Team notebook

BRUR-Crackdown

August 11, 2021

Contents

1 Counting

1.1 NCR mod M

```
struct ncr_mod_m //Works only when m is prime
```

```
{
        ll fac[S+5], inv_fac[S+5];
        ll mod_expo(ll b,ll p,ll m)
        {
            ll ret=1,cur=b%m;
            while(p)
            {
                if(p & 1)
                {
                    ret=(cur*ret)%m;
                }
                cur=(cur*cur)%m;
                p=p >> 1;
            }
            return ret;
        }

        void init()
        {
          fac[0] = 1;
          for(int i=1;i<=S;i++)fac[i] = fac[i-1] * i % mod;
          inv_fac[S] = mod_expo(fac[S], mod - 2, mod);
          for(int i=S-1;i>=0;i--)inv_fac[i] = inv_fac[i+1] * (i + 1) % mod;
        }

        ll NCR(int n, int r)
        {
                if (n < r || r < 0) return 0;
                return (fac[n] * inv_fac[r] % mod) * inv_fac[n - r] % mod;
        }

}Comb;
```

1.2 NCR without Overflow

```
ll NCR(ll n, ll r)
{
    ll p = 1, k = 1;

    if (n - r < r)
        r = n - r;

    if (r != 0)
    {
        while (r)
        {
            p *= n;
            k *= r;
            ll m = __gcd(p, k);
            p /= m;
            k /= m;
            n--;
            r--;
        }
    }
    else p = 1;

    return p;
}
```

2 Data Structure

2.1 All Possible IS

```
typedef pair<ll,ll>pll;

const int S=1e5;
const int mod=1e9+7;

int N;
ll tr[S+5];
struct fenwick_tree_RSQ
{
        void init()
        {
                for(int i=0;i<=S;i++)tr[i]=0;
        }
        void update(int x, ll val)
        {
            for(;x<=N;x+=(x & (-x)))
            {
                        tr[x]=(tr[x]+val)%mod;
            }
        }
        ll query(int x)
        {
                ll sum=0;
                for(;x>0;x-=(x & (-x)))
                {
                        sum=(tr[x]+sum)%mod;
                }
                return sum;
        }
}FenTree;

void solve()
{
        FenTree.init();

        cin >> N;
        vector<pll>A;
        for(int i=1;i<=N;i++)
        {
                int x;
                cin >> x;
                A.push_back({x,i});
        }
        sort(all(A));

        ll ans=0,pre=1e15,sum=0;
        for(pll e:A)
        {
                ll cur=(FenTree.query(e.second-1)+1)%mod;
                if(e.first==pre)
                {
                        cur=(cur-sum+mod)%mod;
                        sum=(sum+cur)%mod;
                }
                else sum=cur;
                FenTree.update(e.second,cur);
                ans=(ans+cur)%mod;
                pre=e.first;
        }
        cout << ans << nl;
}

int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(nullptr);
        cout << fixed << setprecision(10);

        int T,cs=1;
        cin >> T;
        while(T--)
        {
                cout << "Case " << cs++ << ": ";
                solve();
        }
        return 0;
}
```

2.2 LCA

```
vector<int> vec[1003];
bool vis[1003];
int level[1003],parent[1003],SPT[1003][10];

void BFS(){
    memset(vis,false,sizeof(vis));
    memset(level,0,sizeof(level));
    memset(parent,-1,sizeof(parent));

    vis[1] = true;
    level[1] = 0;
    queue<int> Q;

    Q.push(1);

    while(!Q.empty()){
        int u = Q.front();
        Q.pop();
        for(auto v : vec[u]){
            if(!vis[v]){
                vis[v] = true;
                level[v] = level[u] + 1;
                Q.push(v);
                parent[v] = u;
            }
        }
    }
}

void sparse_table(int node){
    for(int i=1; i<=node; i++){
        SPT[i][0] = parent[i]; // 0-th parent of i-th node
    }
    for(int j=1; (1<<j)<=node; j++){
        for(int i = 1; i<=node; i++){
            if(SPT[i][j-1]!=-1)
                SPT[i][j] = SPT[SPT[i][j-1]][j-1];
        }
    }
}


void Precal(int node){
    BFS();
    sparse_table(node);
}


int find_LCA(int x, int y, int node){
    if(level[x]<level[y]) swap(x,y); // now level of x is bigger or equal
    for(int i = log2(node); i>=0; i--){
        if(level[x]-(1<<i) >= level[y]){
            x = SPT[x][i];
        }
    }
    // now x and y are in same level

    if(x==y) return x;

    for(int i = log2(node); i>=0; i--){
        if(parent[x]!=-1 and SPT[x][i]!=SPT[y][i]){
            x = SPT[x][i], y = SPT[y][i];
        }
    }
    return parent[x];
}
```

2.3 Lazy

```
int tr[4*S];
int lazy[4*S];
struct seg_tree_lazy_propagation
{
        void init()
        {
                for(int i=0;i<=4*S;i++)tr[i]=lazy[i]=0;
        }

        void propagate(int node, int s, int e)
        {
                //update by lazy[node]
                tr[node]+=(e-s+1)*;
                if(s!=e)
                {
                        lazy[2*node]=;
                        lazy[2*node+1]=;
                }
                lazy[node]=0;
        }

        void update(int node, int s, int e, int l, int r, int val)
        {
                if(lazy[node])propagate(node,s,e);
                if(s>e || s>r || e<l)return;
                if(s>=l && e<=r)
                {
                        //update by val
                        tr[node]+=(e-s+1)*;
                        if(s!=e)
                        {
                                lazy[2*node]=;
                                lazy[2*node+1]=;
                        }
                        return;
                }
                int m=(s+e)/2;
                update(2*node,s,m,l,r,val);
                update(2*node+1,m+1,e,l,r,val);

                //update by left and right child
                tr[node]=tr[2*node] tr[2*node+1];
        }

        int query(int node, int s, int e, int l, int r)
        {
                if(s>e || s>r || e<l)return 0;
                if(lazy[node])propagate(node,s,e);
                if(s>=l && e<=r)return tr[node];

                int m=(s+e)/2;
                return query(2*node,s,m,l,r) query(2*node+1,m+1,e,l,r);
        }

        //starts node number with 1, not with 0.

}SegTree;
```

