MU_RESPLENDENCE 2019

# GetCodes : Part 2.1

## SEU ACM ICPC Dhaka Regional 2019

MU_Resplendence
METROPOLITAN UNIVERSITY , SYLHET

## Table of Contents

# Basic Algorithms

## 1.1 : Ternary Search Algorithm

**Basic Code :**

```
int ternarySearch(int a[],int l,int r,int key)
{
    if(l <= r)
    {
        int mid1 = l + (r-l)/3;
        int mid2 = r - (r-l)/3;

        if(a[mid1] == key)
            return mid1;
        if(a[mid2] == key)
            return mid2;

        if(key > a[mid2])
            return ternarySearch(a,mid2+1,r,key);
        else if(key < a[mid1])
            return ternarySearch(a,l,mid1-1,key);
        return ternarySearch(a,mid1+1,mid2-1,key);
    }
    return -1;
}
int main()
{
    int n;

    cin >> n;

    int a[n+2];

    for(int i=1;i<=n;i++)
        cin >> a[i];

    sort(a+1,a+n+1);

    int key;

    while(cin >> key)
    {
        if(ternarySearch(a,1,n,key) == -1)
            cout << "Value Not Found!!!\n";
        else
            cout << "Found at index : " << ternarySearch(a,1,n,key) << endl;
    }
}
```

**Usage In Problem :**

```
const int MAX = 1e6;

double volume(double w,double l,double x)
{
    return (w-2*x)*(l-2*x)*x;
}
int main()
{
    FastRead

    int t,cas=1;

    cin >> t;

    while(t--)
    {
        double l,w;

        cin >> l >> w;

        double st = 0 , en = min(w/2.0,l/2.0) , mid1 , mid2 , ans = 0;

        while(en-st >= 1e-6)
        {
            mid1 = st+(en-st)/3.0;
            mid2 = en-(en-st)/3.0;

            if(volume(w,l,mid1) >= volume(w,l,mid2))
                en = mid2 , ans = volume(w,l,mid1);
            else
                st = mid1;
        }

        cout << "Case " << cas++ << ": " << fixed << setprecision(10) << ans << endl;
    }
}
```

## 1.2 : Josephus Problem

```
const int MAX = 1e6;

int josephus(int n,int k)
{
    if(n == 1)
        return 1;

    return (josephus(n-1,k)+k-1)%n + 1;
}
int main()
{
    FastRead
    int n,k;
    cin >> n >> k;
    cout <<  josephus(n,k) << endl;
}
```

# Data Structure

## 2.1 : Suffix Array

```
const int MAX = 1e5+20;
const int LOGN = 20;

struct Info
{
    int prev , now , pos;
};

int sa[MAX] , P[LOGN][MAX] , lcp[MAX] , logn , n;
Info L[MAX];

string s;

bool cmp(Info a,Info b)
{
    if(a.prev == b.prev)
        return a.now < b.now;
    return a.prev < b.prev;
}

bool cmp2(int i,int j)
{
    return P[logn][i] < P[logn][j];
}

void buildSuffixArray()
{
    for(int i=0;i<n;i++)
        P[0][i] = s[i]-'a' , sa[i] = i;

    int gap = 1 , step = 1;

    while(gap < n)
    {
        for(int i=0;i<n;i++)
        {
            L[i].prev = P[step-1][i];
            L[i].now = (i+gap < n) ? P[step-1][i+gap] : -1;
            L[i].pos = i;
        }
        sort(L,L+n,cmp);

        for(int i=0;i<n;i++)
        {
            if(i && L[i].prev == L[i-1].prev && L[i].now == L[i-1].now)
                P[step][L[i].pos] = P[step][L[i-1].pos];
            else
                P[step][L[i].pos] = i;
        }

        step++ , gap *= 2;
    }

    logn = step-1;

    sort(sa,sa+n,cmp2);
}
void buildLCP()
{
    lcp[0] = 0;

    for(int i=1;i<n;i++)
    {
        int x = sa[i] , y = sa[i-1];

        lcp[i] = 0;

        for(int j=logn;j>=0 && x<n && y<n;j--)
        {
            if(P[j][x] == P[j][y])
            {
                lcp[i] += (1<<j);
                x += (1<<j);
                y += (1<<j);
            }
        }
    }
}
int main()
{
    cin >> s;

    n = s.size();

    buildSuffixArray();
    buildLCP();

    for(int i=0;i<n;i++)
        cout << "LCP of this suffix is : " << lcp[i] << endl;
}
```

