Team notebook

December 23, 2019

Contents

1 DP

1.1 Convex Hull DP2

```
struct Line { // gives max value of x
    ll k, m;
    mutable ll p;
    bool operator<(const Line& o) const {
        return k < o.k;
    }
    bool operator<(const ll &x) const{
        return p < x;
    }
};
struct LineContainer : multiset<Line, less<>> {
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b){
        return a / b - ((a ^ b) < 0 && a % b);
    }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};

LineContainer lc;
```

2 Data Structures

2.1 2D and Persistent Segment Trees

```
struct D2segtree{
    vector<segtree> seg;
    D2segtree(ll n, ll m, vector<vector<ll>> &ar){
        seg.resize(4*n+4);
        for(ll i=0; i<4*n+4; i++){
            seg[i].seg.resize(4*m+4);
        }
        vector<segtree>temp(n+1);
        for(ll i=1;i<=n;i++){
            temp[i].seg.resize(4*m+4);
            temp[i].build(ar[i], 1, 1, m);
        }
        build(temp, 1, 1, n, m);
    }
    segtree merge(ll m, segtree a, segtree b){
        segtree k(m);
        for(ll i=0; i<4*m+4; i++)
            k.seg[i].sum=a.seg[i].sum+b.seg[i].sum;
        return k;
    }
    void build(vector<segtree> &ar,ll pos,ll l,ll r,ll m){
        if(l==r){
            seg[pos]=ar[l];
            return;
        }
        ll b=(l+r)/2;
        build(ar,2*pos,l,b,m);
        build(ar,2*pos+1,b+1,r,m);
        seg[pos]=merge(m,seg[2*pos],seg[2*pos+1]);
        return;
    }
    ll query(ll n,ll m,ll x1,ll y1,ll x2,ll y2){
        return get(1, 1, n, m, y1, y2).get(1, 1, m, x1, x2).sum;
    }
    segtree get(ll pos,ll l,ll r,ll m,ll st,ll en){
        if(l>en || r<st || l>r){
            segtree k(m);
```

```
            return k;
        }
        if(st<=l && en>=r){
            return seg[pos];
        }
        ll b=(l+r)/2;
        return merge(m,get(2*pos,l,b,m,st,en),get(2*pos+1,b+1,r,m,st,en));
    }
    void update(ll pos,ll l,ll r,ll m,ll x,ll y,ll val){
        if(l==r){
            seg[pos].update(1,1,m,x,val);
            return;
        }
        ll b=(l+r)/2;
        if(y<=b){
            update(2*pos,l,b,m,x,y,val);
        }
        else{
            update(2*pos+1,b+1,r,m,x,y,val);
        }
        seg[pos]=merge(m,seg[2*pos],seg[2*pos+1]);
        return;
    }
};
struct node{
    ll val;
    node *l, *r;
    node(){
        l=r=NULL;
    }
    node(node *left, node *right, ll v){
        l=left;
        r=right;
        val=v;
    }
};
struct psegtree{
    void build(vector<ll>&ar, node *root, ll l, ll r){
        if(l==r){
            root->val=ar[l];
            return;
        }
        ll b=(l+r)/2;
        root->l=new node(NULL, NULL, 0);
        root->r=new node(NULL, NULL, 0);
        build(ar,root->l, l, b);
        build(ar,root->r, b+1, r);
        root->val=root->l->val+root->r->val;
    }
    void upgrade(node *pre,node *cur,ll l,ll r,ll idx,ll val){
        if(l==r){
            cur->val=val;
            return;
        }
        ll b=(l+r)/2;
        if(idx<=b){
            cur->r = pre->r;
            cur->l = new node(NULL, NULL, 0);
            upgrade(pre->l,cur->l,l,b,idx,val);
        }
        else{
            cur->l=pre->l;
            cur->r=new node(NULL, NULL, 0);
            upgrade(pre->r,cur->r,b+1,r,idx,val);
        }
        cur->val=cur->l->val+cur->r->val;
    }
    ll get(node *root,ll l,ll r,ll st,ll en){
        if(l>r || en<l || st>r){
            return 0;
        }
        if(l>=st && r<=en){
            return root->val;
        }
        ll b=(l+r)/2;
        return get(root->l,l,b,st,en)+get(root->r,b+1,r,st,en);
    }
};
```

2.2 BIT

```
/*1 base indexing*/
/* Problem Statement:
Given a sequence of n numbers a1, a2, ..., an and a number of k- queries.
A k-query is a triple (i, j, k) (1<=i<=j<=n). For each k-query
(i, j, k), you have to return the number of elements greater than k in
the subsequence ai, ai+1, ..., aj. */
struct M
```

```
{
	ll key;
	ll key2;
	ll key3;
	ll key4;
};
bool cmp(struct M a, struct M b)
{
	if(a.key==b.key) return b.key4<=a.key4;
	return (a.key > b.key);
}
bool cmp2(struct M a,struct M b)
{
	return a.key4<b.key4;
}
ll bit[30002];
ll update(ll idx,ll n)
{
	while(idx<=n)
	{
		bit[idx]+=1;
		idx=idx+(idx&(-idx    ));
	}
}
ll query(ll idx)
{
	ll sum=0;
	while(idx>0)
	{
		sum+=bit[idx];
		idx=idx-(idx&(-idx));
	}
	return sum;
}
inline int in(){
	int N=0;
	register char c=getchar();
	while( c < 48 || c > 57 ){
		c=getchar();
	}
	while(c>47 && c< 58){
		N = (N << 3) + (N << 1) + (c - 48);
		c=getchar();
	}
	return N;
}
struct M Ssp[230000];
int main()
{
	ll n;n=in();
	ll q;
	for (int i = 0; i < n; ++i)
	{
		ll a;
		a=in();
		Ssp[i].key=a;
		Ssp[i].key2=0;
		Ssp[i].key4=i;
		Ssp[i].key3=0;
	}
	q=in();
	for (int i = 0; i < q; ++i)
	{
		ll l,r,k;
		l=in(); r=in(); q=in();
		Ssp[i+n].key=k;
		Ssp[i+n].key2=l;
		Ssp[i+n].key3=r;
		Ssp[i+n].key4=i+n;
	}
	sort(Ssp, Ssp+n+q, cmp);
	for (int i = 0; i < n+q; ++i)
	{
		if(!Ssp[i].key2)
		{
			update(Ssp[i].key4+1,n);
		}
		else
		{
			Ssp[i].key=query(Ssp[i].key3)-query(Ssp[i].key2-1);
		}
	}
	sort(Ssp, Ssp+n+q, cmp2);
	for (int i = 0; i < n+q; ++i)
	{
		if(Ssp[i].key2)
		{
			printf("%lld\n",Ssp[i].key);
		}
	}
```

```
}
```

2.3 Centroid Decomposition

```
// E. Xenia and Tree, Codeforces
#define ln 20
#define N 100001
#define INF 1e9
ll n;
vector<vector<ll>>ar(N);
ll lev[N];
ll pa[N][ln];
ll centroidMarked[N]={0};
ll sub[N];
ll par[N];
ll ans[N];
// -------------dist(u,v)-----------------
void dfs(ll u,ll p,ll l){
    pa[u][0]=p;
    lev[u]=l;
    for(auto i:ar[u]){
        if(i!=p)
            dfs(i,u,l+1);
    }
}
ll lca(ll u,ll v){
    if(lev[u]<lev[v]) swap(u,v);
    ll log;
    for(log=1;(1<<log)<=lev[u];log++);
    log--;
    for(ll i=log;i>=0;i--){
        if(lev[u]-(1<<i)>=lev[v])
            u=pa[u][i];
    }
    if(u==v) return u;
    for(ll i=log;i>=0;i--){
        if(pa[u][i]!=-1 && pa[u][i]!=pa[v][i])
            u=pa[u][i],v=pa[v][i];
    }
    return pa[u][0];
}
ll dist(ll u,ll v){
    return lev[u]+lev[v]-2*lev[lca(u,v)];
}
// ------------decompose--------------
ll nn;
void dfs1(ll u,ll p){
    nn++;
    sub[u]=1;
    for(auto i:ar[u]){
        if(i!=p && !centroidMarked[i]){
            dfs1(i,u);
            sub[u]+=sub[i];
        }
    }
}
ll dfs2(ll u,ll p){
    for(auto i:ar[u]){
        if(i!=p && !centroidMarked[i] && sub[i]>nn/2)
            return dfs2(i,u);
    }
    return u;
}
void decompose(ll u,ll p){
    nn=0;
    dfs1(u,p);
    ll centroid=dfs2(u,p);
    centroidMarked[centroid]=1;
    par[centroid]=p;
    for(auto i:ar[centroid]){
        if(!centroidMarked[i]){
            decompose(i,centroid);
        }
    }
}
// --------------query--------------
void update(ll u){
    ll x=u;
    while(x!=-1){
        ans[x]=min(ans[x],dist(u,x));
        x=par[x];
    }
}
ll query(ll u){
    ll x=u;
    ll an=INF;
    while(x!=-1){
        an=min(an,ans[x]+dist(u,x));
```

```
        x=par[x];
    }
    return an;
}

int main(){
    ll m;
    cin>>n>>m;
    for(ll i=1,u,v;i<n;i++){
        cin>>u>>v;
        ar[u].pb(v);
        ar[v].pb(u);
    }
    for(ll i=0;i<=n;i++){
        for(ll j=0;j<ln;j++)
            pa[i][j]=-1;
    }
    dfs(1,-1,0);
    for(ll i=1;i<ln;i++){
        for(ll j=1;j<=n;j++)
            if(pa[j][i-1]!=-1)
                pa[j][i]=pa[pa[j][i-1]][i-1];
    }
    decompose(1,-1);
    for(ll i=0;i<=n;i++){
        ans[i]=INF;
    }
    update(1);
    while(m--){
        ll t,v;
        cin>>t;
        if(t==2){
            cin>>v;
            cout << query(v) << "\n";
        }
        else{
            cin>>v;
            update(v);
        }
    }
}
```