2.4 MO

```
/*Mo starts*/
//everything is 0-indexed
const int blockSize=300;
struct moQuery
{
        int l,r,id;
}q[S+5];

bool cmp(moQuery a, moQuery b)
{
  if (a.l/blockSize != b.l/blockSize)return a.l<b.l;
  return (a.l/blockSize)%2 ? a.r>b.r : a.r<b.r;
}

void Add(int idx)
{
        modify
}

void Remove(int idx)
{
        modify
}
```

```
void Mo()
{
        int cl=0,cr=-1;
        int ans[Q+5];
        for(int i=0;i<Q;i++)
        {
                while(cr<q[i].r)Add(++cr);
                while(cl<q[i].l)Remove(cl++);
                while(cr>q[i].r)Remove(cr--);
                while(cl>q[i].l)Add(--cl);

                //calculate ans for every query
        }

        for(int i=0;i<Q;i++)cout << ans[i] << nl;
}
/*Mo ends*/
```

2.5 Number of Diameter

```
struct number_of_diameters
{
        int dia=0;

        //for finding number of diameters with two centers
        /*start----------------------------*/
        ll dfs_two(vector<int>adj[],vector<int>& dist,int u,int p)
        {
                ll sum=0;
                for(int v:adj[u])
                {
                        if(v!=p)
                        {
                                dist[v]=dist[u]+1;
                                sum+=dfs_two(adj,dist,v,u);
                        }
                }
                if(dist[u]==dia/2)return 1;
                return sum;
        }
```

```
ll two_center(vector<int>adj[],int nodes,int c1,int c2)
{
        vector<int>dist(nodes+1);
        dist[c1]=0;
        ll d1=dfs_two(adj,dist,c1,c2);

        dist[c2]=0;
        ll d2=dfs_two(adj,dist,c2,c1);
        return d1*d2;
}
/*end---------------------------*/

//for finding number of diameters with one center
/*start---------------------------*/
ll ans=0;
ll dfs_one(vector<int>adj[],vector<int>& dist,int c,int u,int p)
{
        ll sum=0;
        for(int v:adj[u])
        {
                if(v!=p)
                {
                        dist[v]=dist[u]+1;
                        ll cur=dfs_one(adj,dist,c,v,u);
                        if(u==c)
                        {
                                ans+=(sum*cur);
                        }
                        sum+=cur;
                }
        }
        if(dist[u]==dia/2)return 1;
        return sum;
}

ll one_center(vector<int>adj[],int nodes,int c)
{
        vector<int>dist(nodes+1);
        dist[c]=0;
        dfs_one(adj,dist,c,c,0);
        return ans;
}
/*end---------------------------*/
```

```
//for finding center(s) of tree
/*start---------------------------*/
set<int> convert(vector<int> vec)
{
        set<int>st(all(vec));
        return st;
}

pair<int,int> center(vector<int>temp[],int nodes)
{
        if(nodes==1)return {1,-1};
        set<int>adj[nodes+5];
        for(int i=1;i<=nodes;i++)adj[i]=convert(temp[i]);

        queue<int>q;
        int level[nodes+5];
        for(int i=1;i<=nodes;i++)
        {
                if(sz(adj[i])==1)q.push(i),level[i]=0;
        }

        int c1,c2;
        while(1)
        {
                int u=q.front();
                q.pop();

                int v=*adj[u].begin();
                adj[v].erase(u);

                if(sz(adj[v])==0)
                {
                        c1=v;
                        c2=u;
                        break;
                }
                if(sz(adj[v])==1)
                {
                        q.push(v);
                        level[v]=level[u]+1;
                }
        }

        if(level[c1]==level[c2])return {c1,c2};
        return {c1,-1};
    }
    /*end---------------------------*/

    //For finding tree diameter
    /*start---------------------------*/
    int dfs(vector<int>adj[],int u,int p)
    {
        int mx=0;
        for(int v:adj[u])
        {
            if(v!=p)
            {
                int cur=1+dfs(adj,v,u);
                dia=max(dia,mx+cur);
                mx=max(mx,cur);
            }
        }
        return mx;
    }
    void diameter(vector<int>adj[])
    {
        dfs(adj,1,0);
    }
    /*end---------------------------*/

    ll count_diameters(vector<int>adj[],int nodes)
    {
        diameter(adj);
        int c1,c2;
        tie(c1,c2)=center(adj,nodes);
        if(c2==-1)return one_center(adj,nodes,c1);
        return two_center(adj,nodes,c1,c2);
    }

}Tree;
```

2.6 RMQ

```
int tr[4*S+5];
struct seg_tree_RMQ
{
    void init()
    {
        for(int i=0;i<=4*S;i++)tr[i]=0;
    }
    void update(int node, int s, int e, int idx, int val)
    {
        if(s>e || s>idx || e<idx)return;
        if(s>=idx && e<=idx)
        {
            tr[node]=; //update by val
            return;
        }
        int m=(s+e)/2;
        update(2*node,s,m,idx,val);
        update(2*node+1,m+1,e,idx,val);
        tr[node]=min(tr[2*node],tr[2*node+1]);
    }
    int query(int node, int s, int e, int l, int r)
    {
        if(s>e || s>r || e<l)return inf;
        if(s>=l && e<=r)return tr[node];
        int m=(s+e)/2;
        return min(query(2*node,s,m,l,r),query(2*node+1,m+1,e,l,r));
    }

}SegTree;
```