## 2.2 : Maximum Histogram

```
ll maxHistogram(vector<ll> &hist,int n)
{
    stack<int>st;
    ll mx = -1;
    int i = 0;

    while(i <= n)
    {
        ll h = (i == n) ? 0 : hist[i];
        if(st.empty() || hist[i] >= hist[st.top()])
            st.push(i++);
        else
        {
            int top = st.top();
            st.pop();
            mx = max(mx, hist[top] * (st.empty() ? i : i-1-st.top()));
        }
    }
    return mx;
}
```

## 2.3 : Next Greater Element

```
vector<int> nextGreaterElement(vector<int> &v)
{
    int n = v.size();
    vector<int> ret(n+1,n);
    stack<int>s;

    for(int i=n-1;i>=0;i--)
    {
        while(!s.empty() && v[i] >= v[s.top()])
            s.pop();

        if(!s.empty())
            ret[i] = s.top();

        s.push(i);
    }
    return ret;
}
```

## 2.4 : Sqrt Decomposition

```
const int MAX = 1e5+10;

int BLOCK_SIZE;
int BLOCK[1005];
int a[MAX];

int getID(int idx)
{
    int id = idx/BLOCK_SIZE;
    return id;
}
void init()
{
    for(int i=0;i<1005;i++)
        BLOCK[i] = 0;
}
void del(int idx)
{
    int id = getID(idx);
    BLOCK[id] -= a[idx];
    a[idx] = 0;
}
void add(int idx,int val)
{
    int id = getID(idx);
    BLOCK[id] += val;
}
int query(int l,int r)
{
    int lt = getID(l) , rt = getID(r);
    int ret = 0;

    if(lt == rt)
    {
        for(int i=l;i<=r;i++)
            ret += a[i];
        return ret;
    }
    for(int i=l;i<(lt+1)*BLOCK_SIZE;i++)
        ret += a[i];

    for(int i=lt+1;i<rt;i++)
        ret += BLOCK[i];

    for(int i=rt*BLOCK_SIZE;i<=r;i++)
        ret += a[i];

    return ret;

}
```

```
int main()
{
    // BLOCK_SIZE = sqrt(n);
    // init();
    // a[idx] += val , add(idx,val);
    // del(idx);
    // query(l,r);
}
```

## 2.5 : Kth Element in Segment

```
build(){ // sum in range }

update() { // add value to index idx }

int findKth(int pos,int l,int r,int k)
{
    if(l == r)
        return l;

    int mid = (l+r) >> 1;
    if(tree[pos*2] >= k)
        return findKth(pos*2,l,mid,k);
    else
        return findKth(pos*2+1,mid+1,r,k-tree[pos*2]);
}

MO(){ // for kTh element in l to r range }

main()
{
      // build(1,1,maxElementInArray);

      // update(1,1,n,a[i],1)

      // update(1,1,n,a[i],-1)

      // findKth(1,1,n,k)

      // store frequency of each element in segment array
}
```

## 2.6 : Persistent Segment Tree

### 2.6.1 : Versions in Segment Tree

```
const int MAX = 1e5+10;

struct Node
{
    int left, right, val;

} tree[MAX*20];

int a[MAX], root[MAX], id;

void build(int pos,int l,int r)
{
    if(l == r)
    {
        tree[pos].val = a[l];
        return;
    }

    int mid = (l+r)>>1;
    tree[pos].left = ++id, tree[pos].right = ++id;

    build(tree[pos].left,l,mid);
    build(tree[pos].right,mid+1,r);

    tree[pos].val = tree[tree[pos].left].val + tree[tree[pos].right].val;
}
int update(int pos,int l,int r,int idx,int v)
{
    if(idx > r || idx < l)
        return pos;
    else if(l == r)
    {
        tree[++id] = tree[pos];
        tree[id].val += v;
        return id;
    }

    int mid = (l+r)>>1;
    tree[++id] = tree[pos], pos = id;

    tree[pos].left = update(tree[pos].left,l,mid,idx,v);
    tree[pos].right = update(tree[pos].right,mid+1,r,idx,v);

    tree[pos].val = tree[tree[pos].left].val + tree[tree[pos].right].val;

    return pos;
}
```

```
int query(int pos,int l,int r,int L,int R)
{
    if(l > R || r < L)
        return 0;
    else if(l >= L && r <= R)
        return tree[pos].val;

    int mid = (l+r)/2;

    int x = query(tree[pos].left,l,mid,L,R);
    int y = query(tree[pos].right,mid+1,r,L,R);

    return x+y;
}
void init(int n)
{
    root[0] = id = 1;
    build(root[0],1,n);
}
int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007

    FastRead

    int n,q,type,idx,l,r,pos,v,cc=0;

    cin >> n;

    for(int i=1; i<=n; i++)
        cin >> a[i];

    init(n);

    cin >> q;

    while(q--)
    {
        cin >> type;

        if(type == 1)
        {
            cin >> idx >> pos >> v;
            root[++cc] = update(root[idx],1,n,pos,v);
        }
        else
        {
            cin >> idx >> l >> r;
            cout << query(root[idx],1,n,l,r) << endl;
        }
    }
}
```