2.4 Merge Sort Tree

```
// Merge Sort Tree to calculate kth smallest number in a range
// Works for online queries // Problem Codeforces 1262D2
bool cmp(pll a, pll b){
    if(a.ff == b.ff){
        return a.ss < b.ss;
    }
    return a.ff > b.ff;
}
ll kd[30][L] , a[L] , pos[L] , Real[L];
void init(ll d,ll b,ll e){
    if(b == e){
        kd[d][b] = pos[b];
        return;
    }
    ll m = (b + e) >> 1;
    init(d + 1,b,m);
    init(d + 1,m+1,e);
    ll i = b , j = m + 1;
    ll ptr = 0;
    while(i <= m && j <= e){
        if(kd[d + 1][i] < kd[d + 1][j]){
            kd[d][b + (ptr++)] = kd[d + 1][i++];
        }else{
            kd[d][b + (ptr++)] = kd[d + 1][j++];
        }
    }
    while(i <= m) kd[d][b + (ptr++)] = kd[d + 1][i++];
    while(j <= e) kd[d][b + (ptr++)] = kd[d + 1][j++];
}
inline ll find(ll d,ll b,ll e,ll x1,ll x2){
    return upper_bound(kd[d] + b,kd[d] + e + 1,x2) - lower_bound(kd[d] +
        b,kd[d] + e + 1,x1);
}
ll get(ll n,ll x1,ll x2,ll k){
    ll d = 0 , b = 1 , e = n;
    while(b != e){
        ll lll = find(d + 1,b,(b+e)/2,x1,x2);
        ll mm = ((b + e) >> 1LL);
        if(lll >= k){
            e = mm;
        }else{
            b = mm + 1;
            k -= lll;
        }
```

```
        ++d;
    }
    return b;
}
ll copy_it[L];
int main(){
    ll n;
    cin >> n;
    vector <ll> a(n, 0);
    vector <pll> pq;
    for(ll i=0; i<n; i++){
        ll t;
        cin >> t;
        copy_it[i] = t;
        pq.pb(mp(t, i));
    }
    sort(all(pq), cmp);
    vector <ll> vals;
    for(ll i=1; i<=n; i++){
        a[i] = pq[i-1].ss;
        vals.pb(a[i]);
    }
    sort(all(vals));
    for(ll i=1; i<=n; i++){
        ll old = a[i];
        a[i] = lower_bound(all(vals), a[i]) - vals.begin() + 1;
        pos[a[i]] = i;
        Real[a[i]] = old;
    }
    init(0, 1, n);
    ll m;
    cin >> m;
    while(m--){
        ll k, which;
        cin >> k >> which;
        cout << copy_it[Real[get(n, 1, k, which)]] << endl;
    }
}
```

2.5 Segment Tree with Lazy Propagation

```
// SPOJ CNTPRIME // 1-based indexing
vector <bool> isPrime(M, true);
void sieve(){
    isPrime[0] = false;isPrime[1] = false;
    for(ll i=2; i<M; i++){
        if(isPrime[i]){
            for(ll j=2*i; j<M; j+=i)
                isPrime[j] = false;
        }
    }
}
ll a[L]; ll seg[4*L]; ll lazy[4*L];
ll merge(ll a, ll b){
    return (a+b);
}
void build(ll pos, ll tl, ll tr){
    if(tl == tr){
        if(isPrime[a[tl]])
            seg[pos] = 1;
        return;
    }
    ll mid = tl + (tr-tl)/2;
    build(2*pos, tl, mid);
    build(2*pos+1, mid+1, tr);
    seg[pos] = merge(seg[2*pos], seg[2*pos+1]);
}

void update(ll pos, ll tl, ll tr, ll l, ll r, ll val){
    if(lazy[pos] != 0){
        if(isPrime[lazy[pos]])
            seg[pos] = tr-tl+1;
        else
            seg[pos] = 0;
        if(tl != tr){
            lazy[2*pos] = lazy[pos];
            lazy[2*pos+1] = lazy[pos];
        }
        lazy[pos] = 0;
    }
    if(tl > r || tr < l)
        return;
    if(tl >= l && tr <= r){
        if(isPrime[val])
            seg[pos] = tr-tl+1;
        else
            seg[pos] = 0;
        if(tl != tr){
```

```
            lazy[2*pos] = val;
            lazy[2*pos+1] = val;
        }
        lazy[pos] = 0;
        return;
    }
    ll mid = tl + (tr-tl)/2;
    update(2*pos, tl, mid, l, r, val);
    update(2*pos+1, mid+1, tr, l, r, val);
    seg[pos] = merge(seg[2*pos], seg[2*pos+1]);
}
ll query(ll pos, ll tl, ll tr, ll l, ll r){
    if(lazy[pos] != 0)
        // same as update
    if(l > tr || r < tl)
        return 0;
    if(tl >= l && tr <= r)
        return seg[pos];
    ll mid = tl + (tr-tl)/2;
    return merge(query(2*pos, tl, mid, l, r), query(2*pos+1, mid+1, tr,
        l, r));
}
```

2.6 Segment Tree

```
// SPOJ GSS3 // 1-based indexing
typedef struct node{
    ll ans, pref, suff, sum;
} node;
ll a[L];
node seg[4*L];
node merge(node a, node b){
    node x;
    x.ans = max(a.suff + b.pref, max(a.ans, b.ans));
    x.pref = max(a.pref, a.sum + b.pref);
    x.suff = max(b.suff, a.suff + b.sum);
    x.sum = a.sum + b.sum;
    return x;
}
void build(ll pos, ll tl, ll tr){
    if(tl == tr){
        seg[pos].ans = a[tl];
        seg[pos].pref = a[tl];
        seg[pos].suff = a[tl];
        seg[pos].sum = a[tl];
        return;
    }
    ll mid = tl + (tr-tl)/2;
    build(2*pos, tl, mid);
    build(2*pos+1, mid+1, tr);
    seg[pos] = merge(seg[2*pos], seg[2*pos+1]);
}
void update(ll pos, ll tl, ll tr, ll idx, ll val){
    if(tl == tr){
        seg[pos].sum = val;
        seg[pos].ans = val;
        seg[pos].pref = val;
        seg[pos].suff = val;
        return;
    }
    ll mid = tl + (tr - tl)/2;
    if(tl <= idx && idx <= mid){
        update(2*pos, tl, mid, idx, val);
    }
    else{
        update(2*pos+1, mid+1, tr, idx, val);
    }
    seg[pos] = merge(seg[2*pos], seg[2*pos+1]);
}
node query(ll pos, ll tl, ll tr, ll l, ll r){
    if(tl > r || tr < l){
        node x;
        x.sum = 0;
        x.ans = -1e15;
        x.pref = -1e15;
        x.suff = -1e15;
        return x;
    }
    if(tl >= l && tr <= r){
        return seg[pos];
    }
    ll mid = tl + (tr-tl)/2;
    return merge(query(2*pos, tl, mid, l, r), query(2*pos+1, mid+1, tr,
        l, r));
}
```

2.7 Trie

```
struct node{
    vector<ll>val;
    vector<node*>pt;
    node(){}
    node(ll c){
        val.resize(c,0);
        pt.resize(c,NULL);
    }
};
struct trie{
    ll chr;
    trie(ll c){
        chr=c;
    }
    void add(node *root, string &s){
        node *cur=root;
        for(auto x:s){
            if(cur->val[x-'a']==0){
                cur->val[x-'a']=1;
                cur->pt[x-'a']=new node(chr);
            }
            cur=cur->pt[x-'a'];
        }
    }
    ll find(node *root, string &s, ll x){
        if(s[x]=='\0')
            return 1;
        if(root->val[s[x]-'a']==0){
            return 0;
        }
        else{
            return find(root->pt[s[x]-'a'],s,x+1);
        }
    }
};
int main(){
    trie obj(26);
    node *root=new node(26);
    ll q;
    cin>>q;
    while(q--){
        ll a;
        cin>>a;
        if(a==1){
            string s;
            cin>>s;
            cout << obj.find(root,s,0) << "\n";
        }
        else{
            string s;
            cin>>s;
            obj.add(root,s);
        }
    }
}
```

2.8 Wavelet Tree

```
ll MAX=1e6;
struct wavelet_tree{
        ll lo,hi;
        wavelet_tree *l,*r;
        vector<ll>b;
        wavelet_tree(ll *from,ll *to,ll x,ll y){
                lo = x,hi = y;
                if(lo == hi || from >= to)return;
                ll mid = (lo+hi)/2;
                auto f = [mid](ll x){
                        return x <= mid;
                };
                b.reserve(to-from+1);
                b.push_back(0);
                for(auto it = from; it!=to; it++)
                        b.push_back(b.back() + f(*it));

                auto pivot = stable_partition(from, to, f);
                l = new wavelet_tree(from, pivot, lo, mid);
                r = new wavelet_tree(pivot, to, mid + 1, hi);
        }
        // kth smallest element in [l, r]
        ll kth(ll le,ll ri,ll k){
                if(le > ri) return 0;
                if(lo == hi) return lo;
                ll inLeft = b[ri] - b[le-1];
                ll lb = b[le-1]; //amt of nos in first (l-1) nos that go
                    in left
```

```
            ll rb = b[ri]; //amt of nos in first (r) nos that go in
                left
            if(k <= inLeft) return this->l->kth(lb+1, rb , k);
            return this->r->kth(le-lb, ri-rb, k-inLeft);
        }
        // count of nos in [l, r] less than or equal to k
        ll LTE(ll le,ll ri,ll k){
            if(le>ri || k < this->lo) return 0;
            if(this->hi <= k) return ri-le+1;
            ll lb = b[le-1],rb = b[ri];
            return this->l->LTE(lb+1,rb,k) +
                this->r->LTE(le-lb,ri-rb,k);
        }
        //count of nos in [l, r] equal to k
        int count(ll le,ll ri,ll k) {
            if(le > ri or k < lo or k > hi) return 0;
            if(lo == hi) return ri - le + 1;
            int lb = b[le-1], rb = b[ri], mid = (lo+hi)/2;
            if(k <= mid) return this->l->count(lb+1, rb, k);
            return this->r->count(le-lb, ri-rb, k);
        }
};
int main(){
        ll n; cin>>n;
        ll ar[n+1];
        wavelet_tree obj(ar+1,ar+n+1,1,MAX);
}
```