2.7 Super Frequent Element

```
const int S=3e5;
vector<int>Vid[S+5];

int occurrence(int l,int r,int val)
{
    return upper_bound(all(Vid[val]),r)-lower_bound(all(Vid[val]),l);
}

/*seg_tree_RMQ starts*/
struct Node
{
    int x,cntx;
}tr[4*S+5];
```

```
struct seg_tree_RMQ
{
        void init()
        {
                for(int i=0;i<=4*S;i++)tr[i].x=tr[i].cntx=0;
        }
        void update(int node, int s, int e, int idx, int val)
        {
                if(s>e || s>idx || e<idx)return;
                if(s>=idx && e<=idx)
                {
                        tr[node].x=val; //update by val
                        tr[node].cntx=1;
                        return;
                }
                int m=(s+e)/2;
                update(2*node,s,m,idx,val);
                update(2*node+1,m+1,e,idx,val);

                if(tr[2*node].x == tr[2*node+1].x)
                {
                        tr[node].x=tr[2*node].x;
                        tr[node].cntx=tr[2*node].cntx+tr[2*node+1].cntx;
                }
                else if(tr[2*node].cntx > tr[2*node+1].cntx)
                {
                        tr[node].x=tr[2*node].x;
                        tr[node].cntx=tr[2*node].cntx-tr[2*node+1].cntx;
                }
                else
                {
                        tr[node].x=tr[2*node+1].x;
                        tr[node].cntx=tr[2*node+1].cntx-tr[2*node].cntx;
                }
        }
        int query(int node, int s, int e, int l, int r)
        {
                if(s>e || s>r || e<l)return 0;
                if(s>=l && e<=r)return occurrence(l,r,tr[node].x);
                int m=(s+e)/2;
                return max(query(2*node,s,m,l,r),query(2*node+1,m+1,e,l,r));
        }

}SegTree;
```

```
/*seg_tree_RMQ ends*/

int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(nullptr);
        cout << fixed << setprecision(10);

        int N,Q;
        while(cin >> N >> Q)
        {
                SegTree.init();
                for(int i=0;i<=S;i++)Vid[i].clear();

                int A[N+5];
                for(int i=1;i<=N;i++)
                {
                        cin >> A[i];
                        SegTree.update(1,1,N,i,A[i]);
                        Vid[A[i]].push_back(i);
                }

                while(Q--)
                {
                        int l,r;
                        cin >> l >> r;
                        int f=SegTree.query(1,1,N,l,r);
                        int len=r-l+1;
                        cout << max(1,2*f-len) << nl;
                }
        }
        return 0;
}
```

2.8 Tree Center

```
struct tree_center
{
        set<int> convert(vector<int> vec)
        {
                set<int>st(all(vec));
                return st;
        }

        pair<int,int> center(vector<int>temp[],int nodes)
        {
                if(nodes==1)return {1,-1};
                set<int>adj[nodes+5];
                for(int i=1;i<=nodes;i++)adj[i]=convert(temp[i]);

                queue<int>q;
                int level[nodes+5];
                for(int i=1;i<=nodes;i++)
                {
                        if(sz(adj[i])==1)q.push(i),level[i]=0;
                }

                int c1,c2;
                while(1)
                {
                        int u=q.front();
                        q.pop();

                        int v=*adj[u].begin();
                        adj[v].erase(u);

                        if(sz(adj[v])==0)
                        {
                                c1=v;
                                c2=u;
                                break;
                        }
                        if(sz(adj[v])==1)
                        {
                                q.push(v);
                                level[v]=level[u]+1;
                        }
                }

                if(level[c1]==level[c2])return {min(c1,c2),max(c1,c2)};
                return {c1,-1};
        }

}Tree;
```

2.9 Tree Diameter1

```
struct tree_diameter
{
        int dia;
        int dfs(vector<int>adj[],int u,int p)
        {
                int mx=0;
                for(int v:adj[u])
                {
                        if(v!=p)
                        {
                                int cur=1+dfs(adj,v,u);
                                dia=max(dia,mx+cur);
                                mx=max(mx,cur);
                        }
                }
                return mx;
        }
        int diameter(vector<int>adj[])
        {
                dia=0;
                dfs(adj,1,0);
                return dia;
        }
}Tree;
```

2.10 Tree Diameter2

```
struct tree_diameter
{
        void dfs(vector<int>adj[],vector<int>& dist,int u,int p)
        {
                for(int v:adj[u])
                {
                        if(v!=p)
                        {
                                dist[v]=dist[u]+1;
                                dfs(adj,dist,v,u);
                        }
                }
        }
        int diameter(vector<int>adj[],int nodes)
        {
                vector<int>dist(nodes+1);
                dist[1]=0;
                dfs(adj,dist,1,0);
                int u=max_element(1+all(dist))-dist.begin();
                dist[u]=0;
                dfs(adj,dist,u,0);
                return *max_element(1+all(dist));
        }
}Tree;
```

3 Graph

3.1 0-1 BFS

```
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

3.2 Bellman Ford

```
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    for (;;)
    {
        bool any = false;