## 2.6.2 : Kth element in sorted range (l,r)

```
const int MAX = 1e5+10;
struct Node
{
    int left, right, val;

} tree[MAX*20];

int a[MAX], root[MAX], id;
map<int,int>mp,origin;

int update(int pos,int l,int r,int idx)
{
    if(idx > r || idx < l)
        return pos;
    else if(l == r)
    {
        tree[++id] = tree[pos];
        tree[id].val++;
        return id;
    }

    int mid = (l+r)>>1;
    tree[++id] = tree[pos] , pos = id;

    tree[pos].left = update(tree[pos].left,l,mid,idx);
    tree[pos].right = update(tree[pos].right,mid+1,r,idx);

    tree[pos].val = tree[tree[pos].left].val + tree[tree[pos].right].val;

    return pos;
}
int kthElement(int a,int b,int l,int r,int k)
{
    if(l == r)
        return l;

    int mid = (l+r)>>1;
    int cnt = tree[tree[a].left].val - tree[tree[b].left].val;

    if(cnt >= k)
        return kthElement(tree[a].left,tree[b].left,l,mid,k);
    else
        return kthElement(tree[a].right,tree[b].right,mid+1,r,k-cnt);
}
void init(int n,int m)
{
    root[0] = tree[0].left = tree[0].right = tree[0].val = 0;
    for(int i=1;i<=n;i++)
        root[i] = update(root[i-1] , 1 , m , mp[a[i]]);
}
```

```
int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007

    FastRead

    int n,q,l,r,k,m=0;

    cin >> n >> q;

    set<int>ss;
    for(int i=1; i<=n; i++)
        cin >> a[i] , ss.insert(a[i]);

    for(auto i : ss)
        mp[i] = ++m , origin[m] = i;

    init(n,m);

    while(q--)
    {
        cin >> l >> r >> k;

        cout << origin[kthElement(root[r],root[l-1],1,m,k)] << endl;
    }
}
```

## 2.6.2 : Number of element less than or equal to K

```
const int MAX = 1e5+10;

struct Node
{
    int left, right, val;

} tree[MAX*20];

int a[MAX], root[MAX], id;
map<int,int>mp,origin;

int update(int pos,int l,int r,int idx)
{
    if(idx > r || idx < l)
        return pos;
    else if(l == r)
    {
        tree[++id] = tree[pos];
        tree[id].val++;
        return id;
    }
    int mid = (l+r)>>1;
    tree[++id] = tree[pos] , pos = id;

    tree[pos].left = update(tree[pos].left,l,mid,idx);
    tree[pos].right = update(tree[pos].right,mid+1,r,idx);

    tree[pos].val = tree[tree[pos].left].val + tree[tree[pos].right].val;

    return pos;
}
int lessCnt(int a,int b,int l,int r,int idx)
{
    if(r <= idx)
        return tree[a].val - tree[b].val;

    int mid = (l+r)>>1;

    if(idx <= mid)
        return lessCnt(tree[a].left,tree[b].left,l,mid,idx);
    else
        return lessCnt(tree[a].left,tree[b].left,l,mid,idx) +
lessCnt(tree[a].right,tree[b].right,mid+1,r,idx);
}
void init(int n,int m)
{
    root[0] = tree[0].left = tree[0].right = tree[0].val = 0;
    for(int i=1;i<=n;i++)
        root[i] = update(root[i-1] , 1 , m , mp[a[i]]);
}
int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007

    int n,q,l,r,k,m=0;

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

    set<int>ss;
    for(int i=1; i<=n; i++)
        scanf("%d",&a[i]) , ss.insert(a[i]);

    for(auto i : ss)
        mp[i] = ++m , origin[m] = i;

    init(n,m);

    while(q--)
    {
        scanf("%d%d%d",&l,&r,&k);

        int st = 1 , en = m , mid , best;
        while(st <= en)
        {
            mid = (st+en)/2;

            int cnt = 0;
            if(mid > 1)
                cnt = lessCnt(root[r],root[l-1],1,m,mid-1);

            if(cnt <= k-1)
                st = mid+1 , best = mid;
            else
                en = mid-1;
        }

        printf("%d\n",origin[best]);
    }
}
```