3 Graphs

3.1 Basic Graph Algorithms

```
vector<ll>path(N, INF); // Dijkstras
vector<ll>visit(N, 0);
void dijk(auto &ar, ll x){
        priority_queue<pair<ll,ll>, vector<pair<ll,ll>>,
            greater<pair<ll,ll>>>pq;
        pq.push(make_pair(x, 0));
        path[x] = 0;
        while(!pq.empty()){
            auto p=pq.top(); pq.pop();
            if(visit[p.first] == 1) continue;
            visit[p.first] = 1;
            for(auto i:ar[p.first]){
                if(visit[i.first] == 1){
                    continue;
                }
                if(path[i.first] > path[p.first] + i.second){
                    path[i.first] = path[p.first] + i.second;
                    pq.push(make_pair(i.first, path[i.first]));
                }
            }
        }
}
struct edge{ // Bellman Ford
        ll u,v,w;
};
vector<ll>path(N, INF);
vector<ll>par(N, 0);
ll n;
ll bellman_ford(auto &ar, ll x){
        ll m = sz(ar);
        path[x] = 0;
        for(ll i=1; i < n; i++){
            for(ll j = 0; j < m; j++){
                if(path[ar[j].v] > path[ar[j].u] + ar[j].w){
                    path[ar[j].v] = path[ar[j].u] + ar[j].w;
                    par[ar[j].v] = ar[j].u;
                }
            }
        }
        for(ll i = 0; i < m; i++){
            if(ar[i].v > ar[i].u + ar[i].w)
                return 0;
        }
        return 1;
}
ll graph[N][N]; // Floyd Warshall
ll n;
void floydWarshal(){
        for(ll k = 1; k <= n; k++){
            for(ll i = 1; i <= n; i++){
                for(ll j = 1; j <= n; j++){
                    if(graph[i][j] > graph[i][k] + graph[k][j]){
                        graph[i][j] = graph[i][k] +
                            graph[k][j];
                    }
```

```
                        }
                }
        }
}
vector<ll>visit(N, 0); // Shortest Path in DAG
stack<ll>st;
void st_dfs(auto &ar, ll x){
        visit[x] = 1;
        for(auto i:ar[x]){
                if(visit[i.first] == 0){
                        st_dfs(ar, i.first);
                }
        }
        st.push(x);
}
void toposort(auto &ar){
        ll n = sz(ar)-1;
        for(ll i=1; i <= n; i++){
                if(visit[i] == 0)
                        st_dfs(ar, i);
        }
}
vector<ll>path(N, INF);
void shortpathDAG(auto &ar, ll x){
        toposort(ar);
        path[x] = 0;
        while(!st.empty()){
                auto t = st.top(); st.pop();
                if(t == x){
                        st.push(x);
                        break;
                }
        }
        while(!st.empty()){
                auto t = st.top(); st.pop();
                for(auto i:ar[t]){
                        if(path[i.first] > path[t] + i.second){
                                path[i.first] = path[t] + i.second;
                        }
                }
        }
}
vector<ll>path(N, INF); // Johnson's Algorithm
vector<ll>par(N, 0);
ll n;
ll bellman_ford(auto &ar, ll x){
        path[x]=0;
        for(ll i = 0; i < n; i++){
                for(ll j = 0; j <= n; j++){
                        for(auto it:ar[j]){
                                if(path[it.first] > path[j] + it.second){
                                        path[it.first] = path[j] + it.second;
                                        par[it.first] = j;
                                }
                        }
                }
        }
        for(ll j = 0; j <= n; j++){
                for(auto it:ar[j]){
                        if(path[it.first] > path[j] + it.second){
                                return 0;
                        }
                }
        }
        return 1;
}
ll spath[N][N];
vector<ll>visit(N, 0);
void dijk(auto &ar, ll x){
        priority_queue<pair<ll,ll>, vector<pair<ll,ll>>,
            greater<pair<ll,ll>>>pq;
        pq.push(make_pair(x, 0));
        spath[x][x]=0;
        while(!pq.empty()){
                auto p = pq.top(); pq.pop();
                if(visit[p.first] == 1) continue;
                visit[p.first] = 1;
                for(auto i:ar[p.first]){
                        if(visit[i.first] == 1){
                                continue;
                        }
                        if(spath[x][i.first] > spath[x][p.first] +
                            i.second){
                                spath[x][i.first] = spath[x][p.first] +
                                    i.second;
                                pq.push(make_pair(i.first,
                                    spath[x][i.first]));
                        }
                }
        }
```

```
}
ll jhonson(auto &ar){
	for(ll i = 1; i <= n; i++){
		ar[0].pb(make_pair(i, 0));
	}
	if(!bellman_ford(ar, 0)){
		return 0;
	}
	for(ll i = 1; i <= n; i++){
		for(auto &j:ar[i]){
			j.second += path[i]-path[j.first];
		}
	}
	for(ll i = 1; i <= n; i++){
		for(ll j = 1; j <= n; j++){
			visit[j] = 0;
			spath[i][j] = INF;
		}
		dijk(ar, i);
	}
	return 1;
}
```

3.2 Dinics EV^2*PushRelabel*

```
/*	Push Relabel O(n^3) implimentation using FIFO method to chose push
    vertex.
	This uses gapRelabel heuristic to fasten the process even further.
	    If only
	the maxFlow value is required then the algo can be stopped as soon
	    as the
	gap relabel method is called. However, to get the actual flow
	    values in the
	edges, we need to let the algo terminate itself.
	This implementation assumes zero based vertex indexing. Edges to
	    the graph
	can be added using the addEdge method only. capacity for residual
	    edges is
	set to be zero. To get the actual flow values iterate through the
	    edges and
	check for flow for an edge with cap > 0.
	This implimentaion is superior over dinic's for graphs where graph
	    is dense
	locally at some places and mostly sparse. For randomly generated
	    graphs, this
	implimentation gives results within seconds for n = 10000 nodes, m
	    = 1000000
	edges. */
typedef ll fType;
struct edge
{
	ll from, to;
	fType cap, flow;
	edge(ll from, ll to, fType cap, fType flow = 0) : from(from),
	    to(to), cap(cap), flow(flow) {}
};
struct PushRelabel
{
	ll N;
	vector<edge> edges;
	vector<vector<ll> > G;
	vector<ll> h, inQ, count;
	vector<fType> excess;
	queue<ll> Q;
	PushRelabel(ll N) : N(N), count(N<<1), G(N), h(N), inQ(N),
	    excess(N) {}
	void addEdge(ll from, ll to, ll cap) {
		G[from].push_back(edges.size());
		edges.push_back(edge(from, to, cap));
		G[to].push_back(edges.size());
		edges.push_back(edge(to, from, 0));
	}
	void enQueue(ll u) {
		if(!inQ[u] && excess[u] > 0) Q.push(u), inQ[u] = true;
	}
	void Push(ll edgeIdx) {
		edge & e = edges[edgeIdx];
		ll toPush = min<fType>(e.cap - e.flow, excess[e.from]);
		if(toPush > 0 && h[e.from] > h[e.to]) {
			e.flow += toPush;
			excess[e.to] += toPush;
			excess[e.from] -= toPush;
			edges[edgeIdx^1].flow -= toPush;
			enQueue(e.to);
		}
	}
	void Relabel(ll u) {
		count[h[u]] -= 1; h[u] = 2*N-2;
```

```
		for (ll i = 0; i < G[u].size(); ++i) {
			edge & e = edges[G[u][i]];
			if(e.cap > e.flow) h[u] = min(h[u], h[e.to]);
		}
		count[++h[u]] += 1;
	}
	void gapRelabel(ll height) {
		for (ll u = 0; u < N; ++u) if(h[u] >= height && h[u] < N) {
			count[h[u]] -= 1;
			count[h[u] = N] += 1;
			enQueue(u);
		}
	}
	void Discharge(ll u) {
		for (ll i = 0; excess[u] > 0 && i < G[u].size(); ++i) {
			Push(G[u][i]);
		}
		if(excess[u] > 0) {
			if(h[u] < N && count[h[u]] < 2) gapRelabel(h[u]);
			else Relabel(u);
		}
		else if(!Q.empty()) { // dequeue
			Q.pop();
			inQ[u] = false;
		}
	}
	fType getFlow(ll src, ll snk) {
		h[src] = N; inQ[src] = inQ[snk] = true;
		count[0] = N - (count[N] = 1);
		for (ll i = 0; i < G[src].size(); ++i) {
			excess[src] += edges[G[src][i]].cap;
			Push(G[src][i]);
		}
		while (!Q.empty()) {
			Discharge(Q.front());
		}
		return excess[snk];
	}
};
int main()
{
	ll n, m;
	cin >> n >> m;
	PushRelabel df(n);
	while(m--) {
		ll x, y, c;
		cin >>x >> y >> c;
		--x, --y;
		if(x != y){
			df.addEdge(x, y, c);
			df.addEdge(y, x, c);
		}
	}
	cout << df.getFlow(0, n-1) << "\n";
	return 0;
}
```

3.3 Dinics EV^2

```
const ll N=1e4+5,inf=1e10;
struct edge
{
	int a,b;
	ll c,f ;
	edge(int u,int v,ll cap):a(u),b(v),c(cap),f(0){}
};
struct flows
{
	const static ll inf = 1e18 ;
	int level[N], Dptr[N], s, t ;
	queue<int> Q; vector<edge> E,E2; vll ad[N] ;
	void add(int a,int b,int c)
	{
		if(a==b)return ;
		ad[a].pb(E.size()),E.pb(edge(a,b,c)) ;
		ad[b].pb(E.size()),E.pb(edge(b,a,0)) ;
	}
	bool bfs(void)
	{
		memset(level,0,sizeof(level));
		Q.push(s);
		level[s]=1;
		while(!Q.empty())
		{
			int sz=Q.size(),v ;
			while(sz--)
			{
				v = Q.front();Q.pop() ;
```

```
                        for(auto &e:ad[v])
                        {
                                if(!level[E[e].b]&&E[e].f<E[e].c)
                                {
                                        level[E[e].b]=level[v]+1;
                                        Q.push(E[e].b);
                                }
                        }
                }
        }
        return level[t]>0 ;
}
ll dfs(int x,ll flow)
{
        if(!flow) return 0;
        if(x==t) return flow ;
        for(int &pt=Dptr[x];pt<ad[x].size();++pt)
        {
                int e=ad[x][pt];
                if(level[E[e].b]==level[x]+1)
                {
                        if(ll
                            pushed=dfs(E[e].b,min(flow,E[e].c-E[e].f)))
                        {
                                E[e].f+=pushed ;
                                E[e^1].f -= pushed;
                                return pushed ;
                        }
                }
        }
        return 0 ;
}
ll dinic(void)
{
        ll flow=0 ;
        while(bfs())
        {
                memset(Dptr,0,sizeof(Dptr));
                while(ll pushed=dfs(s,inf)) flow+=pushed;
        }
        return flow ;
}
void reset(void)
{
        for(auto &e:E)e.f=0;
        }
};
int main()
{
        ll n,m;
        cin >> n >> m;
        flows F;
        lp(i,0,m)
        {
                ll a,b,w;
                cin >> a >> b >> w;
                F.add(a,b,w);
        }
        cin >> F.s>> F.t;
        cout<<F.dinic()<<endl;
        return 0;
}
```