        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] + e[j].cost;
                    any = true;
                }

        if (!any) break;
    }
    // display d, for example, on the screen
}

//path
void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n, -1);

    for (;;)
    {
        bool any = false;
        for (int j = 0; j < m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] + e[j].cost;
                    p[e[j].b] = e[j].a;
                    any = true;
                }
        if (!any) break;
    }

    if (d[t] == INF)
        cout << "No path from " << v << " to " << t << ".";
    else
    {
        vector<int> path;
        for (int cur = t; cur != -1; cur = p[cur])
            path.push_back (cur);
        reverse (path.begin(), path.end());

        cout << "Path from " << v << " to " << t << ": ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
```

3.3 Bipartite

```
int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v]) {
                if (side[u] == -1) {
                    side[u] = side[v] ^ 1;
                    q.push(u);
                } else {
                    is_bipartite &= side[u] != side[v];
                }
            }
        }
    }
}

cout << (is_bipartite ? "YES" : "NO") << endl;
```

3.4 Cycle Detection

```
bool white[S+5],grey[S+5],black[S+5]; //initialize in case of multiple test
    cases
struct graph_cycle
{
    bool cycle(int u)
    {
        white[u]=false;
        grey[u]=true;

        int siz=graph[u].size();
        for(int j=0;j<siz;j++)
        {
            int v=graph[u][j];

            if(black[v])continue;
            if(grey[v])return true;
            if(cycle(v))return true;
        }

        grey[u]=false;
        black[u]=true;
        return false;
    }
}Graph;
```

3.5 Dijkstra

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);
    vector<bool> u(n, false);

    d[s] = 0;
    for (int i = 0; i < n; i++) {
        int v = -1;
        for (int j = 0; j < n; j++) {
            if (!u[j] && (v == -1 || d[j] < d[v]))
                v = j;
        }

        if (d[v] == INF)
            break;

        u[v] = true;
        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
            }
        }
    }
}

//path
vector<int> restore_path(int s, int t, vector<int> const& p) {
    vector<int> path;

    for (int v = t; v != s; v = p[v])
        path.push_back(v);
    path.push_back(s);

    reverse(path.begin(), path.end());
    return path;
}
```

3.6 Floyd Warshal

```
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}
```

3.7 MST Kruskal

```
vector<int>rt(S+5),sv(S+5);
struct tree_kruskal
{
        int root(int x)
        {
                return (x==rt[x]) ? x : (rt[x] = root(rt[x]));
        }

        void make_union(int a,int b)
        {
                a=root(a), b=root(b);
                if(a!=b)
                {
                        if(sv[a]>sv[b])swap(a,b);
                        sv[a]+=sv[b];
                        rt[b]=a;
                }
        }

        ll kruskal()
        {
                for(int i=0;i<=S;i++)rt[i]=i,sv[i]=1;

                ll cost=0;
                sort(all(edge));
                for(int i=0;i<sz(edge);i++)
                {
                        int w,u,v;
                        tie(w,u,v)=edge[i];
                        if(root(u)!=root(v))
                        {
                                make_union(u,v);
                                cost+=w;
                        }
                }
                return cost;
        }
}Tree;
```

3.8 SCC

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;

    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);

    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...

    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }

    used.assign(n, false);

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);

    used.assign(n, false);
    reverse(order.begin(), order.end());

    for (auto v : order)
        if (!used[v]) {
            dfs2 (v);

            // ... processing next component ...

            component.clear();
        }
}
```

3.9 Topological Sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

4 MISC

4.1 Morsa

```
int Z[100010];                          ///Z
void Z_Algo(string ch)
{
    int pos, st, endd;
    int l = ch.size();
    Z[0] = l ;
    for(pos=1, st=0,endd=0; pos<l; pos++)
    {
        if( pos <= endd )
            Z[pos] = min(endd-pos+1, Z[pos-st]);
        while( pos+Z[pos]<l && ch[Z[pos]]==ch[pos+Z[pos]])
            ++Z[pos] ;
        if(pos+Z[pos]-1 > endd) /// ekhane update korte hobe
            st = pos, endd = pos + Z[pos]-1;
    }
}

string make_string(string s)             /// manachar
{
    int l = s.size();
    string st;
    st += "^#";
    for(int i=0; i<l; i++)
    {
        st += s[i];
        st += '#';
    }
    st += "$";
    return st; /// notun length = 2*len+3
}
int man[3000100];
inline string manacher(string str)
{
    string ch = make_string(str);
    //cout<<ch<<endl;
    int n = ch.size();
    int left = 0, right = 0;
    for(int i=1; i<=n; i++)
    {
        int i_mirror = left - (i-left);
        if(right>i) man[i] = min(right-i, man[i_mirror]);
        else man[i] = 0;

        while((i+1+man[i]) <= n && (i-1-man[i]) >= 0 && ch[i+1+man[i]] ==
             ch[i-1-man[i]])
            man[i]++;

        if(i + man[i] > right)
        {
            left = i;
            right = i + man[i];
        }
    }

    int mx = 0;
    int center = 0;
    for(int i = 0; i <= n; i++)
    {
        if(man[i] > mx)
        {
            mx = man[i];
            center = i;
        }
    }
    //cout<<mx<<endl;
    return str.substr((center-mx-1)/2, mx);
}

int table[1010][1010];                           ///LCS length and print lcs
int lcs_length(string &text1, string &text2)
{
    int l = text1.size();
    int ll = text2.size();
    for(int i=0; i<=l; i++) table[i][0] = 0;
    for(int i=0; i<=ll; i++) table[0][i] = 0;
    for(int i=1; i<=l; i++)
    {
        for(int j=1; j<=ll; j++)
        {
            if(text1[i-1]==text2[j-1])
                table[i][j] = table[i-1][j-1]+1;
            else
                table[i][j] = max(table[i-1][j], table[i][j-1]);
        }
    }
    for(int i=0; i<=l; i++)
    {
        for(int j=0; j<=ll; j++)
        {
            printf("%d ",table[i][j]);
        }
        cout<<endl;
    }
    return table[l][ll];
}