# Graph Algorithm

## 3.1 : Minimum Vertext Cover

```
const int MAX = 1e5+10;

int dp[MAX][2];
vector<int>adj[MAX];

int DP(int src,int isGuard,int par)
{
    if(~dp[src][isGuard])
        return dp[src][isGuard];

    int sum = 0;
    for(auto i : adj[src])
    {
        if(i == par)
            continue;

        if(isGuard)
            sum += min(DP(i,1,src),DP(i,0,src));
        else
            sum += DP(i,1,src);
    }

    return dp[src][isGuard] = sum + isGuard;
}
int main()
{
    FastRead
    int n,u,v;

    cin >> n;

    for(int i=1;i<n;i++)
    {
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    memset(dp,-1,sizeof dp);

    if(n == 1)
        cout << 0 << endl;
    else
        cout << min(DP(1,1,-1),DP(1,0,-1)) << endl;
}
```

## 3.2 : Articular Bridge

```
const int MAX = 1e5+10;

vector<int>adj[MAX];
set<pii>bridges;
bool vis[MAX];
int dist[MAX] , lowTime[MAX] , timo;

void init()
{
    timo = 0;
    memset(vis,0,sizeof vis);
    memset(dist,0,sizeof dist);
    memset(lowTime,0,sizeof lowTime);
    for(int i=0;i<MAX;i++)
        adj[i].clear();
}
void addBridge(int u,int v)
{
    if(u > v)
        swap(u,v);
    bridges.insert({u,v});
}
void DFS(int src,int par)
{
    vis[src] = 1;
    dist[src] = lowTime[src] = timo++;

    for(auto i : adj[src])
    {
        if(i == par)
            continue;
        if(!vis[i])
        {
            DFS(i,src);
            lowTime[src] = min(lowTime[src],lowTime[i]);

            if(dist[src] < lowTime[i])
                addBridge(i,src);
        }
        else
            lowTime[src] = min(lowTime[src],dist[i]);
    }
}
void findBridges()
{
    DFS(1,-1);

    cout << "Articular Bridges are :\n";
    for(auto i : bridges)
        cout << i.first << "->" << i.second << endl;
}

int main()
{
    FastRead

    int n,m,u,v;

    init();

    cin >> n >> m;

    for(int i=0;i<m;i++)
    {
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    findBridges();
}
```

## 3.3 : Articulation Point

```
const int MAX = 1e4+10;

vector<int>adj[MAX];
bool vis[MAX], isArt[MAX];
int dist[MAX], lowTime[MAX], timo;

void init()
{
    timo = 1;
    memset(dist,0,sizeof dist);
    memset(lowTime,0,sizeof lowTime);
    memset(vis,0,sizeof vis);
    memset(isArt,0,sizeof isArt);
    for(int i=0; i<MAX; i++)
        adj[i].clear();
}
void DFS(int src,int par)
{
    vis[src] = 1;
    dist[src] = lowTime[src] = timo++;

    int child = 0;
    for(auto i : adj[src])
    {
        if(i == par)
            continue;
        if(!vis[i])
        {
            DFS(i,src);
            lowTime[src] = min(lowTime[src],lowTime[i]);

            if(dist[src] <= lowTime[i] && par != -1)
                isArt[src] = 1;
            child++;
        }
        else
            lowTime[src] = min(lowTime[src],dist[i]);
    }
    if(child > 1 && par == -1)
        isArt[src] = 1;
}
void findArticulationPoints(int n)
{
    DFS(1,-1);

    cout << "Articulation Points : ";
    for(int i=1; i<=n; i++)
    {
        if(isArt[i])
            cout << i << " -> ";
    }
    cout << endl;
}


int main()
{
    FastRead

    int n,m,u,v;

    init();

    cin >> n >> m;
    for(int i=0; i<m; i++)
    {
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    findArticulationPoints(n);
}
```

