Team notebook

greedy is good

December 24, 2019

Contents

1 Basics **1**
- 1.1 Dos and Donts . . . 1
- 1.2 Templates . . . 1
  - 1.2.1 Akshat . . . 1
  - 1.2.2 Lad . . . 1

2 DP **2**
- 2.1 Convex Hull . . . 2
- 2.2 LIS Using Segment Tree . . . 3
- 2.3 SOS DP . . . 4

3 Data Structures **4**
- 3.1 BIT . . . 4
- 3.2 Centroid Decomposition . . . 5
- 3.3 Merge Sort Tree . . . 7
- 3.4 Persistent Segment Tree . . . 8
- 3.5 SQRT Decomposition . . . 9
- 3.6 Segment Tree with Lazy Propagation . . . 10
- 3.7 Segment Tree . . . 10
- 3.8 Trie . . . 11
- 3.9 Wavelet Tree . . . 12

4 Graphs **12**
- 4.1 Basic Graph Algorithms . . . 12
- 4.2 Dinics Push Relabel EV^2 . . . 14
- 4.3 Dinics with Binary Search . . . 16
- 4.4 Kruskal's Algorithm . . . 17
- 4.5 LCA . . . 18
- 4.6 Min-Cost Max-Flow . . . 19

5 Hare Tortoise Mehod **20**

6 KMP **20**

7 Math and Number Theory **21**
- 7.1 Extended Euclidean . . . 21
- 7.2 FFT . . . 21
- 7.3 Matrix Exponentiation . . . 22
- 7.4 NTT . . . 22
- 7.5 Shoelace Formula . . . 23
- 7.6 Union of Rectangles . . . 23

8 Theory **24**

1 Basics

1.1 Dos and Donts

```
/* INSTRUCTIONS
1. Focus on the problem, Not on the Scoreboard (Specially Lad)
2. Review code before submitting. 2 min review << 20 min penalty
3. Watch out for overflows, out of bound errors
4. Stay Calm. Good Luck. Have Fun :) */

// Compiler Settings : alias g++=g++ -g -O2 -std=gnu++14 -Wall
```

1.2 Templates

1.2.1 Akshat

```
// #pragma GCC optimize("Ofast")
// #pragma GCC optimize ("unroll-loops")
// #pragma GCC
   target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#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
```

1.2.2 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);}
```

2 DP

2.1 Convex Hull

```
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;
```

2.2 LIS Using Segment Tree

```
int compare(pair<int, int> p1, pair<int, int> p2){
    if (p1.first == p2.first)
        return p1.second > p2.second;
    return p1.first < p2.first;
}
void buildTree(int* tree, int pos, int low, int high,int index,
    int value) {
    if (index < low || index > high)
        return;
    if (low == high) {
        tree[pos] = value;
        return;
    }
    int mid = (high + low) / 2;
    buildTree(tree, 2 * pos + 1, low, mid, index, value);
    buildTree(tree, 2 * pos + 2, mid + 1, high, index, value);
    tree[pos] = max(tree[2 * pos + 1], tree[2 * pos + 2]);
}
int findMax(int* tree, int pos, int low, int high, int start,
    int end) {
    if (low >= start && high <= end)
        return tree[pos];
    if (start > high || end < low)
        return 0;
    int mid = (high + low) / 2;
    return max(findMax(tree, 2 * pos + 1, low, mid, start, end),
               findMax(tree, 2 * pos + 2, mid + 1, high, start,
                   end));
}
int findLIS(int arr[], int n) {
    pair<int, int> p[n];
    for (int i = 0; i < n; i++) {
        p[i].first = arr[i];
        p[i].second = i;
    }
    sort(p, p + n, compare);
    int len = pow(2, (int)(ceil(sqrt(n))) + 1) - 1;
    int tree[len];
    memset(tree, 0, sizeof(tree));
    for (int i = 0; i < n; i++) {
        buildTree(tree, 0, 0, n - 1, p[i].second,
        findMax(tree, 0, 0, n - 1, 0, p[i].second) + 1);
    }
    return tree[0];
}
```

2.3 SOS DP

```
ll N = 1000;//small n is log2(number of elements in a)
void SumOverSubsets(ll a[], ll n){
        ll sos[1 << n]={0};
        ll dp[N][N];
        for(ll x=0; x<(1LL<<n); x++){
                for(ll i=0; i<n; i++){
                        if(x & (1 << i)){
                                if(i==0)
                                dp[x][i]=a[x]+a[x^(1<<i)];
                                else
                                dp[x][i]=dp[x][i-1]+dp[x^(1<<i)][i-1];
                        }
```

