
```

```
  }
  isEnd[cur]=true;
}
```

Suffix Array N log2n

```
using namespace std;

const int N = 1e6+6, M = 2e6+5, OO = 0x3f3f3f3f;

int sufA[N];
int rnk[N];
int trnk[N];

char s[N];
int np1;

void buildSufA(){
	for(np1 = 0 ; !np1 || s[np1-1] ; ++np1){
		sufA[np1] = np1;
		rnk[np1] = s[np1];
	}
	int len = 1;
	do{
		auto cmp = [len](int a, int b){
			return rnk[a]<rnk[b] || rnk[a]==rnk[b] &&
rnk[a+len]<rnk[b+len];
		};
		sort(sufA, sufA+np1, cmp);
		//Compute New Rank
		for(int i = 1 ; i < np1 ; ++i)
			trnk[i] = trnk[i-1] + cmp(sufA[i-1], sufA[i]);
		//Update Ranks
		for(int i = 0 ; i < np1 ; ++i)
			rnk[sufA[i]] = trnk[i];
		len <<= 1;
	}while(trnk[np1-1] != np1-1);
}

int LCP[N];
void buildLCP(){
	int cnt = 0;
	for(int i = 0 ; i < np1-1 ; ++i){
		int j = sufA[rnk[i]-1];
		while(s[i+cnt] == s[j+cnt]) ++cnt;
		LCP[rnk[i]] = cnt;
		if(cnt) cnt--;
	}
}

int n, t, q;

int main(){
	scanf("%d", &n);
	scanf("%s", s);
```

```
	scanf("%s", s+n+1);
	s[n] = '#';
	buildSufA();
	buildLCP();
	int mx = 0, mxi;
	for(int i = 1 ; i < np1 ; ++i){
		if((sufA[i] < n) != (sufA[i-1] < n)){
			if(LCP[i] > mx){
				mx = LCP[i];
				mxi = sufA[i];
			}
		}
	}
	s[mxi+mx] = 0;
	printf("%s\n", s+mxi);
	return 0;
}
```

Count Distinct Substrings

```
const int N = 1e5 + 3, NODES = 2 * N;

int child[NODES][128], len[NODES], fail[NODES], nNodes, last;
int childSz[NODES];
char childChars[NODES][128];

int addNode(int Len) {
    memset(child[nNodes], -1, sizeof child[0]);
    childSz[nNodes] = 0;
    len[nNodes] = Len;
    return nNodes++;
}

void init() {
    nNodes = 1;
    last = 1;
    addNode(0);
    fill(child[0], child[0] + 128, 1);
    iota(childChars[0], childChars[0] + 128, '\0');
    childSz[0] = 128;
    len[0] = -1;
}

int cpyNode(int id, int Len) {
    len[nNodes] = Len;
    memcpy(child[nNodes], child[id], sizeof child[0]);
    childSz[nNodes] = childSz[id];
    memcpy(childChars[nNodes], childChars[id], childSz[id] *
sizeof(childChars[0][0]));
    fail[nNodes] = fail[id];
    return nNodes++;
}

void addEdge(int f, int t, char c) {
    child[f][c] = t;
```

```
    childChars[f][childSz[f]++] = c;
}


void addChar(char c) {
    int cur = last;
    last = addNode(len[last] + 1);
    while (!~child[cur][c]) {
        addEdge(cur, last, c);
        cur = fail[cur];
    }
    int nxt = child[cur][c], clone = nxt;
    if (len[cur] + 1 != len[nxt]) {
        clone = cpyNode(nxt, len[cur] + 1);
        while (child[cur][c]==nxt) {
            child[cur][c]=clone;
            cur = fail[cur];
        }
        fail[nxt] = clone;
    }
    fail[last] = clone;
}

void print() {
    for (int i = 1; i < nNodes; ++i) {
        printf("i=%d l=%d f=%d:", i, len[i], fail[i]);
        for (int k = 0; k < childSz[i]; ++k) {
            char c = childChars[i][k];
            int j = child[i][c];
            printf(" (%d, %c)", j, c);
        }
        puts("");
    }

    for (int i = 1; i < nNodes; ++i) {
        for (int k = 0; k < childSz[i]; ++k) {
            char c = childChars[i][k];
            int j = child[i][c];
            printf("%d %d %c\n", i, j, c);
        }
    }
    printf("last=%d\n", last);
    printf("------------------\n");
    fflush(stdout);
}

void build(const char *s) {
    init();
    for (; *s; ++s)
        addChar(*s);//, print();
}

long long mem[NODES];
int vis[NODES], vid;
```

```
long long countDistSubstring(int cur = 1) {
    long long &ret = mem[cur];
    if (vis[cur] == vid)
        return ret;
    vis[cur] = vid;
    ret = 0;
    for (int i = 0; i < childSz[cur]; ++i)
        ret +=1+countDistSubstring(child[cur][childChars[cur][i]]);
    return ret;
}

char s[N];

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "rt", stdin);
    freopen("output.txt", "wt", stdout);
#endif
    int t;
    scanf("%d", &t);
    while (t--) {
        scanf("%s", s);
        build(s);
        ++vid;
        printf("%lld\n", countDistSubstring() );
    }
    return 0;
}
```

Longest Palindromic Substring (Mancher)

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

using namespace std;

template <class T>
int manacher (const T & v) {
    int n = v.size() * 2 + 1;
    T arr(n) , res(n , 0);
    arr[0] = -1 , arr.back() = -3;

    for (int i = 1 ;i < n - 1;i++)
        arr[i] = i&1 ? v[i / 2] : -1;

    int c = 0 , r = 0 , mx = 1;
    for (int i = 1 ;i < arr.size() - 1;i++) {
        int mirror = 2 * c - i;

        if (i < r)
            res[i] = min(res[mirror] , r - i);

        while (arr[i - res[i] - 1] == arr[i + res[i] + 1])
            res[i]++;

        if (i + res[i] > r) {
```

```
            c = i;
            r = i + res[i];
        }

        if (i&1)
            mx = max(mx , res[i] + (res[i] % 2 == 0));
        else
            mx = max(mx , res[i] + (res[i]&1));
    }

    return mx;
}
```

Number Of Palindromes (Mancher)

```
template <class T>
int manacher (const T & v) {
    int n = v.size() * 2 + 1;
    T arr(n) , res(n , 0);
    arr[0] = -1 , arr.back() = -3;

    for (int i = 1 ;i < n - 1;i++)
        arr[i] = i&1 ? v[i / 2] : -1;

    int c = 0 , r = 0 , ret = 0;
    for (int i = 1 ;i < arr.size() - 1;i++) {
        int mirror = 2 * c - i;

        if (i < r)
            res[i] = min(res[mirror] , r - i);

        while (arr[i - res[i] - 1] == arr[i + res[i] + 1])
            res[i]++;

        if (i + res[i] > r) {
            c = i;
            r = i + res[i];
        }

        if (i&1)
            ret += res[i] / 2 + 1;
        else
            ret += (res[i] + 1) / 2;
    }

    return ret;
}
```

Aho Algorithm

```
const int N = 1005;
const int M = 26;
int trie[N][M];
int go[N][M];
int mrk[N], f[N];
```

```
int ptr = 1;
void BFS(){

  queue<int> q;
      for(int i = 0; i < M; i++)
            if(trie[0][i])
                  q.push(trie[0][i]), f[trie[0][i]] = 0;
      for(int i = 0; i < M; i++) go[0][i] = trie[0][i];
      while(!q.empty()) {
            int x = q.front(); q.pop();
            for(int i = 0; i < M; i++) {
                  if(trie[x][i]) {
                        int y = trie[x][i];
                        f[y] = f[x];
                        while(f[y] && !trie[f[y]][i])
                              f[y] = f[f[y]];
                        if(trie[f[y]][i]) f[y] = trie[f[y]][i];
                        mrk[y] += mrk[f[y]];
                        q.push(y);
                  }
                  if(trie[x][i]) go[x][i] = trie[x][i];
                  else go[x][i] = go[f[x]][i];
            }
      }

}
void ins(string x){
  int v;  cin >> v;
  int cur = 0;
  for(int i = 0; i < x.size(); i++){
      if(!trie[cur][x[i]-'a'])
        trie[cur][x[i]-'a'] = ptr++;
      cur = trie[cur][x[i]-'a'];
  }
  mrk[cur] += v;
}
```