## 3.4 : Diameter of a Tree

```
const int MAX = 1e5+10;

vector<int>adj[MAX];
int root , mx;

void DFS(int src,int par,int lev)
{
    if(lev > mx)
    {
        mx = lev;
        root = src;
    }
    for(auto i : adj[src])
    {
        if(i != par)
            DFS(i,src,lev+1);
    }
}

int main()
{
    int n,u,v;

    cin >> n;

    for(int i=1;i<n;i++)
    {
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    root = 1;
    DFS(root,-1,0);

    int root1 = root;
    DFS(root,-1,0);

    cout << root1 << " -> " << root << " : " << mx << endl;
}
```

## 3.4 : Heavy-Light Decomposition

```
const int MAX = 1e4+10;

vector<pii>adj[MAX] , idx;

///  For HLD
int chainNo , ptr , chainHead[MAX] , chainPos[MAX] , chainIdx[MAX] , sub[MAX];

/// For Segment Tree
int arr[MAX] , tree[4*MAX];

/// For LCA
int dep[MAX] , T[MAX] , P[MAX][20];

void init()
{
    idx.clear();
    for(int i=0;i<MAX;i++)
        adj[i].clear();
    chainNo = 0; ptr = 0;
    memset(chainHead,-1,sizeof chainHead);
    memset(P,-1,sizeof P);
}

/// Segment Tree Part
void segBuild(int pos,int l,int r)
{
    if(l == r)
    {
        tree[pos] = arr[l];
        return;
    }

    int mid = (l+r)/2 , lt = pos*2, rt = pos*2+1;

    segBuild(lt,l,mid);
    segBuild(rt,mid+1,r);

    tree[pos] = max(tree[lt],tree[rt]);
}
void segUpdate(int pos,int l,int r,int idx,int val)
{
    if(l > idx || r < idx)
        return;
    else if(l == r && l == idx)
    {
        tree[pos] = val;
        return;
    }

    int mid = (l+r)/2 , lt = pos*2, rt = pos*2+1;
    segUpdate(lt,l,mid,idx,val);
    segUpdate(rt,mid+1,r,idx,val);

    tree[pos] = max(tree[lt],tree[rt]);
}
int segQuery(int pos,int l,int r,int L,int R)
{
    if(l > R || r < L)
        return 0;
    else if(l >= L && r <= R)
        return tree[pos];

    int mid = (l+r)/2 , lt = pos*2, rt = pos*2+1;

    int x = segQuery(lt,l,mid,L,R);
    int y = segQuery(rt,mid+1,r,L,R);

    return max(x,y);
}

/// LCA Part
void DFS(int src,int par,int lev)
{
    dep[src] = lev;
    T[src] = par;
    sub[src] = 0;

    for(int i=0;i<adj[src].size();i++)
    {
        int x = adj[src][i].first;

        if(x == par)
            continue;

        DFS(x,src,lev+1);
        sub[src] += sub[x];
    }
}
void initLCA(int n)
{
    memset(P,-1,sizeof P);

    for(int i=1;i<=n;i++)
        P[i][0] = T[i];

    for(int j=1; 1<<j <n;j++)
    {
        for(int i=1;i<=n;i++)
        {
            if(P[i][j-1] != -1)
                P[i][j] = P[P[i][j-1]][j-1];
        }
    }
}
```

```
int query(int n,int u,int v)
{
    if(dep[u] < dep[v])
        swap(u,v);

    int log = log2(n);

    for(int i=log;i>=0;i--)
    {
        if(dep[u]-(1<<i) >= dep[v])
            u = P[u][i];
    }

    if(u == v)
        return u;

    for(int i=log;i>=0;i--)
    {
        if(P[u][i] != -1 && P[u][i] != P[v][i])
        {

            u = P[u][i];
            v = P[v][i];
        }
    }
    return T[u];
}

/// HLD Part
void HLD(int cur,int cost,int p)
{
    if(chainHead[chainNo] == -1)
        chainHead[chainNo] = cur;

    chainIdx[cur] = chainNo;
    chainPos[cur] = ptr+1;
    arr[++ptr] = cost;

    int id = -1 , mx = -1 , newCost = -1;

    for(int j=0;j<adj[cur].size();j++)
    {
        pii i = adj[cur][j];

        if(i.first != p && sub[i.first] > mx)
        {
            mx = sub[i.first];
            id = i.first;
            newCost = i.second;
        }
    }

    if(id != -1)
        HLD(id,newCost,cur);
    for(int j=0;j<adj[cur].size();j++)
    {
        pii i = adj[cur][j];

        if(i.first != p && i.first != id)
            chainNo++ , HLD(i.first,i.second,cur);
    }
}
int query_up(int u,int v)
{
    if(u == v)
        return 0;

    int uchain , vchain = chainIdx[v] , ans = -1;

    while(1)
    {
        uchain = chainIdx[u];

        if(uchain == vchain)
        {
            if(u == v)
                break;

            int val = segQuery(1,1,ptr,chainPos[v]+1,chainPos[u]);
            ans = max(ans,val);
            break;
        }

        int val = segQuery(1,1,ptr,chainPos[chainHead[uchain]],chainPos[u]);
        ans = max(ans,val);

        u = chainHead[uchain];
        u = P[u][0];
    }
    return ans;
}
int ansQuery(int n,int u,int v)
{
    int lca = query(n,u,v);

    int q1 = query_up(u,lca);
    int q2 = query_up(v,lca);

    return max(q1,q2);
}
/// Change Query and Main Function part
void change(int u,int v)
{
    if(chainPos[idx[u].first] > chainPos[idx[u].second])
        u = idx[u].first;
    else
        u = idx[u].second;
    segUpdate(1,1,ptr,chainPos[u],v);
}
```

```
int main()
{
    int t;

    scanf("%d",&t);

    while(t--)
    {
        init();

        int n,u,v,w;

        scanf("%d",&n);

        for(int i=1;i<n;i++)
        {
            scanf("%d%d%d",&u,&v,&w);

            adj[u].push_back({v,w});
            adj[v].push_back({u,w});
            idx.emplace_back(u,v);
        }

        DFS(1,-1,0);
        initLCA(n);
        HLD(1,0,-1);

        segBuild(1,1,ptr);

        char type[105];

        while(scanf("%s",type))
        {
            if(type[0] == 'D')
                break;

            scanf("%d%d",&u,&v);

            if(type[0] == 'C')
                change(u-1,v);
            else
                printf("%d\n",ansQuery(n,u,v));
        }
    }
}
```