3.4 Ford Fulkerson

```
const ll inf=1e10,N=1005;
ll flow[N][N],cap[N][N],p[N],timer,ans,vis[N];
vll G[N];
bool bfs(ll st,ll end)
{
        queue<ll> q;
        q.push(st);
        while(!q.empty())
        {
                ll a=q.front();
                q.pop();
                if(a==end)
                {
                        return true;
                }
                lp(i,0,G[a].size())
                {
                        ll u=G[a][i];
                        if(vis[u]!=timer && cap[a][u] > flow[a][u])
                        {
                                p[u] = a;
                                vis[u]=timer;
                                q.push(u);
```

```
                }
            }
        }
        return false;
}
int main()
{
        ll n,m;
        cin >> n>> m;
        lp(i,0,m)
        {
            ll a,b,w;
            cin >> a >> b >> w;
            G[a].pb(b);
            G[b].pb(a);
            cap[a][b]=w;
        }
        ll st,end;
        cin >> st >> end;
        ll x=inf;
        timer++;
        while(bfs(st,end))
        {
            cout<<endl;
            timer++;
            ll mn=inf;
            ll i=end;
            while(i!=st)
            {
                cout<<i<<" ";
                mn=min(mn,cap[p[i]][i]-flow[p[i]][i]);
                i=p[i];
            }
            cout<<endl;
            i=end;
            while(i!=st)
            {
                flow[p[i]][i]+=mn;
                flow[i][p[i]]-=mn;
                i=p[i];
            }
            cout<<mn<<endl;
            ans+=mn;
            memset(p,0,sizeof p);
        }
        cout<<ans<<endl;
        return 0;
}
```

3.5 Heavy Light Decomposition

```
// QTREE SPOJ
struct node{
    ll depth,par,size,chain,posInBase;
};
#define ln 16
#define N 100001
ll n,chainNo,ptr;
vector<vector<pair<ll,pair<ll,ll>>>>ar(N);
node nd[N];
ll chainHead[N],otherEnd[N];
vector<ll> baseArray(N);
ll pa[N][ln];
struct segtree{
    struct node{
        ll sum;
    };
    vector<node> seg;
    segtree(){}
    segtree(ll n){
        seg.resize(4*n+4,{0});
    }
    segtree(ll n, vector<ll> &ar){
        seg.resize(4*n+4);
        build(ar, 1, 1, n);
    }
    node merge(node a, node b){
        node k;
        k.sum=max(a.sum,b.sum);
        return k;
    }
    // build segtree
    node get(ll pos,ll l,ll r,ll st,ll en){
        if(l>en || r<st || l>r){
            node k={-1};
            return k;
        }
        if(st<=l && en>=r){
```

```
            return seg[pos];
        }
        ll b=(l+r)/2;
        return merge(get(2*pos,l,b,st,en),get(2*pos+1,b+1,r,st,en));
    }
    // update segtree
};
ll query(segtree &obj,ll u,ll v){
    if(u==v) return 0;
    ll uchain,vchain=nd[v].chain,cost=0;
    while(1){
        uchain=nd[u].chain;
        if(uchain==vchain){
            if(u==v)return cost;
            return
                max(cost,obj.get(1,1,n-1,nd[v].posInBase+1,nd[u].posInBase).sum);
        }
        cost=max(cost,obj.get(1,1,n-1,nd[chainHead[uchain]].posInBase,nd[u].posInBase).sum);
        u=nd[chainHead[uchain]].par;
    }
}
ll lca(ll u,ll v){
    if(nd[u].depth < nd[v].depth) swap(u,v);
    ll log;
    for(log=1;(1<<log)<=nd[u].depth;log++);
    log--;
    for(ll i=log;i>=0;i--){
        if(nd[u].depth-(1<<i)>=nd[v].depth){
            u=pa[u][i];
        }
    }
    if(u==v) return v;
    for(ll i=log;i>=0;i--){
        if(pa[u][i]!=-1 && pa[u][i]!=pa[v][i])
            u=pa[u][i],v=pa[v][i];
    }
    return pa[u][0];
}
void hld(ll cur,ll cost,ll pre){
    if(chainHead[chainNo]==-1){
        chainHead[chainNo]=cur;
    }
    nd[cur].chain=chainNo;
    nd[cur].posInBase=ptr;
    baseArray[ptr++]=cost;
    ll sc=-1,ncost;
    for(auto i:ar[cur]){
        if(i.first==pre) continue;
        if(sc==-1 || nd[sc].size<nd[i.first].size){
            sc=i.first;
            ncost=i.second.first;
        }
    }
    if(sc!=-1){
        hld(sc,ncost,cur);
    }
    for(auto i:ar[cur]){
        if(i.first==pre) continue;
        if(sc!=i.first){
            chainNo++;
            hld(i.first,i.second.first,cur);
        }
    }
}
void dfs(ll x,ll p,ll d){
    nd[x].depth=d;
    nd[x].par=p;
    nd[x].size=1;
    for(auto i:ar[x]){
        if(i.first==p) continue;
        otherEnd[i.second.second]=i.first;
        dfs(i.first,x,d+1);
        nd[x].size+=nd[i.first].size;
    }
}
int main(){
    ll t; cin>>t;
    while(t--){
        cin>>n;
        chainNo=0,ptr=0;
        for(ll i=0;i<=n;i++){
            ar[i].clear();
            chainHead[i]=-1;
            for(ll j=0;j<ln;j++){
                pa[i][j]=-1;
            }
        }
        for(ll i=1,u,v,w;i<n;i++){
            cin>>u>>v>>w;
```

```
        ar[u].push_back(mk(v,mk(w,i)));
        ar[v].push_back(mk(u,mk(w,i)));
    }
    dfs(1,0,-1);
    hld(1,-1,-1);
    segtree obj(n-1,baseArray);
    for(ll i=1;i<=n;i++)
        pa[i][0]=nd[i].par;
    for(ll i=1;i<ln;i++)
        for(ll j=1;j<=n;j++)
            if(pa[j][i-1]!=-1)
                pa[j][i]=pa[pa[j][i-1]][i-1];
    while(1){
        string s;
        cin>>s;
        if(s[0]=='D') break;
        ll a,b;
        cin>>a>>b;
        if(s[0]=='Q')
            cout << max(query(obj,a,lca(a,b)),query(obj,b,lca(a,b)))
                << "\n";
        else{
            obj.update(1,1,n-1,nd[otherEnd[a]].posInBase,b);
        }
    }
  }
}
```

3.6 Kruskals Algorithm

```
ll find(ll s){
    if(parent[s]==s){
        return s;
    }
    return parent[s]=find(parent[s]);
}
void unionSet(ll x, ll y){
    ll a = find(x);
    ll b = find(y);
    if(unionSize[a] > unionSize[b]){
        swap(x, y);
    }
    parent[a] = b;
    unionSize[b] += unionSize[a];
}
ll kruskals(ll M){
    ll ans = 0;
    for(ll i=0; i<M; i++){
        ll u = weights[i].ss.ff;
        ll v = weights[i].ss.ss;
        ll w = weights[i].ff;

        if(find(u)!=find(v))
        {
            ans+=w;
            unionSet(u, v);
        }
    }
    return ans;
}
int main(){
    ll N, M;
    cin >> N >> M;
    for(ll i=0; i<L; i++)
    {
        parent[i] = i;
        unionSize[i] = 1;
    }
    for(ll i=0; i<M; i++)
    {
        ll u, v, w;
        cin >> u >> v >> w;

        adj[u].pb(mp(v, w));
        adj[v].pb(mp(u, w));

        weights.pb(mp(w, mp(u, v)));
    }
    sort(weights.begin(), weights.end());
    cout << kruskals(M) << endl;
}
```

3.7 LCA

```
struct LCA {
    vector<ll> height, euler, first, segtree;
```

```
vector<bool> visited;
ll n;
LCA(vector<vector<ll>> &adj, ll root = 0) {
    n = adj.size();
    height.resize(n);
    first.resize(n);
    euler.reserve(n * 2);
    visited.assign(n, false);
    dfs(adj, root);
    ll m = euler.size();
    segtree.resize(m * 4);
    build(1, 0, m - 1);
}
void dfs(vector<vector<ll>> &adj, ll node, ll h = 0) {
    visited[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.push_back(node);
    for (auto to : adj[node]) {
        if (!visited[to]) {
            dfs(adj, to, h + 1);
            euler.push_back(node);
        }
    }
}
void build(ll node, ll b, ll e) {
    if (b == e) {
        segtree[node] = euler[b];
    } else {
        ll mid = (b + e) / 2;
        build(node << 1, b, mid);
        build(node << 1 | 1, mid + 1, e);
        ll l = segtree[node << 1], r = segtree[node << 1 | 1];
        segtree[node] = (height[l] < height[r]) ? l : r;
    }
}
ll query(ll node, ll b, ll e, ll L, ll R) {
    if (b > R || e < L)
        return -1;
    if (b >= L && e <= R)
        return segtree[node];
    ll mid = (b + e) >> 1;

    ll left = query(node << 1, b, mid, L, R);
    ll right = query(node << 1 | 1, mid + 1, e, L, R);
        if (left == -1) return right;
        if (right == -1) return left;
        return height[left] < height[right] ? left : right;
    }
    ll lca(ll u, ll v) {
        ll left = first[u], right = first[v];
        if (left > right)
            swap(left, right);
        return query(1, 0, euler.size() - 1, left, right);
    }
};
vector<vector<ll>>ar;
LCA obj(ar);
```

4 Math

4.1 Extended Euclidean

```
ll x, y;
ll extendedeuc(ll a, ll b){
        if (b==0){
                x=1;
                y=0;
        }
        else{
                extendedeuc(b, a%b);
                ll t=x;
                x=y;
                y=t-y*(a/b);
        }
}
int main(){
    ll a, b, c;
    cin >> a >> b >> c;
        if (c%gcd(a, b)!=0)
        {
                cout << "-1";
                return 0;
        }
        extendedeuc(a, b);
        cout << -x*(c)/gcd(a,b) <<" "<<-y*c/gcd(a, b);
    return 0;
```

```
}
```

4.2 FFT

```
typedef complex<double> cd;
const double PI = acos(-1);
void fft(vector<cd> &a, bool invert){
    ll n=a.size();
    for(ll i=1,j=0; i<n; i++){
        ll bit=n>>1;
        for(; j&bit; bit>>=1)
            j ^= bit;
        j ^= bit;
        if(i < j)
            swap(a[i], a[j]);
    }
    for(ll len=2; len<=n; len <<= 1){
        double ang=2*PI/len*(invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for(ll i=0; i<n; i+=len){
            cd w(1);
            for(ll j=0; j<len/2; j++){
                cd u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if(invert){
        for(cd & x : a)
            x /= n;
    }
}
vector<ll> multiply(vector<ll> const &a, vector<ll> const &b){
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    ll n=1;
    while(n < a.size()+b.size())
        n <<= 1;
    fa.resize(n,0);
    fb.resize(n,0);
    fft(fa, false);
    fft(fb, false);
    for(ll i=0; i<n; i++)
        fa[i] *= fb[i];
    fft(fa, true);
    vector<ll> result(n);
    for(ll i=0; i<n; i++)
        result[i] = llround(fa[i].real());
    return result;
}
int main(){
    ll t;
    cin>>t;
    while(t--){
        ll n;
        cin>>n;
        vector<ll>a(n+1), b(n+1);
        for(ll i=0;i<=n;i++){
            cin>>a[n-i];
        }
        for(ll i=0;i<=n;i++){
            cin>>b[n-i];
        }
        auto c = multiply(a, b);
        for(ll i=2*n;i>=0;i--){
            cout << c[i] << " ";
        }
        cout << "\n";
    }
}
```