Suffix Automaton

```
struct state {
    int len, link;
    long long cnt; // number of times the strings in this state occur in the
original string
    bool is_final;
    map<char, int> next;

    state() {
        cnt = len = is_final = 0;
        link = -1;
        next.clear();
    }
};

const int MAXLEN = 200002;
```

```
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
    st[0] = state();
    sz = last = 0;
    ++sz;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur] = state();
    st[cur].len = st[last].len + 1;
    st[cur].cnt = 1;
    int p;
    for (p = last; p != -1 && !st[p].next.count(c); p = st[p].link)
        st[p].next[c] = cur;
    if (p == -1)
        st[cur].link = 0;
    else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len)
            st[cur].link = q;
        else {
            int clone = sz++;
            st[clone] = state();
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            st[clone].cnt = 0;
            for (; p != -1 && st[p].next[c] == q; p = st[p].link)
                st[p].next[c] = clone;
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

void sa_buildcnt() {
    vector<pair<int, int> > v;
    for (int i = 1; i < sz; i++) v.emplace_back(st[i].len, i);
    sort(v.rbegin(), v.rend());
    for (int i = 0; i < sz - 1; i++) {
        int suf = st[v[i].second].link;
        st[suf].cnt += st[v[i].second].cnt;
    }
}

void sa_build(const string &str) {
    sa_init();
    for (char c : str)
        sa_extend(c);
    int cur = last;
    while (cur) {
        st[cur].is_final = true;
```

```
        cur = st[cur].link;
    }
    sa_buildcnt();
}
```

Suffix Array (N log N)

```
#include <bits/stdc++.h>
#define ll long long
#define mp make_pair
#define MOD 1000000007

using namespace std;

typedef pair<int,int> ii;

const int n_ = 200005;

char str[n_];
int rnk[n_], suf[n_], newRnk[n_], LCP[n_], head[128], nxt[n_], rnkStrt[n_],
newSuf[n_];

struct cmp{
  int len;
  bool operator()(int a, int b)const {
    return rnk[a] < rnk[b] || (rnk[a] == rnk[b] && rnk[a + len] < rnk[b +
len]);
  }
};

void buildSuffArray(){
  int len = 0;
  memset(head, -1, sizeof head);
  for(; !len || str[len - 1]; len++){
    nxt[len] = head[str[len]];
    head[str[len]] = len;
  }
  int ng = 0, ns = 0;
  for(int i=0; i<128; i++){
    if(head[i] == -1) continue;
    rnkStrt[ng] = ns;
    for(int j=head[i]; j != -1 ; j = nxt[j]){
      suf[ns++] = j;
      rnk[j] = ng;
    }
    ng++;
  }
  newRnk[len - 1] = -1;
  newSuf[0] = len - 1;
  for(int h=1; newRnk[len - 1] != len - 1; h <<= 1){
    cmp c = {h};
    for(int i=0; i<len; i++){
      int j = suf[i] - h;
      if(j < 0) continue;
```

```
      newSuf[rnkStrt[rnk[j]]++] = j;
    }
    for(int i=1; i<len; i++){
      bool newGrp = c(newSuf[i-1], newSuf[i]);
      newRnk[i] = newRnk[i-1] + newGrp;
      if(newGrp)
        rnkStrt[newRnk[i]] = i;
    }
    for(int i=0; i<len; i++){
      suf[i] = newSuf[i];
      rnk[suf[i]] = newRnk[i];
    }
  }
}

void buildLCP(){
  int len = 0;
  for(int i=0; str[i]; i++){
    int j = suf[rnk[i] - 1];
    while(str[i + len] == str[j + len])
      len++;
    LCP[rnk[i]] = len;
    if(len) len--;
  }
}

int T, k;
int suf2[100005];

int main()
{
  //freopen("input.txt", "r", stdin);
  //freopen("output.txt", "w", stdout);
  scanf("%d", &T);
  while(T--){
    scanf("%d %s", &k, str);
    buildSuffArray();
    int e = e=strlen(str);
    for(int i=0; i<e; i++) suf2[i] = suf[i];
    reverse(str, str+e);
    buildSuffArray();
    int i;
    for(i=0; i<e; i++){
      if(k <= e - suf2[i]) break;
      k -= e - suf2[i];
    }
    for(int j=0; j<e; j++){
      if(e - suf[j] + e - suf2[i] <= e) k--;
      if(k == 0){
        for(int x=e-1; x>=suf[j]; x--){
          putchar(str[x]);
        }
        for(int x=e-1; x>=suf2[i]; x--){
          putchar(str[e - x]);
        }
```

```
        break;
      }
    }
    puts("");
  }
  return 0;
}
```

Suffix Array And LCP (Wolf)

```
template<typename T>
vector<int> sort_cyclic_shifts(const T &s) {
    int n = s.size();
    const int alphabet = 256; // change according to the problem
    vector<int> p(n) , g(n) , cnt(max(n , alphabet) , 0);

    for (int i = 0 ;i < n ;i++) cnt[s[i]]++;
    for (int i = 1 ;i < alphabet ;i++) cnt[i] += cnt[i - 1];
    for (int i = 0 ;i < n ;i++) p[--cnt[s[i]]] = i;

    g[p[0]] = 0;
    int groups = 1;
    for (int i = 1 ;i < n ;i++) {
        groups += (s[p[i - 1]] != s[p[i]]);
        g[p[i]] = groups - 1;
    }

    vector<int> pn(n) , gn(n);
    for (int h = 0 ; (1 << h) < n ;h++) {
        fill(cnt.begin() , cnt.begin() + groups , 0);

        for (int i = 0 ;i < n ;i++) pn[i] = (p[i] - (1 << h) + n) % n;
        for (int i = 0 ;i < n ;i++) cnt[g[pn[i]]]++;
        for (int i = 1 ;i < groups ;i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1 ;i >= 0 ;i--) p[--cnt[g[pn[i]]]] = pn[i];

        gn[p[0]] = 0;
        groups = 1;
        for (int i = 1 ;i < n ;i++) {
            groups += tie(g[p[i]] , g[(p[i] + (1 << h)) % n])
                      != tie(g[p[i - 1]] , g[(p[i - 1] + (1 << h)) % n]);
            gn[p[i]] = groups - 1;
        }
        g = gn;
    }

    return p;
}

template <typename T>
vector<int> build_lcp(T const& s, const vector<int> &p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
```

```
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}

template <typename T>
vector<int> suffix_array_construction(T s) {
    s.push_back(*min_element(s.begin() , s.end()) - 1); // don't if you want
to sort cyclic strings
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
```

General

DCMP

```
int dcmp(double a, double b) {
      return fabs(a-b) <= EPS ? 0 : a < b ? -1 : 1;
}
```