string lcs_print(string &text1, string &text2)
{
    int l = text1.size();
    int ll = text2.size();
    string s;
    s.clear();
    while(table[l][ll])
    {
        if(table[l-1][ll]<table[l][ll] && table[l][ll-1]<table[l][ll])
            printf("%d %d\n",l, ll), s += text1[l-1], l--, ll--;
        else if(table[l-1][ll]==table[l][ll] && table[l][ll-1]==table[l][ll])
            l--; //ekhane upore jassi 1 ghor. issa korle 1 ghor
        //left eo jawa jeto.
        else if(table[l-1][ll]==table[l][ll]) l--;
        else if(table[l][ll-1]==table[l][ll]) ll--;
    }
    reverse(s.begin(), s.end());
    return s;
}

vector<int> tab;                   ///LIS nlogn solution
int *dp = new int[1000100];
int LIS(vector<int> vec, int n)
{
    dp[0] = 1;
    tab.pb(vec[0]);
    int lis_length = 1; /// etai amar LIS er length hobe.
    for(int i=1; i<n; i++)
    {
        if(vec[i]>tab.back())
        {
            tab.pb(vec[i]);
            lis_length++;
            dp[i] = lis_length;
        }
        else
        {
            vector<int> :: iterator it;
            it = lower_bound(tab.begin(), tab.end(), vec[i]);
            *it = vec[i];
            dp[i] = (it-tab.begin()) + 1;
        }
    }
    return lis_length;
}
vector<int> ans;
void print_LIS(vector<int> vec, int n, int Lis_length)
{
    for(int i=n-1; i>=0; i--)
    {
        if(dp[i]==Lis_length)
        {
            ans.pb(vec[i]);
            Lis_length--;
        }
    }
    reverse(ans.begin(), ans.end());
    for(auto i: ans)
        pi(i), nl;
}

int kadanes(int n)                        ///Kadanes with indexes
{
    int mx = ara[0];
    int start = 0;
    int endd = 0;
    int mstart = 0;
    int mend = 0;
    int cur_max = ara[0];
    fr(i, 1, n)
    {
        if(ara[i]>ara[i]+cur_max)
            cur_max = ara[i], start = i;
        else
            cur_max += ara[i];
        endd = i;
        if(cur_max>mx)
        {
            mx = cur_max;
            mstart = start;
            mend = endd;
        }
    }
    int val = mx;
    fr(i, mend+1, n)
    {
        val += ara[i];
        if(val>=mx) mend = i, mx = val;
    }

//    printf("%d %d %d\n", mstart, mend, mx);
    return mx;
}

int phi[100010];

void calculatePhi()                         ///Eular Phi
{
    for(int i=2; i<=100010; i++)
        phi[i] = i;
    for(int i =2; i<=100010; i++)
    {
        if(phi[i]==i)
        {
            for(int j=i; j<=100010; j+=i)
                phi[j]-=phi[j]/i;
        }
    }
}

inline int range_hash(int l, int r)                       ///Hashing
{
    /// main funcion e range() funcion ta call dewar age
    /// modulas_inverse() function ta call dite hobe
    int val = (hashh[r + 1] - hashh[l]) * 1LL * inv[l] % mod;
    if (val < 0) val += mod;
    return val;
}
void makehashh_array(string tun)
{
    int n = tun.size();
    int power = 1;
    for (int i = 0; i < n; ++i)
    {
        hashh[i + 1] = (hashh[i] + power * 1LL * (tun[i] - 96)) % mod;
        power = power * 1LL * BASE % mod;
    }
}
ll bigmod (ll a, ll e)
{
    if (e == -1) e = mod - 2;
    ll ret = 1;
    while (e)
    {
        if (e & 1) ret = ret * 1LL * a % mod;
        a = a * 1LL * a % mod, e >>= 1;
    }
    return ret;
}
void modulas_inverse()
{
    inv[0] = 1;
    inv[1] = bigmod(BASE, -1);
    cout<<inv[1], nl;
    for (int i = 2; i < NUM; ++i)
        inv[i] = inv[i - 1] * 1LL * inv[1] % mod;
//    for (int i = 2; i < 10; ++i)
//        printf("%d = %d\n", i, inv[i]);
}
```

4.2 Sumon1

```
//result could intersercts in the range
struct info{
    int total,prefix,suffix,result;
}tree[200005];

void makeTree(int node, int L, int R)
{
    if(L==R){
        tree[node].total = arr[L];
        tree[node].prefix = arr[L];
        tree[node].suffix = arr[L];
        tree[node].result = arr[L];
        return;
    }

    int mid = L + (R-L)/2;
    int left = 2*node;
    int right = 2*node + 1;

    makeTree(left,L,mid);
    makeTree(right,mid+1,R);

    tree[node].total = tree[left].total + tree[right].total;
    tree[node].prefix =
        max(tree[left].prefix,tree[left].total+tree[right].prefix);
    tree[node].suffix =
        max(tree[right].suffix,tree[right].total+tree[left].suffix);
    tree[node].result = max(tree[left].suffix+tree[right].prefix,
        max(tree[left].result,tree[right].result));
}


info Query(int node, int L, int R, int i, int j)
{
    if(L>j or R<i) return {-15008,-15008,-15008,-15008}; // out_of_segment
    if(i<=L and R<=j) return tree[node]; // relevent_segment