4.3 Gauss

```
// --------------- Gauss Jordan ---------------
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big
    number
int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
```

```
            sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }
    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }
    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
//Gauss Jordan For Mod
int gauss (vector < bitset<N> > a, int n, int m, bitset<N> & ans) {
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        for (int i=row; i<n; ++i)
            if (a[i][col]) {
                swap (a[i], a[row]);
                break;
            }
        if (! a[row][col])
            continue;
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row && a[i][col])
                a[i] ^= a[row];
        ++row;
    }
    // The rest of implementation is the same as above
}
```

4.4 Shoelace Formula

```
// Used to calculate area of convex polygon, given
// its coordinates in the x and y plane
// (X[i], Y[i]) are coordinates of i'th point.
double polygonArea(double X[], double Y[], int n) {
    double area = 0.0;
    int j = n - 1;
    for (int i = 0; i < n; i++){
        area += (X[j] + X[i]) * (Y[j] - Y[i]);
        j = i; // j is previous vertex to i
    }
    return abs(area / 2.0);
}
```

4.5 Union of Rectangles

```
/*primes*/
//ll p1=1e6+3, p2=1616161, p3=3959297, p4=7393931;
int n; const int N=1e6;
struct rect{
    int x1, y1, x2, y2;
};
struct event_x{
    int typ, x, idx;
    event_x(int x, int t, int idx):x(x), typ(t), idx(idx){}
};
struct event_y{
    int typ, y, idx;
    event_y(int y, int t, int idx):y(y), typ(t), idx(idx){}
};
vector<rect> vec;
vector<event_x> Sx;
vector<pii> tree;
```

```
vi lazy;
void init(){
    vec.resize(n);
    tree.resize(4*N, mp(0, 0));
    lazy.resize(4*N, 0);
}
bool comp_x(event_x e1, event_x e2){
    if(e1.x!=e2.x) return e1.x<e2.x;
    return e1.typ<e2.typ;
}
void update(int start, int end, int node, int l, int r, int delta){
    int len=end-start+1;
    if(start>r || end<l) return ;

    if(start>=l && end<=r){
        tree[node].ss+=delta;
        if(tree[node].ss==0)
            tree[node].ff=tree[2*node].ff+tree[2*node+1].ff;
        else tree[node].ff=len;
        return ;
    }

    int mid=(start+end)/2;
    update(start, mid, 2*node, l, r, delta);
    update(mid+1, end, 2*node+1, l, r, delta);
    if(tree[node].ss==0) tree[node].ff=tree[2*node].ff+tree[2*node+1].ff;
    return ;
}
int query(int start, int end, int node, int l, int r){
    if(start>r || end<l) return 0;
    if(start>=l && end<=r){
        return tree[node].ff;
    }
    int mid=(start+end)/2;
    return query(start, mid, 2*node, l, r)+query(mid+1, end, 2*node+1, l,
        r);
}
int main(){
    cin>>n;
    init();
    fr(i, n){
        cin>>vec[i].x1>>vec[i].y1>>vec[i].x2>>vec[i].y2;
        Sx.pb(event_x(vec[i].x1, 0, i));
        Sx.pb(event_x(vec[i].x2, 1, i));
    }
    sort(all(Sx), comp_x);
    ll ans=0;
    ll px=Sx[0].x, dy, dx, cnt, py;
    for(auto i:Sx){
        dx=i.x-px;
        dy=query(0, N, 1, 0, N);
        ans+=dx*dy;
        px=i.x;
        if(i.typ==0){
            update(0, N, 1, vec[i.idx].y1, vec[i.idx].y2-1, 1);
            continue;
        }
        update(0, N, 1, vec[i.idx].y1, vec[i.idx].y2-1, -1);
    }
    cout<<ans<<endl;
}
```

5 Misc

5.1 Hare Tortoise Mehod

```
// UVA 11053
ll a, b, N;
ll f(ll x){
    return (((a*x)%N*x)%N + b)%N;
}
int main()
{
    cin >> N >> a >> b;
    ll tortoise = f(0);
    ll hare = f(f(0));
    while(tortoise != hare)
    {
        tortoise = f(tortoise);
        hare = f(f(hare));
    }
    ll die = 1;
    tortoise = f(tortoise);
    while(tortoise != hare)
    {
        tortoise = f(tortoise);
        die++;
```

```
    }
    cout << N - die << endl;
}
```

6 SqrtDecomp

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

int build(int ary[],int sto[],int n)
{
	int a=sqrt(n);

	for (int i = 0; i < n; ++i)
	{
		sto[i/a]+=ary[i];
	}

	for (int i = 0; i < ceil(sqrt(n)); ++i)
	{
		cout << sto[i]<<" ";
	}
	cout << endl;

}


int main()
{
	int n;
	cin >> n;
	int ary[n];
	for (int i = 0; i < n; ++i) cin >> ary[i];

	int a=sqrt(n);
	int sto[a+1];
	for (int i = 0; i < a+1; ++i)sto[i]=0;
	build(ary,sto,n);

	int q;
	cin >> q;
	while(q--)
	{
		int type;
		cin >> type;
		if(type==1) //update
		{
			int ind,val;
			cin >> ind >> val;
			sto[ind/a]+=(val-ary[ind]);
			ary[ind]=val;
		}
		else
		{
			int l,r;
			cin >> l >> r;
			int ans=0;
			for (int i = l; i <=r;)
			{
				if(i%a==0&&r-i>=a)
				{
					ans+=sto[i/a];
					i+=a;
				}
				else
				{
					ans+=ary[i];
					i++;
				}
			}
			cout << ans << endl;
		}
	}
	return 0;

}
```

7 String Algorithms

7.1 KMP

```
int main(){
    string c,t;
    cin>>c>>t;
```

```
    ll l=t.length();
    vector<ll>p(l);
    p[0]=0;
    for(ll i = 1, j = 0; i < l; i++){
        while(j > 0 && t[i] != t[j]){
            j = p[j-1];
        }
        if(t[i] == t[j])
            j++;
        p[i] = j;
    }
    ll n = c.length(), ans=0;
    for(ll i = 0,j = 0; i < n; i++){
        if(c[i] == t[j]){
            if(j == l-1){
                ans++;
                j = p[j];
                continue;
            }
            j++;
        }
        else if(j > 0){
            j = p[j-1];
            i--;
        }
    }
}
```

8 Templates

9 closest$_p air$

```
const ll N=1e5+5,inf=1e18;
pll pnts [N];
int compare(pll a, pll b)
{
        return a.px<b.px;
}
double closest_pair(int n)
{
        sort(pnts,pnts+n,compare);
        double best=inf;
        set<pll> box;
        box.insert(pnts[0]);
        int left = 0;
        for (int i=1;i<n;++i)
        {
            while (left<i && pnts[i].px-pnts[left].px > best)
                box.erase(pnts[left++]);
            ll cnt=0;
            cout<<pnts[i].px<<" "<<pnts[i].py<<endl;
            for(auto it=box.lower_bound(make_pair(pnts[i].py-best,
                 pnts[i].px-best));it!=box.end() &&
                 pnts[i].py+best>=it->py;it++)
            {
                cnt++;
                best = min(best, sqrt(pow(pnts[i].py - it->py,
                    2.0)+pow(pnts[i].px - it->px, 2.0)));
            }
            box.insert(pnts[i]);
        }
        return best;
}
int main()
{
        ll n;
        cin >> n;
        lp(i,0,n)
        {
                ll a,b;
                cin >> a >> b;
                pnts[i].px=a;
                pnts[i].py=b;
        }
        cout<<closest_pair(n)<<endl;
        return 0;
}
```

10 maxflow$_m anish$

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

```
class Dinics {
public:
	typedef int flowType; // can use float/double
	static const flowType INF = 1e9; // maximum capacity
	static const flowType EPS = 0; // minimum capacity/flow change

private:
	int nodes, src, dest;
	vector<int> dist, q, work;
	struct Edge {
	  int to, rev;
	  flowType f, cap;
	};
	vector< vector<Edge> > g;

	bool dinic_bfs() {
	  fill(dist.begin(), dist.end(), -1);
	  dist[src] = 0;
	  int qt = 0;
	  q[qt++] = src;
	  for (int qh = 0; qh < qt; qh++) {
	    int u = q[qh];
	    for (int j = 0; j < (int) g[u].size(); j++) {
	      Edge &e = g[u][j];
	      int v = e.to;
	      if (dist[v] < 0 && e.f < e.cap) {
	        dist[v] = dist[u] + 1;
	        q[qt++] = v;
	      }
	    }
	  }
	  return dist[dest] >= 0;
	}
	int dinic_dfs(int u, int f) {
	  if (u == dest)
	    return f;
	  for (int &i = work[u]; i < (int) g[u].size(); i++) {
	    Edge &e = g[u][i];
	    if (e.cap <= e.f) continue;
	    int v = e.to;
	    if (dist[v] == dist[u] + 1) {
	      flowType df = dinic_dfs(v, min(f, e.cap - e.f));
	      if (df > 0) {
	        e.f += df;
	        g[v][e.rev].f -= df;
	        return df;
	      }
	    }
	  }
	  return 0;
	}

public:
	Dinics(int n): dist(n, 0), q(n, 0),
		work(n, 0), g(n), nodes(n) {}

	// s->t (cap); t->s (rcap)
	void addEdge(int s, int t, flowType cap, flowType rcap = 0) {
	  g[s].push_back({t, (int) g[t].size(), 0, cap});
	  g[t].push_back({s, (int) g[s].size() - 1, 0, rcap});
	}
	flowType maxFlow(int _src, int _dest) {
	  src = _src;
	  dest = _dest;
	  flowType result = 0;
	  while (dinic_bfs()) {
	    fill(work.begin(), work.end(), 0);
		flowType delta;
	    while ((delta = dinic_dfs(src, INF)) > EPS)
	      result += delta;
	  }
	  return result;
	}
};

vector<pair<ll,ll>> g[100];
int main()
{
	ll n,m,x;
	cin>>n>>m>>x;
	for(ll i=1;i<=m;i++)
	{
		ll u, v, c;
		cin>>u>>v>>c;
		g[u].push_back({v, c});
		// g[v].push_back({u, c});
	}
	double lb=0, ub=10000000, mid/*(lb+ub)/2*/;
	double ans=0;
	int cnt=100;
```