Fast Input Output

```
int readInt(){
      char c=gc();
      int sign=1,ret=0;
      while(isspace(c))   c=gc();
      if(c=='-')   sign=-1,c=gc();
      while(isdigit(c)) ret*=10,ret+=c-'0',c=gc();
      return ret*sign;
}

double readDouble(){
      char c=gc();
      int sign=1;
      double ret=0,t=0.1;
      while(isspace(c))   c=gc();
      if(c=='-')   sign=-1,c=gc();
      while(isdigit(c)) ret*=10,ret+=c-'0',c=gc();
      if(c=='.')
```

```
	{
		c=gc();
		while(isdigit(c))	ret+=t*(c-'0'),t/=10,c=gc();
	}
	return ret*sign;
}
```

Compress Array

```
void compress(){
    sort(values,values+sz);
    sz = unique(values,values+sz)-values ;
    for(int i=0;i<n;++i) arr[i] = lower_bound(values,values+sz,arr[i]) -
values ;

}
```

Hash Pair (Unordered Map)

```
struct hash_pair {
    template <class T1, class T2>
    size_t operator()(const pair<T1, T2>& p) const
    {
        auto hash1 = hash<T1>{}(p.first);
        auto hash2 = hash<T2>{}(p.second);
        return hash1 ^ hash2;
    }
};
```

Big Int

```
struct bigint {
	vector<int> a;
	int sign;

	bigint() :
		sign(1) {
	}

	bigint(long long v) {
		*this = v;
	}

	bigint(const string &s) {
		read(s);
	}

	void operator=(const bigint &v) {
		sign = v.sign;
		a = v.a;
```

```
	}

	void operator=(long long v) {
		sign = 1;
		if (v < 0)
			sign = -1, v = -v;
		for (; v > 0; v = v / base)
			a.push_back(v % base);
	}

	bigint operator+(const bigint &v) const {
		if (sign == v.sign) {
			bigint res = v;

			for (int i = 0, carry = 0; i < (int) max(a.size(),
v.a.size()) || carry; ++i) {
				if (i == (int) res.a.size())
					res.a.push_back(0);
				res.a[i] += carry + (i < (int) a.size() ? a[i] :
0);
				carry = res.a[i] >= base;
				if (carry)
					res.a[i] -= base;
			}
			return res;
		}
		return *this - (-v);
	}

	bigint operator-(const bigint &v) const {
		if (sign == v.sign) {
			if (abs() >= v.abs()) {
				bigint res = *this;
				for (int i = 0, carry = 0; i < (int) v.a.size() ||
carry; ++i) {
					res.a[i] -= carry + (i < (int) v.a.size() ?
v.a[i] : 0);
					carry = res.a[i] < 0;
					if (carry)
						res.a[i] += base;
				}
				res.trim();
				return res;
			}
			return -(v - *this);
		}
		return *this + (-v);
	}

	void operator*=(int v) {
		if (v < 0)
			sign = -sign, v = -v;
		for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
			if (i == (int) a.size())
				a.push_back(0);
```

```
                        long long cur = a[i] * (long long) v + carry;
                        carry = (int) (cur / base);
                        a[i] = (int) (cur % base);
                        //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur),
"c"(base));
                }
                trim();
        }

        bigint operator*(int v) const {
                bigint res = *this;
                res *= v;
                return res;
        }

        friend pair<bigint, bigint> divmod(const bigint &a1, const bigint
&b1) {
                int norm = base / (b1.a.back() + 1);
                bigint a = a1.abs() * norm;
                bigint b = b1.abs() * norm;
                bigint q, r;
                q.a.resize(a.a.size());

                for (int i = a.a.size() - 1; i >= 0; i--) {
                        r *= base;
                        r += a.a[i];
                        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
                        int s2 = r.a.size() <= b.a.size() - 1 ? 0 :
r.a[b.a.size() - 1];
                        int d = ((long long) base * s1 + s2) / b.a.back();
                        r -= b * d;
                        while (r < 0)
                                r += b, --d;
                        q.a[i] = d;
                }

                q.sign = a1.sign * b1.sign;
                r.sign = a1.sign;
                q.trim();
                r.trim();
                return make_pair(q, r / norm);
        }

        bigint operator/(const bigint &v) const {
                return divmod(*this, v).first;
        }

        bigint operator%(const bigint &v) const {
                return divmod(*this, v).second;
        }

        void operator/=(int v) {
                if (v < 0)
                        sign = -sign, v = -v;
                for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
```

```
                long long cur = a[i] + rem * (long long) base;
                a[i] = (int) (cur / v);
                rem = (int) (cur % v);
        }
        trim();
}

bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
}

int operator%(int v) const {
        if (v < 0)
                v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
                m = (a[i] + m * (long long) base) % v;
        return m * sign;
}

void operator+=(const bigint &v) {
        *this = *this + v;
}
void operator-=(const bigint &v) {
        *this = *this - v;
}
void operator*=(const bigint &v) {
        *this = *this * v;
}
void operator/=(const bigint &v) {
        *this = *this / v;
}

bool operator<(const bigint &v) const {
        if (sign != v.sign)
                return sign < v.sign;
        if (a.size() != v.a.size())
                return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
                if (a[i] != v.a[i])
                        return a[i] * sign < v.a[i] * sign;
        return false;
}

bool operator>(const bigint &v) const {
        return v < *this;
}
bool operator<=(const bigint &v) const {
        return !(v < *this);
}
bool operator>=(const bigint &v) const {
        return !(*this < v);
}
```

```
	bool operator==(const bigint &v) const {
		return !(*this < v) && !(v < *this);
	}
	bool operator!=(const bigint &v) const {
		return *this < v || v < *this;
	}

	void trim() {
		while (!a.empty() && !a.back())
			a.pop_back();
		if (a.empty())
			sign = 1;
	}

	bool isZero() const {
		return a.empty() || (a.size() == 1 && !a[0]);
	}

	bigint operator-() const {
		bigint res = *this;
		res.sign = -sign;
		return res;
	}

	bigint abs() const {
		bigint res = *this;
		res.sign *= res.sign;
		return res;
	}

	long long longValue() const {
		long long res = 0;
		for (int i = a.size() - 1; i >= 0; i--)
			res = res * base + a[i];
		return res * sign;
	}

	friend bigint gcd(const bigint &a, const bigint &b) {
		return b.isZero() ? a : gcd(b, a % b);
	}
	friend bigint lcm(const bigint &a, const bigint &b) {
		return a / gcd(a, b) * b;
	}

	void read(const string &s) {
		sign = 1;
		a.clear();
		int pos = 0;
		while (pos < (int) s.size() && (s[pos] == '-' || s[pos] ==
'+')) {
			if (s[pos] == '-')
				sign = -sign;
			++pos;
		}
		for (int i = s.size() - 1; i >= pos; i -= base_digits) {
```

```
                int x = 0;
                for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                        x = x * 10 + s[j] - '0';
                a.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1)
                stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
                stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
    }

    static vector<int> convert_base(const vector<int> &a, int old_digits,
int new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
                p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++) {
                cur += a[i] * p[cur_digits];
                cur_digits += old_digits;
                while (cur_digits >= new_digits) {
                        res.push_back(int(cur % p[new_digits]));
                        cur /= p[new_digits];
                        cur_digits -= new_digits;
                }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back())
                res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a, const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
                for (int i = 0; i < n; i++)
```

```
                        for (int j = 0; j < n; j++)
                                res[i + j] += a[i] * b[j];
                return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
                a2[i] += a1[i];
        for (int i = 0; i < k; i++)
                b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i++)
                r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
                r[i] -= a2b2[i];

        for (int i = 0; i < (int) r.size(); i++)
                res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i++)
                res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
                res[i + n] += a2b2[i];
        return res;
}

bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->a, base_digits, 6);
        vector<int> b6 = convert_base(v.a, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
                a.push_back(0);
        while (b.size() < a.size())
                b.push_back(0);
        while (a.size() & (a.size() - 1))
                a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
                long long cur = c[i] + carry;
                res.a.push_back((int) (cur % 1000000));
                carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
```

```
            return res;
        }
};
```