    int mid = L + (R-L)/2;
    int left = 2*node;
    int right = 2*node + 1;

    info temp,leftTree,rightTree;

    leftTree = Query(left,L,mid,i,j);
    rightTree = Query(right,mid+1,R,i,j);

    temp.total = leftTree.total + rightTree.total;
    temp.prefix = max(leftTree.prefix, leftTree.total+rightTree.prefix);
    temp.suffix = max(rightTree.suffix, rightTree.total+leftTree.suffix);
    temp.result = max(leftTree.suffix+rightTree.prefix,
        max(leftTree.result,rightTree.result));

    return temp;
}

void Update(int node, int L, int R, int pos, int val)
{
    if(pos<=L and R<=pos){
        tree[node].total = val;
        tree[node].prefix = val;
        tree[node].suffix = val;
        tree[node].result = val;
        return;
    }

    if(R<pos or pos<L) return;

    int mid = L + (R-L)/2;
    int left = 2*node;
    int right = 2*node + 1;

    Update(left,L,mid,pos,val);
    Update(right,mid+1,R,pos,val);

    tree[node].total = tree[left].total + tree[right].total;
    tree[node].prefix =
        max(tree[left].prefix,tree[left].total+tree[right].prefix);
    tree[node].suffix =
        max(tree[right].suffix,tree[right].total+tree[left].suffix);
    tree[node].result =
        max(tree[left].suffix+tree[right].prefix,max(tree[left].result,tree[right].res
}
```

4.3 Sumon2

```
//SegmentTree with Propagation
struct info
{
  ll sum,lazy;
}tree[400005];

void Propagation(int node, int l, int r){
  tree[node].sum += (r-l+1)*tree[node].lazy;
  if(l!=r){
    tree[2*node].lazy += tree[node].lazy;
    tree[2*node+1].lazy += tree[node].lazy;
  }
  tree[node].lazy = 0;
}


void makeTree(int node, int l, int r){
  if(l==r){
    tree[node] = {0,0};
    return;
  }
  int m = l+(r-l)/2;
  int left = 2*node;
  int right = 2*node+1;

  makeTree(left,l,m);
  makeTree(right,m+1,r);

  tree[node] = {0,0};
}

void UPDATE(int node, int l , int r, int x, int y, int val){
  if(tree[node].lazy){
    Propagation(node,l,r);
  }
  if(r<x or y<l) return;
  if(x<=l and r<=y){
    tree[node].lazy += val;
    Propagation(node,l,r);
    return;
  }
  int m = l+(r-l)/2;
  int left = 2*node;
  int right = 2*node+1;

  UPDATE(left,l,m,x,y,val);
  UPDATE(right,m+1,r,x,y,val);

  tree[node].sum = tree[left].sum + tree[right].sum;
}

ll QUERY(int node, int l, int r, int x, int y){
  if(tree[node].lazy){
    Propagation(node,l,r);
  }
  if(r<x or y<l) return 0;
  if(x<=l and r<=y){
    return tree[node].sum;
  }
  int m = l+(r-l)/2;
  int left = 2*node;
  int right = 2*node+1;

  return QUERY(left,l,m,x,y) + QUERY(right,m+1,r,x,y);
}
```

4.4 Temp

```
#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 nl ’\n’
#define mem(x,val) memset(x,val,sizeof x)
#define all(x) (x).begin(),(x).end()
#define rall(x) (x).rbegin(),(x).rend()
#define sz(x) (int)(x).size()
#define vt vector

#define sim template < class c
#define ris return * this
#define dor > debug & operator <<
#define eni(x) sim > typename \
       enable_if<sizeof dud<c>(0) x 1, debug&>::type operator<<(c i) {
sim > struct rge { c b, e; };
sim > rge<c> range(c i, c j) { return rge<c>{i, j}; }
sim > auto dud(c* x) -> decltype(cerr << *x, 0);
sim > char dud(...);
struct debug {
#ifdef LOCAL
~debug() { cerr << endl; }
eni(!=) cerr << boolalpha << i; ris; }
eni(==) ris << range(begin(i), end(i)); }
```

```
sim, class b dor(pair < b, c > d) {
        ris << "(" << d.first << ", " << d.second << ")";
}
sim dor(rge<c> d) {
        *this << "[";
        for (auto it = d.b; it != d.e; ++it)
                *this << ", " + 2 * (it == d.b) << *it;
        ris << "]";
}
#else
sim dor(const c&) { ris; }
#endif
};
#define imie(...) " [" << #__VA_ARGS__ ": " << (__VA_ARGS__) << "] "

//typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> new_data_set;
//mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
typedef long long ll;



int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(nullptr);
        cout << fixed << setprecision(10);


        return 0;
}
```

5 Number Theory

5.1 Big Mod

```
ll mod_expo(ll b,ll p,ll m)
{
    ll ret=1,cur=b%m;
    while(p)
    {
        if(p & 1)
        {
            ret=(cur*ret)%m;
        }
        cur=(cur*cur)%m;
        p=p >> 1;
    }
    return ret;
}
```

5.2 Divisor (Fast)

```
// C++ program to count distinct divisors
// of a given number n
#include <bits/stdc++.h>
using namespace std;

void SieveOfEratosthenes(int n, bool prime[],
                       bool primesquare[], int a[])
{
    // Create a boolean array "prime[0..n]" and
    // initialize all entries it as true. A value
    // in prime[i] will finally be false if i is
    // Not a prime, else true.
    for (int i = 2; i <= n; i++)
        prime[i] = true;

    // Create a boolean array "primesquare[0..n*n+1]"
    // and initialize all entries it as false. A value
    // in squareprime[i] will finally be true if i is
    // square of prime, else false.
    for (int i = 0; i <= (n * n + 1); i++)
        primesquare[i] = false;