```
	while(cnt)
	{
		cnt--;
		mid=(lb+ub)/2;
		Dinics d(n);
		for (int i = 1; i < n+1; ++i)
		{
			for(auto j:g[i])
			{
				if (j.second/mid>1e7)
				{
					d.addEdge(i-1, j.first-1, x);
				}
				else
				{
					d.addEdge(i-1, j.first-1,
					    floor((j.second)/mid));
				}
			}
		}
		if(d.maxFlow(0, n-1)>=x)
		{
			lb=mid;
		}
		else
		{
			ub=mid;
		}
		ans=mid;
	}
	cout <<fixed<<setprecision(10)<< ans*x;
	return 0;
}
```

11 minCost$_{m}axFlow$

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

struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
```

```
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}

int main()
{
    return 0;
}
```

12 mosAlgo$_{w}ithUpdates$

```
// 1 based indexing

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

```
int n;
int z;
bool cmp0(const vector <int> & a,const vector < int> & b)
{
        if(a[1]/z==b[1]/z)
        {
                if(a[2]==b[2])
                {
                        return a[3]<b[3];
                }
                return a[2]<b[2];
        }
        else
        {
                return a[1]<b[1];
        }
}



bool cmp3(const vector <int> & a,const vector < int> & b)
{
        return a[0]<b[0];
}

int main()
{
        cin >> n;
        int ary[n+1];
        for (int i = 1; i <= n; ++i) scanf("%d",&ary[i]);

        int q;
        cin >> q;
        vector<vector<int> > ques(q);
        vector<vector<int> > upd(q);
        int t=0;        //timestamp
        int j=0;
        for (int i = 0; i < q; ++i)
        {
                int type;
                cin >> type
                if(type==1)
                {
                        int ind,val;
```

```
                cin >> ind >> val;
                upd[t].push_back(ind);
                upd[t].push_back(val);
                t++;
        }
        else
        {
                int l,r;
                scanf("%d%d",&l,&r);
                ques[j].push_back(i);
                ques[j].push_back(l);
                ques[j].push_back(r);
                ques[j].push_back(t);
                j++;
        }
}
z=sqrt(n);
sort(ques.begin(),ques.end(),cmp0);

int a=sqrt(n);

int x=0,y=0;

int freq[1000001];
for (int i = 0; i < 1000001; ++i) freq[i]=0;

int left=ques[0][1];
int c =1;
int right=ques[0][1];
freq[ary[right]]++;
int time=0;
for (int i = 0; i < j
        ; ++i)
{
        if(ques[i][3]>time)
        {
                while(ques[i][3]!=time)
                {
                        time++;
                        if(left<=upd[time][0]&&right>=upd[time][0])
                        {
                                freq[ ary[ upd[time][0] ] ]--;
                                freq[upd[time][1]]++;
                        }
                        //jab vaapis ana ho uske liye jugaad
                        int temp=ary[ upd[time][0] ];
                        ary[upd[time][0]]=upd[time][1];
                        upd[time][1]=temp;
                }
        }
        else if(ques[i][3]<time)
        {
                while(ques[i][3]!=time)
                {
                        if(left<=upd[time][0]&&right>=upd[time][0])
                        {
                                freq[ ary[ upd[time][0] ] ]--;
                                freq[upd[time][1]]++;
                        }

                        int temp=ary[ upd[time][0] ];
                        ary[upd[time][0]]=upd[time][1];
                        upd[time][1]=temp;
                        time--;
                }
        }

        if(ques[i][1]>left)
        {
                while(ques[i][1]!=left)
                {
                        freq[ary[left]]--;
                        if(freq[ary[left]]==0)
                        {
                                c--;
                        }
                        left++;
                        }
                }
        else if(ques[i][1]<left)
        {
                while(ques[i][1]!=left)
                {
                        left--;
                        freq[ary[left]]++;
                        if(freq[ary[left]]==1)
                        {
                                c++;
                        }
                }
```

```
        }

        if(ques[i][2]>right)
        {
                while(ques[i][2]!=right)
                {
                        right++;
                        freq[ary[right]]++;
                        if(freq[ary[right]]==1)
                        {
                                c++;
                        }
                }
        }

        if(ques[i][2]<right)
        {
                while(ques[i][2]!=right)
                {
                        freq[ary[right]]--;
                        if(freq[ary[right]]==0)
                        {
                                c--;
                        }
                        right--;
                }
        }


        ques[i].push_back(c);
}
        sort(ques.begin(),ques.end(),cmp3);
        for (int i = 0; i < q; ++i)
        {
                printf("%d\n",ques[i][3]);
        }
}
```

13 mos$_algo$

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

void remove(idx); // TODO: remove value at idx from data structure
void add(idx);   // TODO: add value at idx from data structure
int get_answer(); // TODO: extract the current answer of the data
    structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l,
        cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
```

```
    }
    return answers;
}

int main()
{
    return 0;
}
```

14 ntt

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long int
#define ve vector
#define INF 1e16


// -------------------- Fast WalshHadamard transform (XOR)-------------

#define poly vector<ll>

poly FWHT(poly P, bool inverse) {
    for (len = 1; 2 * len <= degree(P); len <<= 1) {
        for (i = 0; i < degree(P); i += 2 * len) {
            for (j = 0; j < len; j++) {
                u = P[i + j];
                v = P[i + len + j];
                P[i + j] = u + v;
                P[i + len + j] = u - v;
            }
        }
    }

    if (inverse) {
        for (i = 0; i < degree(P); i++)
            P[i] = P[i] / degree(P);
    }

    return P;
}
```

```
// ----------------------- & operator -----------------

poly transform(poly P, bool inverse) {
    for (len = 1; 2 * len <= degree(P); len <<= 1) {
        for (i = 0; i < degree(P); i += 2 * len) {
            for (j = 0; j < len; j++) {
                u = P[i + j];
                v = P[i + len + j];

                if (!inverse) {
                    P[i + j] = v;
                    P[i + len + j] = u + v;
                } else {
                    P[i + j] = -u + v;
                    P[i + len + j] = u;
                }
            }
        }
    }

    return P;
}

// ------------------ NTT ---------------------------

//                              k             g
// 5767169     19      3
// 7340033     20      3
// 23068673    21      3
// 104857601   22      3
// 167772161   25      3
// 469762049   26      3
// 998244353   23      3
// 1004535809  21      3
// 2013265921  27      31
// 2281701377  27      3


const ll mod = 998244353;
ll inverse(ll x, ll y){
       ll rem = 1;
       while(y != 0){
              if(y % 2 == 1){
                     rem=(rem * x) % mod;
              }
```

```
                x=(x * x) % mod;
                y /= 2;
        }
        return rem;
}
const ll root = 3;
const ll root_1 = inverse(root, mod - 2);
const ll root_pw = 1 << 23;
void ntt(vector<ll> &a, bool invert){
        ll n = a.size();
        for(ll i = 1, j = 0; i < n; i++){
                ll bit = n >> 1;
                for(; j & bit; bit >>= 1)
                        j ^= bit;
                j ^= bit;
                if(i < j)
                        swap(a[i], a[j]);
        }

        for(ll len = 2; len <= n; len <<= 1){
                ll wlen = invert ? root_1 : root;
                for(ll i = len; i < root_pw; i <<= 1)
                        wlen = wlen * wlen % mod;

                for(ll i = 0; i < n; i += len){
                        ll w = 1;
                        for(ll j = 0; j < len / 2; j++){
                                ll u = a[i + j], v = a[i + j + len / 2] * w
                                    % mod;
                                a[i + j] = u + v < mod ? u + v : u + v - mod;
                                a[i + j + len / 2] = u - v >= 0 ? u - v : u
                                    - v + mod;
                                w = w * wlen % mod;
                        }
                }
        }

        if(invert){
                ll n_1 = inverse(n, mod - 2);
                for(ll &x:a)
                        x = x * n_1 % mod;
        }
}

vector<ll> multiply(vector<ll> const &a, vector<ll> const &b){
        vector<ll> fa(a.begin(), a.end()), fb(b.begin(), b.end());
        ll n = 1;
        while(n < a.size() + b.size())
                n <<= 1;
        fa.resize(n, 0);
        fb.resize(n, 0);
        ntt(fa, false);
        ntt(fb, false);
        for(ll i = 0; i < n; i++)
                fa[i] = fa[i] * fb[i] % mod;
        ntt(fa, true);

        return fa;
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    ll t;
    cin>>t;
    while(t--){
        ll n;
        cin>>n;
        vector<ll>a(n+1), b(n+1);
        for(ll i=0;i<=n;i++){
            cin>>a[n-i];
        }
        for(ll i=0;i<=n;i++){
            cin>>b[n-i];
        }
        auto c = multiply(a, b);
        for(ll i=2*n;i>=0;i--){
            cout << c[i] << " ";
        }
        cout << "\n";
    }
}
```

15 persistent$_{s}egTress$

```
// spoj problem COT using persistent seg trees
```

```
/* Problem Statement:
You are given a tree with N nodes. The tree nodes are numbered
from 1 to N. Each node has an integer weight.

We will ask you to perform the following operation:

    u v k : ask for the kth minimum weight on the path from
    node u to node v

Input
In the first line there are two integers N and M. (N, M <= 100000)
In the second line there are N integers. The ith integer denotes the
weight of the ith node.
In the next N-1 lines, each line contains two integers u v,
which describes an edge (u, v).
In the next M lines, each line contains three integers u v k,
which means an operation asking for the kth minimum weight on
the path from node u to node v.

Output
For each operation, print its result.
*/

#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
typedef long double ld;

#define lp(var,start,end) for (ll var = start; var <end ; ++var)
#define rlp(var,start,end) for(ll var = start; var>=end ; var--)
#define pb push_back
#define mp make_pair
#define pf push_front
#define F first
#define S second
#define vll vector<ll>
#define pll pair<ll,ll>
#define endl "\n"

const ll N=1e5+5,LOGN=20;
ll v[N],av[N],dp[N][LOGN],lvl[N],n,c;
map<ll,ll> M;
vll G[N];
```

```
struct node{
        node * left;
        node * right;
        ll val;
};


node * root;
node * R[N];

node * init()
{
        node * temp=new node;
        temp->left = temp->right = NULL;
        temp->val =    0;
        return temp;
}

ll jump(ll x, ll d){
        for(ll i = LOGN-1 ; i>=0 ; i--){
                if( (1<<i) <= d ){
                        x = dp[x][i];
                        d -= (1<<i);
                }
        }
        return x;
}

ll LCA(ll a, ll b){
        if( lvl[a] > lvl[b] ) a = jump(a, lvl[a]-lvl[b]);
        if( lvl[b] > lvl[a] ) b = jump(b, lvl[b]-lvl[a]);

        if( a == b )  return a;
        for(ll i = LOGN-1 ; i>=0 ; i--)
        {
                if( dp[a][i] != dp[b][i] )
                {
                        a = dp[a][i];
                        b = dp[b][i];
                }
        }
        return dp[a][0];
}
```