Binary Search

```
//Searching for the last True in TTTTTTTTTTTTTTFFFFFFFFFFFFF pattern
int b_s(int s, int e){
      while (s < e){
            int mid = s + (e - s + 1) / 2;
            if (valid(mid))
                s = mid;
            else
                e = mid - 1;
      }
        return s;
}
//Searching for the first True in FFFFFFFFTTTTTTTTT pattern
int bs(int s, int e){
    while (s < e){
        int mid = (s + (e - s) / 2);
        if (valid(mid))
            e = mid;
        else
            s = mid + 1;
    }
    return s;
}
```

Fast Input

```
inline void scan(int &x) {
    register int c = getchar();
    x = 0;
    int neg = 0;
    for(;((c<48 || c>57) && c != '-');c = getchar());
    if(c=='-') {neg=1;c=getchar();}
    for(;c>47 && c<58;c = getchar()) {x = (x<<1) + (x<<3) + c - 48;}
    if(neg) x=-x;
}
```

Overflow Check

```
long long a = 1e18 , b = 1e9 , tmp;
bool flag = __builtin_add_overflow(a , b , &tmp);
bool flag2 = __builtin_mul_overflow(a , b , &tmp);
bool flag3 = __builtin_sub_overflow(a , b , &tmp);
if(flag)
    cout << "overFlow" << endl;
else cout << tmp << endl;
-----------------------------------------------------
bool overflow(long long a, long long b) {
    long double res = a * b;
    if (a == 0 || b == 0 || res / b == a)
        return false;
```

```
    return true;
}
```

Randomization

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int>(0, i)(rng)
shuffle(permutation.begin(), permutation.end(), rng);
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM =
chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int , int , chash> h;
```

Stress Test

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

int main(){
    system("g++ -lm -O2 -std=c++11 -pipe -o ac.exe _ac.cpp");
    system("g++ -lm -O2 -std=c++11 -pipe -o wa.exe _wa.cpp");


    int tc = 1;
    while (1) {
        ofstream ofs("test.in");

      ////////////////////////////////////////////////
        int t = 2;
        ofs << t << endl;
        while (t--) {
            int n = 5 + rand() % 3, m = rand() % (n * (n - 1) / 2), b = 1
                    + rand() % 5;
            ofs << n << " " << m << " " << b << endl;
            for (int i = 0; i < n; i++)
                ofs << rand() % 10 + 1 << " \n"[i == n - 1];
            for (int i = 0; i < b; i++)
                ofs << rand() % 10 + 1 << " \n"[i == b - 1];
            set<pair<int,int> > s;
            while(m--){
                int a,b;
                do{
                    a=rand()%n+1;
                    b=rand()%n+1;
                }while(a==b || !s.insert({min(a,b),max(a,b)}).second);
                ofs<<a<<" "<<b<<endl;
            }
        }

      ////////////////////////////
```

```
        ofs.close();
        system("ac.exe <test.in >ac.txt");
        system("wa.exe <test.in >wa.txt");
        ifstream acs("ac.txt");
        ifstream was("wa.txt");
        string ac, wa;
        getline(was, wa, (char) EOF);
        getline(acs, ac, (char) EOF);
        was.close();
        acs.close();
        cerr << tc++ << endl;
        if (ac != wa)
            break;
    }
}
```

Sub Masks Of Mask

```
for (int sub = mask; sub!= 0; sub = (sub - 1) & mask);
```

Template

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

#define loop(i,n) for(int i = 0;i < int(n);i++)
#define rloop(i,n) for(int i = int(n);i >= 0;i--)
#define range(i,a,b) for(int i = int(a);i <= int(b);i++)
#define SZ(c) int(c.size())
#define ALL(c) c.begin(), c.end()
#define RALL(c) c.rbegin(), c.rend()
#define PI acos(-1)
#define pb push_back
#define mp make_pair
#define fr first
#define sc second
#define sfi1(v) scanf("%d",&v)
#define sfi2(v1,v2) scanf("%d %d",&v1,&v2)
#define sfi3(v1,v2,v3) scanf("%d %d %d",&v1,&v2,&v3)
#define sfll1(v) scanf("%I64d",&v);
#define sfll2(v1,v2) scanf("%I64d %I64d",&v1,&v2)
#define sfll3(v1,v2,v3) scanf("%I64d %I64d %I64d",&v1,&v2,&v3)
#define endl '\n'

typedef vector<int> vi;
typedef vector<pair<int,int> > vii;
typedef long long ll;
typedef pair<int, int> pii;



int main()
{

#ifndef ONLINE_JUDGE
```

```
        //freopen("in.in", "r", stdin);
        //freopen("out.in", "w", stdout);
#endif
        ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);



    return 0;
}
```

Ternary Search

```
/\
/  \ <====== This curve.

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

Binary Search Doubles

```
double bs(double s, double e) {
    for (double sz = (e - s) / 2; sz > EPS; sz *= 0.5) {
        /// assuming e = s + sz + sz
        if (valid(s + sz)) { /// valid(mid)
            s += sz; /// s = mid
        }
    }
    return s;
}
```

XOR

```
int basis[d]; // basis[i] keeps the mask of the vector whose f value is i
int sz; // Current size of the basis
void insertVector(int mask) {
      for (int i = 0; i < d; i++) {
            if ((mask & 1 << i) == 0) continue; // continue if i != f(mask)
```

```
            if (!basis[i]) { // If there is no basis vector with the i'th
bit set, then insert this vector into the basis
                  basis[i] = mask;
                  ++sz;
                  return;
            }
            mask ^= basis[i]; // Otherwise subtract the basis vector from
this vector
      }
}
```

Gauss XOR

```
struct gauss {
      vector<ll> setOfXors;
      void ins(ll x) {
            for (auto v : setOfXors)
                  x = min(x, x ^ v);
            if (x)
                  setOfXors.push_back(x);
      }
      ll qry(ll x) {
            for (auto v : setOfXors)
                  x = max(x, x ^ v);
            return x;
      }
};
```