    // 1 is not a prime number
    prime[1] = false;

    for (int p = 2; p * p <= n; p++) {
        // If prime[p] is not changed, then
        // it is a prime
        if (prime[p] == true) {
            // Update all multiples of p
            for (int i = p * 2; i <= n; i += p)
                prime[i] = false;
        }
    }

    int j = 0;
    for (int p = 2; p <= n; p++) {
        if (prime[p]) {
            // Storing primes in an array
            a[j] = p;

            // Update value in primesquare[p*p],
            // if p is prime.
            primesquare[p * p] = true;
            j++;
        }
    }
}

// Function to count divisors
int countDivisors(int n)
{
    // If number is 1, then it will have only 1
    // as a factor. So, total factors will be 1.
    if (n == 1)
        return 1;

    bool prime[n + 1], primesquare[n * n + 1];

    int a[n]; // for storing primes upto n

    // Calling SieveOfEratosthenes to store prime
    // factors of n and to store square of prime
    // factors of n
    SieveOfEratosthenes(n, prime, primesquare, a);

    // ans will contain total number of distinct
    // divisors
    int ans = 1;

    // Loop for counting factors of n
    for (int i = 0;; i++) {
        // a[i] is not less than cube root n
        if (a[i] * a[i] * a[i] > n)
            break;

        // Calculating power of a[i] in n.
        int cnt = 1; // cnt is power of prime a[i] in n.
        while (n % a[i] == 0) // if a[i] is a factor of n
        {
            n = n / a[i];
            cnt = cnt + 1; // incrementing power
        }

        // Calculating the number of divisors
        // If n = a^p * b^q then total divisors of n
        // are (p+1)*(q+1)
        ans = ans * cnt;
    }

    // if a[i] is greater than cube root of n

    // First case
    if (prime[n])
        ans = ans * 2;

    // Second case
    else if (primesquare[n])
        ans = ans * 3;

    // Third case
    else if (n != 1)
        ans = ans * 4;

    return ans; // Total divisors
}

// Driver Program
int main()
{
    cout << "Total distinct divisors of 100 are : "
         << countDivisors(100) << endl;
    return 0;
}
```

5.3 Factorization (Frequency)

```
vector<pair<ll, ll>>prime_factors_with_frequency(ll B)
{
    vector<pair<ll, ll>>freq;
    for(int i=2;i<=sqrt(B);i++)
    {
        if(B%i==0)
        {
                ll cnt=0;
                while(B%i==0)
                {
                        B/=i;
                        cnt++;
                }
                freq.push_back({cnt,i});
        }
    }
    if(B!=1)freq.push_back({1,B});
    sort(rall(freq));

    return freq;
}
```

5.4 Factorization (SPF)

```
int spf[S+5];
void smallest_prime_factor()
{
        for(int i=1;i<=S;i++)spf[i]=i;
        for(int i=2;i<=S;i+=2)spf[i]=2;
        for(int i=3;i<=sqrt(S);i+=2)
        {
                if(spf[i]==i)
                {
                        for(int j=i*i;j<=S;j+=i)
                        {
                                if(spf[j]==j)spf[j]=i;
                        }
                }
        }
}
```

5.5 Inverse Factorial

```
ll fac[S+5];
ll invFac[S+5];

ll bigMod(ll b,ll p)
{
    ll ret=1;
    while(p)
    {
        if(p&1)ret=(ret*b)%mod;
        b=(b*b)%mod;
        p>>=1;
    }
    return ret;
}

void calFac()
{
    fac[0]=1;
    for(ll i=1;i<=S;i++)fac[i]=(fac[i-1]*i)%mod;

    invFac[S]=bigMod(fac[S],mod-2);

    for(ll i=S-1;i>=0;i--)invFac[i]=((i+1)*invFac[i+1])%mod;

}
```

5.6 Inverse Mod

```
//Using extended gcd
struct mod_inverse
{
        ll ext_gcd(ll a, ll b, ll &x, ll &y)
        {
            if (b == 0)
            {
                x = 1;
                y = 0;
                return a;
            }
            ll x1, y1;
            ll d = gcd(b, a % b, x1, y1);
            x = y1;
            y = x1 - y1 * (a / b);
            return d;
        }

        ll inverse(ll a,ll m)
        {
                ll x,y;
                ll g=ext_gcd(a,m,x,y);
                if(g!=1)return -1;
                return (x%m+m)%m;
        }

}Mod;

//Using Fermet little Theorem
ll x = mod_expo( a, mod - 2, mod );

//Finding the modular inverse for every number modulo M
inv[1] = 1;
for(int i = 2; i < m; ++i)
    inv[i] = m - (m/i) * inv[m%i] % m;
```

5.7 Pairs Forming LCM

```
const int S=10000007;
const int P=664579;
bool mark[S+5];
int prime[P+5];
int id=0;

void sieve()
{
    for(int i=2;i<=sqrt(S);i++)
    {
        if(!mark[i])
        {
            for(int j=i*i;j<=S;j+=i)mark[j]=true;
        }
    }
    for(int i=2;i<=S;i++)if(!mark[i])prime[id++]=i;
}

int main()
{
    //#ifndef ONLINE_JUDGE
    //FI;
    //FO;
    //#endif
    //ios_base::sync_with_stdio(0); cin.tie(0);
    sieve();
    int test,cas=1;
    si(test);
    while(test--)
    {
        ll n;
        sl(n);
        ll x=n;
        ll ans=1;
        for(int i=0;i<id && prime[i]<=sqrt(n);i++)
        {
            ll p=prime[i];
            if(x%p==0)
            {
                int cnt=0;
                while(x%p==0)x/=p,cnt++;
                ans*=(2*cnt+1);
            }
        }
        if(x>1)ans*=3;
        ans++;
        pf("Case %d: %lld\n",cas++,ans/2);
    }
    return 0;
}
```

5.8 SOD