```
node * build(ll l,ll r,ll w,node * par)
{
	// cerr <<l<<"  "<<r<<endl;
	if(l==r)
	{
		node * temp=init();
		temp->val=par->val+1;

		return temp;
	}
	if(w>=l&&w<=r)
	{
		ll mid=(l+r)/2;
		node * temp= init();
		if(w<=mid)
		{
			temp->left=build(l,mid,w,par->left);
			temp->right=par->right;
		}
		else
		{
			temp->right=build(mid+1,r,w,par->right);
			temp->left=par->left;
		}
		temp->val=temp->right->val+temp->left->val;
		return temp;
	}
}

node *buildspecial(ll l,ll r)
{
	if(l==r)
	{
		node * temp =init();
		return temp;
	}
	ll mid=(l+r)/2;
	node * temp= init();
	temp->left=buildspecial(l,mid);
	temp->right=buildspecial(mid+1,r);
	temp->val=temp->right->val+temp->left->val;
	return temp;
}
ll dfs(ll a,ll par)
{
	// cerr<<a<<" "<<v[a]<<" "<<M[v[a]]<<endl;
	R[a]=init();
	R[a]=build(0,c-1,M[v[a]],R[par]);
	lp(i,0,G[a].size())
	{
		ll x=G[a][i];
		if(x!=par)
		{
			dp[x][0]=a;
			lvl[x]=lvl[a]+1;
			dfs(x,a);
		}
	}
}
ll query(node* a,node* b,node* c,node* d,ll l,ll r,ll k)
{
	if(l==r)
		return l;
	ll ct=a->left->val+b->left->val-c->left->val-d->left->val;
	ll m=(l+r)>>1;
	if(ct>=k)
		return query(a->left,b->left,c->left,d->left,l,m,k);
	return query(a->right,b->right,c->right,d->right,m+1,r,k-ct);
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
	cout.tie(NULL);
    ll m;
    cin >> n >> m;
    root = init();
    lp(i,0,n)
    {
        cin >> v[i+1];
        M[v[i+1]];
    }
    for(auto it= M.begin();it!=M.end();it++)
    {
        M[it->first]=c;
        av[c++]=it->first;
    }
    lp(i,0,n-1)
    {
        ll a,b;
```

```
        cin >> a >> b;
        G[a].pb(b);
        G[b].pb(a);
    }
    root->left = init();
    root->right = init();

    dp[1][0]=0;
    lvl[1]=0;
    R[0]=buildspecial(0,c-1);
    dfs(1,0);


        lp(j,1,LOGN)
        {
                lp(i,1,n+1)
                {
                        if(lvl[i]-(1<<j)>=0)
                                dp[i][j]=dp[dp[i][j-1]][j-1];
                        else dp[i][j]=1;
                }
        }
        while(m--)
        {
                ll u,v,k;
                cin>>u>>v>>k;
                ll t=LCA(u,v);
                cout<<av[query(R[u],R[v],R[t],R[dp[t][0]],(ll)0,c-1,k)]<<endl;
        }
        return 0;
}
```

16 persistent$_t ries$

```
/*
CodeChef Problem XORMIN
Problem Statement:

You are given a rooted tree with N vertices (numbered 1 through N);
vertex 1 is the root. Each vertex has a weight; let's denote the
weight of vertex i by wi.

You should answer Q queries. The queries have to be processed online,
i.e. to obtain each query, you need the answer to the previous query.

In each query, you are given a vertex v
and a parameter k. For each vertex u in the subtree of v (including v),
consider the value wuk (    denotes the bitwise XOR operation). The
answer to this query is the maximum of these values and the smallest u
such that vertex u is in the subtree of vertex v and wuk is equal to
this maximum.

Credits - vivace_jr
*/

#include<bits/stdc++.h>
using namespace std;
typedef int ll;
typedef long double ld;

#define lp(var,start,end) for (ll var = start; var <end ; ++var)
#define rlp(var,start,end) for(ll var = start; var>=end ; var--)
#define pb push_back
#define mp make_pair
#define pf push_front
#define ff first
#define ss second
#define vll vector<ll>
#define vld vector<ld>
#define pll pair<ll,ll>
#define pld pair<ld,ld>
#define vpll vector<pll>
#define vpld vector<pld>

#define all(X) X.begin(),X.end()
#define endl "\n"

const ll N=2e5+5;
ll st[N],ed[N];
vll G[N];
vll tour;
ll tme=0,cur=0;
ll dfs(ll a,ll p)
{
        tour.pb(a);
        st[a]=tme++;
        lp(i,0,G[a].size())
```

```
                if(G[a][i]!=p)
                        dfs(G[a][i],a);
        ed[a]=tme-1;
}
struct trie{
        ll last[2];
        ll nxt[2];
};

trie Node[N*21];
ll getNode()
{
        trie temp;
        temp.last[0]=-1;
        temp.last[1]=-1;
        temp.nxt[0]=-1;
        temp.nxt[1]=-1;
        Node[cur]=temp;
        cur++;
        return (cur-1);
}
ll root[N];

ll insert(ll nd,ll par,vll s,ll id)
{
        lp(i,0,20)
        {
                if(Node[nd].nxt[s[i]]==-1)
                {
                        ll z=getNode();
                        Node[nd].nxt[s[i]]=z;
                        Node[nd].last[s[i]]=id;
                        if(par!=-1)
                        {
                                Node[nd].nxt[1-s[i]]=Node[par].nxt[1-s[i]];
                                Node[nd].last[1-s[i]]=Node[par].last[1-s[i]];
                        }
                }
                nd=Node[nd].nxt[s[i]];
                if(par!=-1)
                        par=Node[par].nxt[s[i]];
        }
}

int main()
{
    ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
        ll t;
        cin >> t;
        while(t--)
        {
                ll n,q;
                cin >> n >> q;
                ll ary[n];
                lp(i,0,n)
                        cin >> ary[i];
                lp(i,0,n-1)
                {
                        ll a,b;
                        cin >> a >> b;
                        G[a].pb(b);
                        G[b].pb(a);
                }
                dfs(1,0);
                map<ll,ll> M;
                ll mc=0;
                vll tp;
                lp(i,0,n)tp.pb(ary[i]);
                sort(all(tp));
                lp(i,0,n)
                {
                        if(M.count(tp[i])==0)M[tp[i]]=mc++;
                }
                lp(i,0,n)tp[i]=M[ary[i]];
                cout<<endl;
                vpll freq[mc+1];
                ll sorted[mc+1];
                memset(sorted,0,sizeof(sorted));
                vll sparse[mc+1][20];
                lp(i,0,n)
                {
                        vll s;
                        freq[tp[tour[i]-1]].pb(mp(i,tour[i]-1));
                }
                lp(i,0,n)
                {
                        if(sorted[tp[i]]==0)
                        {
                                sorted[tp[i]]=1;
                                ll sz=freq[tp[i]].size();
```

```
                lp(j,0,sz)
                {
                        sparse[tp[i]][0].pb(freq[tp[i]][j].ss);
                }
                lp(j,1,20)
                {
                        if((1<<j)>sz)break;
                        ll flag=0;
                        lp(k,0,sz)
                        {
                                if(k+(1<<(j-1))>=sz)
                                {
                                        break;
                                }
                                if(sparse[tp[i]][j-1][k] <
                                    sparse[tp[i]][j-1][k+(1<<(j-1))])
                                {
                                        sparse[tp[i]][j].pb(sparse[tp[i]][j-1][k]);
                                }
                                else
                                {
                                        sparse[tp[i]][j].pb(sparse[tp[i]][j-1][k+(1<<(j-1))]);
                                }
                        }
                }
        }
}
lp(i,0,n)
{
        root[i]=getNode();
        vll s;
        rlp(j,19,0)
        {
                if((ary[tour[i]-1]&(1<<j))==(1<<j))
                {
                        s.pb(1);
                }
                else
                {
                        s.pb(0);
                }
        }
        if(i>0)
                insert(root[i],root[i-1],s,i);
        else
                insert(root[i],-1,s,i);
}
ll p1=0,p2=0;
while(q--)
{
        ll val,k;
        cin >> val >> k;
        valˆ=p1;
        kˆ=p2;
        vll s;
        ll x=0;
        ll cur2 = root[ed[val]];
        lp(i,0,20)
        {
                ll d=0;
                if( (k&(1<<(19-i))) == (1<<(19-i))  )
                        d=1;
                if(Node[cur2].nxt[1-d]!=-1&&Node[cur2].last[1-d]>=st
                {
                        x+=(1<<(19-i));
                        cur2=Node[cur2].nxt[1-d];
                }
                else
                {
                        cur2=Node[cur2].nxt[d];
                }
        }
        x=xˆk;
        ll y=M[x];
        ll
            l=lower_bound(all(freq[y]),mp(st[val],(ll)-1))-freq[y].
        ll
            r=upper_bound(all(freq[y]),mp(ed[val],N*100))-freq[y].b
        r--;
        ll len=log2(r-l+1);
        ll ans=1e8;
        if(sparse[y][len][l] <=
            sparse[y][len][r-(1<<len)+1])
        {
                ans=sparse[y][len][l];
        }
        else
        {
                ans=sparse[y][len][r-(1<<len)+1];
        }
```

```
                cout<<ans+1<<" "<<(x^k)<<endl;
                p1=ans+1;
                p2=x^k;
            }
            cur=0;
            tme=0;
            lp(i,0,n+1)
            {
                root[i]=-1;
                G[i].clear();
            }
            tour.clear();
        }
        return 0;
}
```

17 sparse$_m$atrix

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
typedef long double ld;

#define lp(var,start,end) for (ll var = start; var <end ; ++var)
#define rlp(var,start,end) for(ll var = start; var>=end ; var--)
#define pb push_back
#define mp make_pair
#define pf push_front
#define F first
#define S second
#define vll vector<ll>
#define pll pair<ll,ll>
#define endl "\n"
const ll N=3e3+5,mod=998244353;
typedef vector<pair<pll,ll> > spm;
ll n,r,k;

ll expo(ll b,ll e,ll m)
{
    ll a=1;
   while(e !=0 )
   {
        if((e&1) == 1)
            a= (a*b)%m;
        b=(b*b)%m; e>>= 1;
        }
        return a%m;
}

spm mul(spm a,spm b)
{
        spm result;
        // b=transpose(b);
        ll M[n+1][n+1];

        memset(M,0,sizeof(M));
        lp(i,0,a.size())
        {
                lp(j,0,b.size())
                {
                        if(a[i].F.S==b[j].F.F)
                        {
                                M[a[i].F.F][b[j].F.S]+=a[i].S*b[j].S;
                                M[a[i].F.F][b[j].F.S]%=mod;
                        }
                }
        }
        lp(i,1,n+1)
        {
                lp(j,1,n+1)
                {
                        if(M[i][j]>0)
                        {
                                result.pb(mp(mp(i,j),M[i][j]));
                        }
                }
        }
        return result;
}

spm pow(spm a,ll k)
{
        if(k==1)return a;
        if(k%2)
        {
                return mul(a,pow(a,k-1));
        }
```

```
        spm ret = pow(a,k/2);
        return mul(ret,ret);
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
        cin >> n;
        spm sp;
        lp(i,0,n-1)
        {
                ll a,b;
                cin >>a >> b;
                sp.pb(mp(mp(a,b),(ll)1));
                sp.pb(mp(mp(b,a),(ll)1));
        }
        sort(sp.begin(),sp.end());
        cin >> r >> k;
        sp = pow(sp,k);
        ll ans[n+1];
        memset(ans,0,sizeof(ans));
        lp(i,0,sp.size())
        {
                if(sp[i].F.S==r)
                {
                        ans[sp[i].F.F]=sp[i].S;
                }
        }
        lp(i,1,n+1)
        {
                cout<<ans[i]%mod<<" ";
        }
        cout<<endl;
        return 0;
}
```