Geometry

Library Coach

```
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <complex>
#include <iostream>
using namespace std;

typedef complex<long double> point;
#define sz(a) ((int)(a).size())
#define all(n) (n).begin(),(n).end()
#define EPS 1e-9
#define OO 1e9
#define X real()
#define Y imag()
#define vec(a,b) ((b)-(a))
#define polar(r,t) ((r)*exp(point(0,(t))))
#define angle(v) (atan2((v).Y,(v).X))
#define length(v) ((long double)hypot((v).Y,(v).X))
#define lengthSqr(v) (dot(v,v))
#define dot(a,b) ((conj(a)*(b)).real())
#define cross(a,b) ((conj(a)*(b)).imag())
#define rotate(v,t) (polar(v,t))
```

```
#define rotateabout(v,t,a)  (rotate(vec(a,v),t)+(a))
#define reflect(p,m) ((conj((p)/(m)))*(m))
#define normalize(p) ((p)/length(p))
#define same(a,b) (lengthSqr(vec(a,b))<EPS)
#define mid(a,b) (((a)+(b))/point(2,0))
#define perp(a) (point(-(a).Y,(a).X))
#define colliner pointOnLine

enum STATE {
	IN, OUT, BOUNDRY
};

bool intersect(const point &a, const point &b, const point &p, const point
&q,
		point &ret) {

    //handle degenerate cases (2 parallel lines, 2 identical lines,   line
is 1 point)

    double d1 = cross(p - a, b - a);
    double d2 = cross(q - a, b - a);
    ret = (d1 * q - d2 * p) / (d1 - d2);
    if(fabs(d1 - d2) > EPS) return 1;
    return 0;
}

bool pointOnLine(const point& a, const point& b, const point& p) {
    // degenerate case: line is a point
    return fabs(cross(vec(a,b),vec(a,p))) < EPS;
}

bool pointOnRay(const point& a, const point& b, const point& p) {
    //IMP NOTE: a,b,p must be collinear
    return dot(vec(a,p), vec(a,b)) > -EPS;
}

bool pointOnSegment(const point& a, const point& b, const point& p) {
    if(!colliner(a,b,p)) return 0;
    return pointOnRay(a, b, p) && pointOnRay(b, a, p);
}

long double pointLineDist(const point& a, const point& b, const point& p) {
	// handle degenrate case: (a,b) is point

	return fabs(cross(vec(a,b),vec(a,p)) / length(vec(a,b)));
}

long double pointSegmentDist(const point& a, const point& b, const point& p)
{
	if (dot(vec(a,b),vec(a,p)) < EPS)
		return length(vec(a,p));
	if (dot(vec(b,a),vec(b,p)) < EPS)
		return length(vec(b,p));
	return pointLineDist(a, b, p);
}
```

```
int segmentLatticePointsCount(int x1, int y1, int x2, int y2) {
	return abs(__gcd(x1 - x2, y1 - y2)) + 1;
}

long double triangleAreaBH(long double b, long double h) {
	return b * h / 2;
}

long double triangleArea2sidesAngle(long double a, long double b, long
double t) {
	return fabs(a * b * sin(t) / 2);
}

long double triangleArea2anglesSide(long double t1, long double t2,
		long double s) {
	return fabs(s * s * sin(t1) * sin(t2) / (2 * sin(t1 + t2)));
}

long double triangleArea3sides(long double a, long double b, long double c)
{
	long double s((a + b + c) / 2);
	return sqrt(s * (s - a) * (s - b) * (s - c));
}

long double triangleArea3points(const point& a, const point& b, const point&
c) {
	return fabs(cross(a,b) + cross(b,c) + cross(c,a)) / 2;
}

//count interior
int picksTheorm(int a, int b) {
	return a - b / 2 + 1;
}

//get angle opposite to side a
long double cosRule(long double a, long double b, long double c) {
	 // Handle denom = 0
	long double res = (b * b + c * c - a * a) / (2 * b * c);
	if ( fabs(res-1)<EPS)
		res = 1;
	 if ( fabs(res+1)<EPS)
		res = -1;
	return acos(res);
}

long double sinRuleAngle(long double s1, long double s2, long double a1) {
	 // Handle denom = 0
	long double res = s2 * sin(a1) / s1;
	if ( fabs(res-1)<EPS)
		res = 1;
	 if ( fabs(res+1)<EPS)
		res = -1;
	return asin(res);
}
```

```
long double sinRuleSide(long double s1, long double a1, long double a2) {
        // Handle denom = 0
       long double res = s1 * sin(a2) / sin(a1);
       return fabs(res);
}

int circleLineIntersection(const point& p0, const point& p1, const point&
cen,
              long double rad, point& r1, point & r2) {

       // handle degenerate case if p0 == p1
       long double a, b, c, t1, t2;
       a = dot(p1-p0,p1-p0);
       b = 2 * dot(p1-p0,p0-cen);
       c = dot(p0-cen,p0-cen) - rad * rad;
       double det = b * b - 4 * a * c;
       int res;
       if (fabs(det) < EPS)
              det = 0, res = 1;
       else if (det < 0)
              res = 0;
       else
              res = 2;
       det = sqrt(det);
       t1 = (-b + det) / (2 * a);
       t2 = (-b - det) / (2 * a);
       r1 = p0 + t1 * (p1 - p0);
       r2 = p0 + t2 * (p1 - p0);
       return res;
}

int circleCircleIntersection(const point &c1, const long double&r1,
              const point &c2, const long double&r2, point &res1, point
&res2) {
       if (same(c1,c2) && fabs(r1 - r2) < EPS) {
              res1 = res2 = c1;
              return fabs(r1) < EPS ? 1 : OO;
       }
       long double len = length(vec(c1,c2));
       if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) - len) < EPS) {
              point d, c;
              long double r;
              if (r1 > r2)
                     d = vec(c1,c2), c = c1, r = r1;
              else
                     d = vec(c2,c1), c = c2, r = r2;
              res1 = res2 = normalize(d) * r + c;
              return 1;
       }
       if (len > r1 + r2 || len < fabs(r1 - r2))
              return 0;
       long double a = cosRule(r2, r1, len);
       point c1c2 = normalize(vec(c1,c2)) * r1;
       res1 = rotate(c1c2,a) + c1;
```

```
	res2 = rotate(c1c2,-a) + c1;
	return 2;
}

void circle2(const point& p1, const point& p2, point& cen, long double& r) {
	cen = mid(p1,p2);
	r = length(vec(p1,p2)) / 2;
}

bool circle3(const point& p1, const point& p2, const point& p3, point& cen,
		long double& r) {
	point m1 = mid(p1,p2);
	point m2 = mid(p2,p3);
	point perp1 = perp(vec(p1,p2));
	point perp2 = perp(vec(p2,p3));
	bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
	r = length(vec(cen,p1));
	return res;
}

STATE circlePoint(const point & cen, const long double & r, const point& p)
{
	long double lensqr = lengthSqr(vec(cen,p));
	if (fabs(lensqr - r * r) < EPS)
		return BOUNDRY;
	if (lensqr < r * r)
		return IN;
	return OUT;
}

int tangentPoints(const point & cen, const long double & r, const point& p,
		point &r1, point &r2) {
	STATE s = circlePoint(cen, r, p);
	if (s != OUT) {
		r1 = r2 = p;
		return s == BOUNDRY;
	}
	point cp = vec(cen,p);
	long double h = length(cp);
	long double a = acos(r / h);
	cp = normalize(cp) * r;
	r1 = rotate(cp,a) + cen;
	r2 = rotate(cp,-a) + cen;
	return 2;
}

typedef pair<point, point> segment;

void getCommonTangents(point c1, double r1, point c2, double r2,
vector<segment> &res) {
  if (r1 < r2) swap(r1, r2), swap(c1, c2);
  double d = length(c1 - c2);
  double theta = acos((r1 - r2) / d);
  point v = c2 - c1;
  v = v / hypot(v.imag(), v.real());
```

```
  point v1 = v * exp(point(0, theta));
  point v2 = v * exp(point(0, -theta));
  res.clear();
  res.push_back(segment(c1 + v1 * r1, c2 + v1 * r2));
  res.push_back(segment(c1 + v2 * r1, c2 + v2 * r2));
  theta = acos((r1 + r2) / d);
  v1 = v * exp(point(0, theta));
  v2 = v * exp(point(0, -theta));
  res.push_back(segment(c1 + v1 * r1, c2 - v1 * r2));
  res.push_back(segment(c1 + v2 * r1, c2 - v2 * r2));
}

// minimum enclosing circle
//init p array with the points and ps with the number of points
//cen and rad are result circle
//you must call random_shuffle(p,p+ps); before you call mec
#define MAXPOINTS 100000
point p[MAXPOINTS], r[3], cen;
int ps, rs;
long double rad;

void mec() {
	if (rs == 3) {
		circle3(r[0], r[1], r[2], cen, rad);
		return;
	}
	if (rs == 2 && ps == 0) {
		circle2(r[0], r[1], cen, rad);
		return;
	}
	if (!ps) {
		cen = r[0];
		rad = 0;
		return;
	}
	ps--;
	mec();
	if (circlePoint(cen, rad, p[ps]) == OUT) {
		r[rs++] = p[ps];
		mec();
		rs--;
	}
	ps++;

}

//to check if the points are sorted anti-clockwise or clockwise
//remove the fabs at the end and it will return -ve value if clockwise
long double polygonArea(const vector<point>&p) {
	long double res = 0;
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		res += cross(p[i],p[j]);
	}
	return fabs(res) / 2;
```

```
}

// return the centroid point of the polygon
// The centroid is also known as the "centre of gravity" or the "center of
mass". The position of the centroid
// assuming the polygon to be made of a material of uniform density.
point polyginCentroid(vector<point> &polygon) {
	long double a = 0;
	 long double x=0.0,y=0.0;
	 for (int i = 0; i < (int) polygon.size(); i++) {
		int j = (i + 1) % polygon.size();

		x += (polygon[i].X + polygon[j].X) * (polygon[i].X *
polygon[j].Y
				- polygon[j].X * polygon[i].Y);

		y += (polygon[i].Y + polygon[j].Y) * (polygon[i].X *
polygon[j].Y
				- polygon[j].X * polygon[i].Y);

		a += polygon[i].X * polygon[j].Y - polygon[i].Y *
polygon[j].X;
	 }

	 a *= 0.5;
	 x /= 6 * a;
	 y /= 6 * a;

	 return point(x,y);
}

int picksTheorm(vector<point>& p) {
	long double area = 0;
	int bound = 0;
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		area += cross(p[i],p[j]);
		point v = vec(p[i],p[j]);
		bound += abs(__gcd((int) v.X, (int) v.Y));
	}
	area /= 2;
	area = fabs(area);
	return round(area - bound / 2 + 1);
}

void polygonCut(const vector<point>& p, const point&a, const point&b,
vector<
		point>& res) {
	res.clear();
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		bool in1 = cross(vec(a,b),vec(a,p[i])) > EPS;
		bool in2 = cross(vec(a,b),vec(a,p[j])) > EPS;
		if (in1)
			res.push_back(p[i]);
```

```
            if (in1 ^ in2) {
                  point r;
                  intersect(a, b, p[i], p[j], r);
                  res.push_back(r);
            }
      }
}

//assume that both are anti-clockwise
void convexPolygonIntersect(const vector<point>& p, const vector<point>& q,
            vector<point>& res) {
      res = q;
      for (int i = 0; i < sz(p); i++) {
            int j = (i + 1) % sz(p);
            vector<point> temp;
            polygonCut(res, p[i], p[j], temp);
            res = temp;
            if (res.empty())
                  return;
      }
}

void voronoi(const vector<point> &pnts, const vector<point>& rect, vector<
            vector<point> > &res) {
      res.clear();
      for (int i = 0; i < sz(pnts); i++) {
            res.push_back(rect);
            for (int j = 0; j < sz(pnts); j++) {
                  if (j == i)
                        continue;
                  point p = perp(vec(pnts[i],pnts[j]));
                  point m = mid(pnts[i],pnts[j]);
                  vector<point> temp;
                  polygonCut(res.back(), m, m + p, temp);
                  res.back() = temp;
            }
      }
}

STATE pointInPolygon(const vector<point>& p, const point &pnt) {
      point p2 = pnt + point(1, 0);
      int cnt = 0;
      for (int i = 0; i < sz(p); i++) {
            int j = (i + 1) % sz(p);
            if (pointOnSegment(p[i], p[j], pnt))
                  return BOUNDRY;
            point r;
            if(!intersect(pnt, p2, p[i], p[j], r))
                  continue;
            if (!pointOnRay(pnt, p2, r))
                  continue;
            if (same(r,p[i]) || same(r,p[j]))
                  if (fabs(r.Y - min(p[i].Y, p[j].Y)) < EPS)
                        continue;
            if (!pointOnSegment(p[i], p[j], r))
```

```
                        continue;
                cnt++;
        }
        return cnt & 1 ? IN : OUT;
}

struct cmp {
        point about;
        cmp(point c) {
                about = c;
        }
        bool operator()(const point& p, const point& q) const {
                double cr = cross(vec(about, p), vec(about, q));
                if (fabs(cr) < EPS)
                        return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);
                return cr > 0;
        }
};

void sortAntiClockWise(vector<point>& pnts) {
        point mn(1 / 0.0, 1 / 0.0);
        for (int i = 0; i < sz(pnts); i++)
                if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y, mn.X))
                        mn = pnts[i];

        sort(all(pnts), cmp(mn));
}

void convexHull(vector<point> pnts, vector<point> &convex) {
        sortAntiClockWise(pnts);
        convex.clear();
        convex.push_back(pnts[0]);
        if (sz(pnts) == 1)
                return;
        convex.push_back(pnts[1]);
        if (sz(pnts) == 2) {
                if (same(pnts[0], pnts[1]))
                        convex.pop_back();
                return;
        }
        for (int i = 2; i <= sz(pnts); i++) {
                point c = pnts[i % sz(pnts)];
                while (sz(convex) > 1) {
                        point b = convex.back();
                        point a = convex[sz(convex) - 2];
                        if (cross(vec(b, a), vec(b, c)) < -EPS)
                                break;
                        convex.pop_back();
                }
                if (i < sz(pnts))
                        convex.push_back(pnts[i]);
        }
```