```
for(ll i=2;i<=sqrt(n);i++)
```

```
{
    ///sequence starts at i and ends at n/i
    ll s=i;
    ll e=n/i;

    ///summation of sequence from 1 to (s-1)
    s--;
    s=(s*(s+1))/2;

    ///summation of sequence from 1 to e
    e=(e*(e+1))/2;

    ///hence, summation of sequence from s to e is (e-s)
    sum+=(e-s);

    ///straight line sum
    sum+=((n/i)-i)*i;
}
```

5.9 Totient

```
void cal_phi()
{
        for(int i=0;i<=S;i++)phi[i]=i;
        phi[1]=0;
        for(int i=2;i<=S;i++)
        {
                if(phi[i]==i)
                {
                        for(int j=i;j<=S;j+=i)
                        {
                                phi[j]-=phi[j]/i;
                        }
                }
        }
}
```

6 String

6.1 Hashing

```
/*5 different base for 5 different hashing. This will reduce the
collison probability significantly*/
vt<ll> a1={6, 9, 69, 168, 16969}, b1={9, 169, 96, 696969, 9696969};
vt<ll> a2={6339285, 9352, 6945, 16358, 12345969}, b2={68479, 13769, 96,
    6936969, 96965969};

vt<ll> evaluate(string &S,int &pos)
{
    if(S[pos]==’(’)
    {
        pos++;
        vt<ll> lhs=evaluate(S,pos);
        char op=S[pos++];
        vt<ll> rhs=evaluate(S,pos);
        pos++;
        if(op==’+’)
        {
            for(int i=0;i<5;i++)lhs[i]=(lhs[i]+rhs[i])%mod;
        }
        else if(op==’*’)
        {
            for(int i=0;i<5;i++)lhs[i]=(lhs[i]*rhs[i])%mod;
        }
        else
        {
            for(int i=0;i<5;i++)
            {
                // hash function
                // ((a1 * lhs[i] + b1) ^ (a2 * rhs[i] + b2))%M
                lhs[i]=(lhs[i]*a1[i]+b1[i])%mod;
                rhs[i]=(rhs[i]*a2[i]+b2[i])%mod;
                lhs[i]^=rhs[i];
                lhs[i]%=mod;
            }
        }
        return lhs;
    }
    else
    {
        ll num=0;
        while(S[pos]>='0' && S[pos]<='9')
        {
            num=(num*10+S[pos]-'0')%mod;
            pos++;
        }
        vt<ll> ret;
        for(int i=0;i<5;i++)ret.push_back(num);
        return ret;
    }
}

void solve()
{
    int N; cin >> N;
    vt<ll>ans;
    map<vt<ll>,int>mp;
    for(int i=0;i<N;i++)
    {
        string S; cin >> S;
        int pos=0;
        vt<ll> V=evaluate(S,pos);
        if(mp.find(V)==mp.end())
        {
            mp.insert({V,sz(mp)+1});
        }
        ans.push_back(mp[V]);
    }
    for(ll e:ans)cout << e << " ";
    cout << nl;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout << fixed << setprecision(10);

    int T,cs=1;
    cin >> T;
    while(T--)
    {
        cout << "Case #" << cs++ << ": ";
        solve();
    }
    return 0;
}
```

6.2 KMP

```
//You are given a string s of length n. The prefix function for this string is
    defined as an array  of length n, where [i] is the length of the longest
    proper prefix of the substring s[0i] which is also a suffix of this
    substring. A proper prefix of a string is a prefix that is not equal to
    the string itself. By definition, [0]=0.

vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

6.3 Trie

```
const int csz=26;//update by number of possible distinct character
struct Node
{
        bool finish;
        Node *next[csz+5];
        Node()
        {
                finish=false;
                for(int i=0;i<csz;i++)next[i]=NULL;
        }
}*Root;
```

```
void insertion(string const& s)
{
        Node *cur=Root;
        for(int i=0;i<sz(s);i++)
        {
                int id=s[i]-'0';
                if(cur->next[id]==NULL)
                {
                        cur->next[id]=new Node();
                }
                cur=cur->next[id];
        }
        cur->finish=true;
}

bool searching(string const& s)
{
        Node *cur=Root;
        for(int i=0;i<sz(s);i++)
        {
                int id=s[i]-'0';
                if(cur->next[id]==NULL)return false;
                cur=cur->next[id];
        }
        if(cur->finish)
        {
                for(int i=0;i<csz;i++)
                {
                        if(cur->next[i])return true;
                }
        }
        return false;
}

void deletion(Node *cur)
{
        for(int i=0;i<csz;i++)
        {
                if(cur->next[i])deletion(cur->next[i]);
        }
        delete(cur);
}
//declare Root=new Node(); at the beginning of every test case
//declare deletion() at the end of every test case
```

6.4 Unique substring

```
int count_unique_substrings(string const& s) {
    int n = s.size();

    const int p = 31;
    const int m = 1e9 + 9;
    vector<long long> p_pow(n);
    p_pow[0] = 1;
    for (int i = 1; i < n; i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i+1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;

    int cnt = 0;
    for (int l = 1; l <= n; l++) {
        set<long long> hs;
        for (int i = 0; i <= n - l; i++) {
            long long cur_h = (h[i + l] + m - h[i]) % m;
            cur_h = (cur_h * p_pow[n-i-1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```