```
                else{ // i-th bit is not set
                    if (i == 0)
                        dp[x][i] = a[x];
                    else
                        dp[x][i] = dp[x][i-1];
                }
            }
            sos[x] = dp[x][n - 1];
        }
        for (ll i = 0; i < (1 << n); i++)
            cout << sos[i] << " ";
}
```

3 Data Structures

3.1 BIT

```
// 1-based 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, key2, key3, 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;
}
struct M Ssp[230000];
int main(){
        ll n;cin >> n; ll q;
        for (int i = 0; i < n; ++i){
                ll a; cin >> a;
                Ssp[i].key=a;
                Ssp[i].key2=Ssp[i].key3=0;
                Ssp[i].key4=i;
        }
        cin >> q;
        for (int i = 0; i < q; ++i){
                ll l,r,k; cin>>l>>r>>k;
                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);
        }
}
```

3.2 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);
        }
    }
}
```

3.3 Merge Sort Tree

```cpp
// 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;
    }
}
```

3.4 Persistent Segment Tree

```
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);
    }
```

```
};
```

3.5 SQRT Decomposition

```
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;
                }
        }
}
```

3.6 Segment Tree with Lazy Propagation

```
// SPOJ CNTPRIME // 1-based indexing
ll a[L]; ll seg[4*L]; ll lazy[4*L];
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));
}
```

3.7 Segment Tree

```
// 1-based indexing
ll a[L];
node seg[4*L];
void build(ll pos, ll tl, ll tr){
    if(tl == tr){
        seg[pos] = a[tl]; // Leaf Node
        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] = val; // Assign updated Value
        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]);
} // Query same as in Lazy Propagation
```

3.8 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);
        }
    }
}
```

3.9 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);
}
```

4 Graphs

4.1 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;
			}
		}
```

```
    }
}
```

4.2 Dinics Push Relabel EV^2

```
/*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";
}
```

4.3 Dinics with Binary Search

```
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;
}
```

4.4 Kruskal's Algorithm

```
ll find(ll s){
    if(parent[s]==s){
        return s;
    }
    return parent[s]=find(parent[s]);
}//Initialise parent[i] to i for each i
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];
}//Initialise unionSize[i] to 1 for each i
ll kruskals(ll M){
    ll ans = 0;//Sort weights first
    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;
}
```

4.5 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);
```

4.6 Min-Cost Max-Flow

```
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;
}
```

5 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 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--;
        }
    }
}
```

7 Math and Number Theory

7.1 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;
}
```

7.2 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;
} // Scan coefficients in reverse order cin >> a[n-i]
```

7.3 Matrix Exponentiation

```
typedef vector<vector<ll> > matrix;
matrix mul(matrix A, matrix B){
    matrix C(K, vector<ll>(K));
    lp(i,0, K) lp(j,0, K) lp(k,0, K)
        C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % mod;
    return C;
}
// Only Square Matrices
matrix pow(matrix A, ll p){
```

```
    if (p == 1)
        return A;
    if (p % 2)
        return mul(A, pow(A, p-1));
    matrix X = pow(A, p/2);
    return mul(X, X);
}
```

7.4 NTT

```
// 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) // standard modexp fn
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;
} // Input coefficients in reverse order
```

7.5 Shoelace Formula

```
double polygonArea(double X[], double Y[], int n) {
    double area = 0.0;
    int j = n - 1; // X and Y are coordinates of points
    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);
}
```

7.6 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)
// Standard Sum query with 1-based indexing
```

```
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;
}
```

8 Theory

```
Total number of spanning trees in a complete graph with n
    vertices is given by n^(n-2)
Sprague-Grundy Theorem: The losing states are exactly those with
    Grundy number equal to 0. Grundy number of the current state
    is the smallest whole number which is not the Grundy number
    of any state that can be reached in the next step.
    Mathematically, if s1, s2  sk are the game states directly
    reachable from s, Grundy(s)=min({0,1,...} -
    {Grundy(s1),Grundy(s2)Grundy(sk)})
Sums of Games: 1. Player chooses a game and makes a move in it.
    Grundy number of a position is xor of grundy numbers of
    positions in summed games. 2. Player chooses a non-empty
    subset of games (possibly, all) and makes moves in all of
    them. A position is losing iff each game is in a losing
    position. 3.Player chooses a proper subset of games (not
    empty and not all), and makes moves in all chosen ones. A
    position is losing iff grundy numbers of all games are
    equal. 4. Player must move in all games, and loses if cant
    move in some game. A position is losing if any of the games
    is in a losing position.
Misere Nim. A position with pile sizes a1, a2,..., an >= 1, not
    all equal to 1, is losing iff a1 xor a2 ... xor an = 0 (like
    in normal nim.) A position with n piles of size 1 is losing
    iff n is odd.
```