Point In Polygon

```
#include <bits/stdc++.h>
#define re return
#define ll long long
#define ull unsigned ll
#define ld long double
#define f first
#define s second
#define pi acosl(-1)
#define oo (ll)1e9+8
#define OO 1e18
#define EPS 1e-6
#define Endl '\n'
#define all(v) (v).begin(),(v).end()
#define FX(n) fixed<<setprecision(n)
#define mm(o,k) memset(o,k,sizeof o)
#define IO ios_base::sync_with_stdio(0),cin.tie(0),cout.tie(0);
#define sz(a) ((int)(a).size())
#define dot(a,b) ((conj(a)*(b)).real())
#define cross(a,b) ((conj(a)*(b)).imag())
#define lengthSqr(v) (dot(v,v))
#define same(a,b) (lengthSqr(vec(a,b))<EPS)
#define vec(a,b) ((b)-(a))
#define colliner pointOnLine
#define X real()
#define Y imag()
using namespace std;
const int N=2e3+5,M=N*2;
struct Point {

    ll x, y;
    Point operator-(Point p){
        return {x-p.x,y-p.y};
    }
    bool operator<(Point p)
    {
        return x < p.x || (x == p.x && y < p.y);
    }
};
ll cross_product( Point A, Point B)
{return A.x*B.y-A.y*B.x;
}
ll orr(Point a,Point b,Point c){
    re cross_product(b-a,c-a);
}
ll abo(Point a,Point p){
    re p.y>=a.y;
}
ll ray(Point a,Point p,Point q){
    re (abo(a,q)-abo(a,p))*orr(a,p,q)>0;
}
ll Dot(Point a,Point b){
```

```
    re a.x*b.x+b.x*b.y;
}
ll ind(Point a,Point b,Point p){
    re Dot(a-p,b-p)<=0;
}
ll on(Point a,Point b,Point p){
    re orr(a,b,p)==0&&ind(a,b,p);
}

int in(vector<Point> p,Point a){
    int num=0,n=p.size();
    for(int i=0;i<n;++i){
        if(on(p[i],p[(i+1)%n],a))
        re 0;
        num+=ray(a,p[i],p[(i+1)%n]);
    }
    re num&1;
}
// Returns a list of points on the convex hull
// in counter-clockwise order
vector<Point> convex_hull(vector<Point> A)
{
    int n = A.size(), k = 0;

    vector<Point> ans(2 * n);

    // Sort points lexicographically
    sort(A.begin(), A.end());

    // Build lower hull
    for (int i = 0; i < n; ++i) {

        // If the point at K-1 position is not a part
        // of hull as vector from ans[k-2] to ans[k-1]
        // and ans[k-2] to A[i] has a clockwise turn
        while (k >= 2 && orr(ans[k - 2],
                             ans[k - 1], A[i]) > 0)
            k--;
        ans[k++] = A[i];
    }

    // Build upper hull
    for (int i = n - 2, t = k + 1; i >= 0; --i) {

        // If the point at K-1 position is not a part
        // of hull as vector from ans[k-2] to ans[k-1]
        // and ans[k-2] to A[i] has a clockwise turn
        while (k >= t && orr(ans[k - 2],
                             ans[k - 1], A[i]) > 0)
            k--;
        ans[k++] = A[i];
    }

    // Resize the array to desired size
    ans.resize(k - 1);
```

```
    return ans;
}
int tc;
bool ok;
int main()
{
    IO;
    int t;
    cin >> t;
    while(t--){
        if(ok)cout<<"\n";ok=1;
        cout<<"Case "<<++tc<<"\n";
        vector<Point> pnts,convex;
        int n,m;
        cin >> n >> m;
        for(int i=0;i<n;++i){
            ll a,b;
            cin >> a >> b;
            Point pnt;
            pnt.x=a;
            pnt.y=b;
            pnts.push_back(pnt);
        }
        convex=convex_hull(pnts);
        cout<<convex[0].x<<" "<<convex[0].y<<endl;
        reverse(convex.begin(),convex.end());
        for(auto i:convex)
            cout<<i.x<<" "<<i.y<<endl;
        for(int i=0;i<m;++i){
            ll a,b;
            cin >> a >> b;
            cout<<a<<" "<<b;
            Point pnt;
            pnt.x=a;
            pnt.y=b;
            if(in(convex,pnt))cout<<" is unsafe!\n";
            else cout<<" is safe!\n";
        }
    }
    re 0;
```