18 sparsetable

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
#define lp(var,start,end) for (ll var = start; var <end ; ++var)
#define rlp(var,start,end) for(ll var = start; var>=end ; var--)
#define pb push_back
#define mp make_pair

/* ZERO-INDEXED SPARSE TABLE FOR RANGE SUM QUERY PROBLEM WITHOUT UPDATES*/

ll dp[10000][100];
ll init(ll pow,ll n)
{
        lp(i,0,n+1)
        {
                lp(j,0,pow+1)
                {
                        dp[i][j]=0;
                }
        }
}
ll ans;
ll ary[100000];

ll query(ll l,ll r)
{
                while(l<=r)
                {
                        ll k=1;
                        ll cnt=0;
                        if(l==r)
                        {
                                ans+=ary[r];
                                break;
                        }
                        while(l+k<=r)
                        {
                                k=k<<1;
                                cnt++;
                        }
                        ans+=dp[l][cnt-1];
                        l=l+(k>>1);
                }
}

int main()
{
```

```
	ll n;
	cin >> n;
	lp(i,0,n) cin >> ary[i];
	ll a=1;
	ll pow=0;
		while(a<n)
		{
			a=a*2;
			pow++;
		}

		lp(i,n,a) ary[i]=0;
		n=a;
		init(pow,n);

		lp(i,0,n) dp[i][0]=ary[i];
	lp(j,1,pow+1)
	{
		lp(i,0,n)
		{
			if(i+(1<<(j-1)) < n)
			{
				dp[i][j]=dp[i][j-1]+dp[i+(1<<(j-1))][j-1];
			}
		}
	}

	ll q;
	cin >> q;
	while(q--)
	{
		ll a,b;
		cin >> a>>b;
		ans=0;
		query(a-1,b-1);

		cerr <<ans<<endl;
	}
}
```

19 suffix$_array$

```
// https://www.spoj.com/problems/SUBST1/
// find distinct substrings in array.

#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;
typedef long double ld;

#define lp(var,start,end) for (ll var = start; var <end ; ++var)
#define rlp(var,start,end) for(ll var = start; var>=end ; var--)
#define pb push_back
#define mp make_pair
#define pf push_front
#define ff first
#define ss second
#define vll vector<ll>
#define pll pair<ll,ll>
#define vpll vector<pll>
#define all(X) X.begin(),X.end()
#define endl "\n"
#define SET(A, val) memset(A, val, sizeof(A))
// Structure to store information of a suffix
#define FN(i, n) for (int i = 0; i < (int)(n); ++i)
#define FEN(i,n) for (int i = 1;i <= (int)(n); ++i)
#define sz(a) (int)(a).size()
#define rep(i,a,b) for(int i=a;i<b;i++)
#define repv(i,a,b) for(int i=b-1;i>=a;i--)
const int N = (int)2e5 + 10;
const int LOGN = 22;
vector<int> lcp(N, 0);
//SA[i] = ith Lexicographically smallest suffixs index.
int
RA[LOGN][N],SA[N],tempSA[N],cnt[N],LCP[LOGN][N];
void cSort(int l,int k,int n) {
	SET(cnt,0);
    rep(i,0,n) cnt[(i+k<n?RA[l][i+k]:0)]++;
    int maxi=max(300,n),t;
    for(int i=0,sum=0;i<maxi;i++) {
        t = cnt[i], cnt[i] = sum, sum += t;// index
    }
    rep(i,0,n)tempSA[cnt[(SA[i]+k<n?RA[l][SA[i]+k]:0)]++]=SA[i];
    rep(i,0,n)SA[i]=tempSA[i];
}//dollar[i]: next       $       in string.
void build_SA(string &s){
    int n = sz(s);
```

```
    rep(i,0,n) RA[0][i] = s[i], SA[i] = i;
    rep(i,0,LOGN-1){
        int k=(1<<i);
        if(k>=n)break;
        cSort(i,k,n);cSort(i,0,n);
        int rank=0;RA[i+1][SA[0]]=rank;
        rep(j,1,n){
            if((RA[i][SA[j]]==RA[i][SA[j-1]])&&(RA[i][SA[j]+k]==RA[i][SA[j-1]+k]))
                RA[i+1][SA[j]]=rank;
            else RA[i+1][SA[j]]=++rank;
        }
    }
}

vector<int> build_LCP(string txt)
{
        ll n=txt.size();
    vector<int> invSuff(n, 0);
    for (int i=0; i < n; i++)
        invSuff[SA[i]] = i;
    int k = 0;
    for (int i=0; i<n; i++)
    {
        if (invSuff[i] == n-1)
        {
            k = 0;
            continue;
        }
        int j = SA[invSuff[i]+1];
        while (i+k<n && j+k<n && txt[i+k]==txt[j+k])
            k++;
        lcp[invSuff[i]] = k; // lcp for the present suffix.
        if (k>0)
            k--;
    }
    return lcp;
}
//build_LCP(s)
// Driver program to test above functions
int main()
{
        ll t;
        cin >> t;
        while(t--)
        {
            memset(SA,0,sizeof(SA));
            memset(RA,0,sizeof(RA));
            memset(tempSA,0,sizeof(tempSA));
            memset(cnt,0,sizeof(cnt));
            memset(LCP,0,sizeof(LCP));
        string s;
        cin >> s;
        s+=(char)('$');
        // cout<<(ll)'$'<<endl;
        build_SA(s);
        build_LCP(s);
        ll x=-1;
        ll sum=0;
        lp(i,0,s.size())SA[i]--;
        s.pop_back();
        // lp(i,0,s.size())
        // {
            // cout<<SA[i]<<" ";
        // }

        lp(i,0,s.size())
        {
            sum+=lcp[i];
            // cout<<lcp[i]<<endl;
        }
        ll n=s.size();
        cout<<(n*n+n)/2-sum<<endl;
    }
}
```

20 suffix$_t ree$

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

#define fpos adla
const int inf = 1e9;
const int maxn = 1e4;
string s;
map<int, int> to[maxn];
int len[maxn], fpos[maxn], link[maxn];
int node, pos;
```

```
int node_num = 1, n = 0;

int make_node(int _pos, int _len)
{
    cerr<<node_num<<" "<<_pos<<" "<<_len<<endl;
    fpos[node_num] = _pos;
    len [node_num] = _len;
    return node_num++;
}

void go_edge()
{
    cout<<pos<<" "<<node<<" "<<s[n-pos]<<" "<<endl;
    while(pos > len[to[node][s[n - pos]]])
    {
        node = to[node][s[n - pos]];
        pos -= len[node];
        cout<<node<<" "<<s[n-pos]<<" "<<endl;
    }

}

void add_letter(int c)
{
    s+=(char)c;
    n++;
    cout<<s<<endl;
    pos++;
    int last = 0;
    while(pos > 0)
    {
        go_edge();
        cout<<"n "<<n<<" "<<pos<<" "<<s[n-pos]<<endl;
        int edge = s[n - pos];
        int &v = to[node][edge];
        int t = s[fpos[v] + pos - 1];
        cout<<pos<<"xxx"<<node<<" "<<(char)edge<<endl;
        cout<<v<<endl;
        if(v == 0)
        {

            cerr<<"root"<<endl;
            v = make_node(n - pos, inf);
            link[last] = node;
            last = 0;
        }
        else if(t == c)
        {
            cerr<<(char)c<<endl;
            link[last] = node;
            return;
        }
        else
        {
            int u = make_node(fpos[v], pos - 1);
            to[u][c] = make_node(n - 1, inf);
            to[u][t] = v;
            fpos[v] += pos - 1;
            len [v] -= pos - 1;
            v = u;
            link[last] = u;
            last = u;
        }
        if(node == 0)
            pos--;
        else
            node = link[node];
    }
}

int main()
{
    len[0] = inf;
    string s;
    cin >> s;
    int ans = 0;
    for(int i = 0; i < s.length(); i++)
    {
        add_letter(s[i]);
        cout<<endl;
    }
}
```

21 template$_a kshat$

```
// #pragma GCC optimize("Ofast")
// #pragma GCC optimize ("unroll-loops")
```

```
// #pragma GCC
    target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#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;
#define ll long long int
#define ld unsigned long long int
#define pi pair<ll,ll>
#define pb push_back
#define pf push_front
#define pu push
#define po pop
#define fi first
#define se second
#define mk make_pair
#define ve vector
#define lr(n) for(ll i=0;i<n;i++)
#define all(x) x.begin(),x.end()
#define be begin
#define sz(a) (ll)a.size()
#define INF 1e18
```

22 template$_{lad}$

```
#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 long long int ll;
typedef unsigned long long int ull;
typedef long double ld;
typedef pair <ll, ll> pll;
typedef pair <int, int> pii;
typedef tree <ll, null_type, less <ll>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
// order_of_key(val): returns the number of values less than val
// find_by_order(k): returns an iterator to the kth largest element
    (0-based)
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define all(a) a.begin(), a.end()
#define sz(a) (ll)(a.size())
#define endl "\n"
template <class Ch, class Tr, class Container>
basic_ostream <Ch, Tr> & operator << (basic_ostream <Ch, Tr> & os,
    Container const& x)
{
    os << "{ ";
    for(auto& y : x)
    {
        os << y << " ";
    }
    return os << "}";
}
template <class X, class Y>
ostream & operator << (ostream & os, pair <X, Y> const& p)
{
    return os << "[" << p.ff << ", " << p.ss << "]";
}
ll gcd(ll a, ll b)
{
    if(b==0)
    {
        return a;
    }
    return gcd(b, a%b);
}
ll modexp(ll a, ll b, ll c)
{
    a%=c;
    ll ans = 1;
    while(b)
    {
        if(b&1)
        {
            ans = (ans*a)%c;
        }

        a = (a*a)%c;
        b >>= 1;
    }
    return ans;
```

```
}
const ll L = 1e5+5;
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
}
```