Li chao

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100005;

long long f[N], cnt[N], ld[N];
long long sum[N];

struct line {
       long long k, b;
       line(long long _k = 0, long long _b = -1e18) {
```

```
		k = _k;
		b = _b;
	}
	long long get(long long x) {
		return k * x + b;
	}
};

line dummy;

struct node;
node *empty;
struct node {
	line ln;
	node *l, *r;

	node() :
			ln(dummy), l(empty), r(empty) {
	}
	node(line _ln, node* _l, node* _r) :
			ln(_ln), l(_l), r(_r) {
	}
};
node* modify(node* cur, long long l, long long r, line v) {
	if (cur == empty) {
		return new node(v, empty, empty);
	}
	if (cur->ln.get(l) > v.get(l) && cur->ln.get(r) > v.get(r))
		return new node(cur->ln, cur->l, cur->r);
	if (cur->ln.get(l) < v.get(l) && cur->ln.get(r) < v.get(r)) {
		return new node(v, cur->l, cur->r);
	}
	long long m = (l + r) >> 1;
	node *nd = new node(cur->ln, cur->l, cur->r);
	if (l == r) {
		if (nd->ln.get(l) < v.get(l)) {
			nd->ln = v;
		}
		return nd;
	}
	if (nd->ln.get(l) < v.get(l))
		swap(nd->ln, v);
	if (nd->ln.get(m) > v.get(m)) {
		nd->r = modify(nd->r, m + 1, r, v);
		return nd;
	} else {
		swap(nd->ln, v);
		nd->l = modify(nd->l, l, m, v);
		return nd;
	}
}
long long get(node *cur, long long l, long long r, long long pos) {
	if (cur == empty) {
		return -1e18;
	}
```

```
	if (l == r)
		return cur->ln.get(pos);
	int m = (l + r) >> 1;
	long long ans = cur->ln.get(pos);
	if (pos <= m)
		ans = max(ans, get(cur->l, l, m, pos));
	else
		ans = max(ans, get(cur->r, m + 1, r, pos));
	return ans;
}
node *roots[N];
long long ans[N];
int main() {
	freopen("just.in", "rt", stdin);
	int tc;
	scanf("%d", &tc);
	line ln;
	roots[0] = empty;
	while (tc--) {
		int n;
		scanf("%d", &n);
		//roots[0] = empty;
		for (int i = 1; i <= n; ++i) {
			scanf("%lld %lld %lld", f + i, cnt + i, ld + i);
			int id = i - cnt[i] + 1;
			sum[i] = f[i] + sum[i - 1];
			ln.k = (-sum[i - 1]);
			ln.b = ans[i - 1];
			roots[i] = modify(roots[i - 1], -1e8, 1e8, ln);
			ans[i] = get(roots[max(i - cnt[i] + 1, 0ll)], -1e8, 1e8,
ld[i])
					+ sum[i] * ld[i];
		}
		printf("%lld\n", ans[n]);
	}
	return 0;
```

Circle Circle Intersection Area

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

using namespace std;
const int N = 3e8 + 1;
const long double PI = acos(-1), EPS = 1e-11;
long double areaOfIntersection(long double x0, long double y0, long double
r0, long double x1, long double y1, long double r1) {
	long double rr0 = r0 * r0;
	long double rr1 = r1 * r1;
	long double d = sqrt((x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0));

	// Circles do not overlap
	if (d + EPS >= r1 + r0) {
		return 0;
	}
```

```
		// Circle1 is completely inside circle0
	else if (d <= fabs(r0 - r1) + EPS && r0 + EPS >= r1) {
		// Return area of circle1
		return PI * rr1;
	}

		// Circle0 is completely inside circle1
	else if (d <= fabs(r0 - r1) + EPS && r0 <= r1 + EPS) {
		// Return area of circle0
		return PI * rr0;
	}

		// Circles partially overlap
	else {
		long double phi = (acos((rr0 + (d * d) - rr1) / (2.0 * r0 *
d))) * 2.0;
		long double theta = (acos((rr1 + (d * d) - rr0) / (2.0 * r1 *
d))) * 2.0;
		long double area1 = 0.5 * theta * rr1 - 0.5 * rr1 * sin(theta);
		long double area2 = 0.5 * phi * rr0 - 0.5 * rr0 * sin(phi);

		// Return area of intersection
		return area1 + area2;
	}
}
int main() {
#ifndef ONLINE_JUDGE
	freopen("input.in", "r", stdin);
#endif
	int t;
	scanf("%d", &t);
	for (int test = 1; test <= t; ++test) {
		int x0, y0, r0, x1, y1, r1;
		scanf("%d %d %d %d %d %d", &x0, &y0, &r0, &x1, &y1, &r1);

		printf("Case #%d: %.8lf\n", test, (double) (PI * r0 * r0 -
areaOfIntersection(x0, y0, r0, x1, y1, r1)));
	}
	return 0;
```

Circle Sweep

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

using namespace std;

typedef complex<double> point;
#define EPS 1e-9
#define X real()
#define Y imag()
#define vec(a, b) ((b)-(a))
#define angle(v) (atan2((v).Y,(v).X))
#define length(v) ((double)hypot((v).Y,(v).X))
```

```
#define lengthSqr(v) (dot(v,v))
#define dot(a, b) ((conj(a)*(b)).real())
#define cross(a, b) ((conj(a)*(b)).imag())
#define normalize(p) ((p)/length(p))
#define perp(a) (point(-(a).Y,(a).X))


/*
 *        u
 *       /|\
 *      / | \
 *    a/  |  \b
 *    /   |m  \
 *   /____|____\
 *  p  h   c-h  q
 */

int dcmp(const double &a, const double &b) {
    if (fabs(a - b) < EPS)
        return 0;
    return ((a > b) << 1) - 1;
}

int triangleThirdPoint(const point &p, const point &q, const double &a,
const double &b, point &u1, point &u2) {
    point pq = vec(p, q);
    double c = length(pq);
    double arr[] = {a, b, c};
    sort(arr, arr + 3);
    if (dcmp(arr[0] + arr[1], arr[2]) < 0)
        return false;
    //m^2=a^2-h^2
    //m^2=b^2-(c-h)^2
    //m^2=b^2-(c^2-2ch+h^2)
    //m^2=b^2-c^2+2ch-h^2
    //a^2-h^2=b^2-c^2+2ch-h^2
    //0=b^2-c^2+2ch-h^2-a^2+h^2
    //0=b^2-c^2+2ch-a^2
    //2ch=a^2-b^2+c^2
    //h=(a^2-b^2+c^2)/2c
    double h = (a * a - b * b + c * c) / (2.0 * c);
    double sq = a * a - h * h;
    if (!dcmp(sq, 0))sq = 0;
    double m = sqrt(sq);
    point npq = normalize(pq);
    point prp = perp(npq);
    u1 = p + (npq * h) + m * prp;
    u2 = p + (npq * h) - m * prp;
    return 1 + (dcmp(arr[0] + arr[1], arr[2]) != 0);
}

const int N = 100005;
point p[N];
double centers[N * 2];
int type[N * 2];
```

```
int sorted[N * 2];
int k, n;
point px(1, 0);

bool valid(double r) {
    int ssz = 0;
    int cnt = 0;

    for (int i = 0; i < n; ++i) {
        point in, out;
        if (triangleThirdPoint(point(0, 0), p[i], r, r, out, in)) {
            //cout << in << " " << out << "\n";
            centers[ssz] = angle(in);
            type[ssz] = 1;
            sorted[ssz] = ssz;
            ++ssz;
            centers[ssz] = angle(out);
            type[ssz] = -1;
            sorted[ssz] = ssz;
            ++ssz;
            if (dcmp(cross(px, in), 0) < 0 && dcmp(cross(px, out), 0) >= 0)
cnt++;
        }
    }

    sort(sorted, sorted + ssz, [](int a, int b) {
        double a1 = (centers[a]), a2 = (centers[b]);
        int cp = dcmp(a1, a2);
        if (cp) return cp < 0;
        return type[a] > type[b];
    });

    int mid = 0;
    while (1) {
        int i = sorted[mid];
        if (dcmp(centers[i], 0) >= 0) break;
        mid++;
    }
    rotate(sorted, sorted + mid, sorted + ssz);
    bool can = (cnt >= k);
    for (int i = 0; i < ssz && (!can); ++i) {
        cnt += type[sorted[i]];
        can |= (cnt >= k);
    }
    return !can;
}

int main() {
    int x, y;
    scanf("%d %d", &n, &k);
    for (int i = 0; i < n; ++i) {
        scanf("%d %d", &x, &y);
        p[i] = {x * 1.0, y * 1.0};
    }
    double s = 0, e = 1e9;
```

```
    for (double sz = (e - s) / 2; sz > 1e-9; sz *= .5) {
        if (valid(s + sz))s += sz;
    }

    if (dcmp(s, 1e9))printf("%.5lf\n", s);
    else puts("-1");
    return 0;
}
```

Line Equation Doubles

```
struct line {
    double a, b, c;

    line(const point &p, const point &q) {
        a = p.Y - q.Y;
        b = q.X - p.X;
        c = -a * p.X - b * p.Y;
        double z = sqrt(a * a + b * b);
        a /= z, b /= z, c /= z;
   }
}
```

Line Equation

```
map<pair<int, int>, set<int> > m;

void add_line(int x1, int y1, int x2, int y2) {
    int dx = x1 - x2, dy = y1 - y2;
    int A = dy;
    int B = -dx;
    int g = __gcd(A, B);
    A /= g, B /= g;
    int C = -(A * x1 + B * y1);
    if (A < 0 || A == 0 && B < 0) {
        A *= -1, B *= -1, C *= -1;
    }
    m[{A, B}].insert(C);
```

Triangle Third Point

```
/*
 *         u
 *        /|\
 *       / | \
 *    a/   |  \b
 *     /   |m  \
 *    /____|____\
 *   p  h    c-h  q
 */
```

```
int dcmp(const double &a, const double &b) {
    if (fabs(a - b) < EPS)
        return 0;
    return ((a > b) << 1) - 1;
}

int triangleThirdPoint(const point &p, const point &q, const double &a,
                       const double &b, point &u1, point &u2) {
    point pq = vec(p, q);
    double c = length(pq);
    double arr[] = { a, b, c };
    sort(arr, arr + 3);
    if (dcmp(arr[0] + arr[1], arr[2]) < 0)
        return false;

    //m^2=a^2-h^2
    //m^2=b^2-(c-h)^2
    //m^2=b^2-(c^2-2ch+h^2)
    //m^2=b^2-c^2+2ch-h^2
    //a^2-h^2=b^2-c^2+2ch-h^2
    //0=b^2-c^2+2ch-h^2-a^2+h^2
    //0=b^2-c^2+2ch-a^2
    //2ch=a^2-b^2+c^2
    //h=(a^2-b^2+c^2)/2c
    double h = (a * a - b * b + c * c) / (2.0 * c);
    double sq=a * a - h * h;
    if(!dcmp(sq,0))sq=0;
    double m = sqrt(sq);
    point npq = normalize(pq);
    point prp = perp(npq);
    u1 = p + (npq * h) + m * prp;
    u2 = p + (npq * h) - m * prp;
    return 1 + (dcmp(arr[0] + arr[1], arr[2]) != 0);
```

Circle Rectangle Intersection

```
typedef long long ll;
typedef long double ld;

const ld eps = 1e-12;
const ld pi = acos(-1);

ld dist(ld x1, ld y1, ld x2, ld y2){
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
bool right(ld cx, ld cy, ld x1, ld y1, ld x2, ld y2, ld angle){
    angle = (angle * pi) / 180.0;
    ld newx = (x1 - cx) * cos(angle) - (y1 - cy) * sin(angle) + cx;
    ld newy = (x1 - cx) * sin(angle) + (y1 - cy) * cos(angle) + cy;
    if(newx > x2 || fabs(newx - x2) < eps)
        return true;
    else
        return false;
}
```

```
int main(){
    int t;
    cin >> t;
    for(int tt = 1; tt <= t; tt++){
        ld x1, y1, r;
        cin >> x1 >> y1 >> r;
        ld x2, y2, x3, y3;
        cin >> x2 >> y2 >> x3 >> y3;
        ld xi1, yi1, xi2, yi2;
        ld xin = x2, yin = y3;
        ld st = 0, en = 1e4;
        for(int i = 0; i < 100; i++){
            ld mid = (st + en) / 2;
            ld newx = xin, newy = yin - mid;
            if(fabs(dist(x1, y1, newx, newy) - r) < eps || dist(x1, y1,
newx, newy) < r){
                xi1 = newx, yi1 = newy;
                st = mid;
            }
            else
                en = mid;
        }
        st = 0, en = 1e4;
        for(int i = 0; i < 100; i++){
            ld mid = (st + en) / 2;
            ld newx = xin + mid, newy = yin;
            if(fabs(dist(x1, y1, newx, newy) - r) < eps || dist(x1, y1,
newx, newy) < r){
                xi2 = newx, yi2 = newy;
                st = mid;
            }
            else
                en = mid;
        }
        st = 0, en = 180;
        ld secangle = asin((dist(xi1, yi1, xi2, yi2) * 0.5) / dist(x1, y1,
xi1, yi1));
        secangle = (secangle * 180.0) / pi * 2;
        ld sol = 0.5 * dist(xin, yin, xi1, yi1) * dist(xin, yin, xi2, yi2) +
(pi * r * r * (secangle / 360.0) - 0.5 * r * r * sin((secangle * pi) /
180));
        cout << "Case " << tt << ": ";
        cout << fixed << setprecision(5) << sol << '\n';
    }
```