## 3.5 : Stable Marriage Problem

```
const int MAX = 105;

int men[MAX][MAX], women[MAX][MAX], choices;
int freeMen[MAX], freeWomen[MAX] , pos[MAX][MAX];
int n;

int isFree()
{
    for(int i=1; i<=n; i++)
    {
        if(freeMen[i])
            continue;
        for(int j=0; j<choices; j++)
        {
            if(!freeWomen[men[i][j]])
                return i;
        }
    }
    return -1;
}



void stableMarriageProblem()
{
    memset(freeMen,0,sizeof freeMen);
    memset(freeWomen,0,sizeof freeWomen);

    while(1)
    {
        int cur = isFree();
        if(cur == -1)
            break;

        for(int j=0; j<choices; j++)
        {
            if(!freeWomen[men[cur][j]])
            {
                freeMen[cur] = men[cur][j];
                freeWomen[men[cur][j]] = cur;
                break;
            }
            else
            {
                int curW = men[cur][j];

                if(pos[curW][freeWomen[curW]] > pos[curW][cur])
                {
                    freeMen[freeWomen[curW]] = 0;
                    freeMen[cur] = curW;
                    freeWomen[curW] = cur;
                    break;
                }
            }
        }
    }
}
int main()
{
    int t,cas=1;

    scanf("%d",&t);

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

        choices = n;

        for(int i=1; i<=n; i++)
        {
            for(int j=0; j<choices; j++)
            {
                scanf("%d",&men[i][j]);
                men[i][j] -= n;
            }
        }

        for(int i=1; i<=n; i++)
        {
            for(int j=0; j<choices; j++)
            {
                scanf("%d",&women[i][j]);
                pos[i][women[i][j]] = j;
            }
        }

        stableMarriageProblem();

        printf("Case %d:",cas++);
        for(int i=1;i<=n;i++)
        {
            if(freeMen[i])
                printf(" (%d %d)",i,freeMen[i]+n);
        }
        printf("\n");
    }
}
```

## 3.6 : Ford Fulkerson Method Edmonds-Karp MaxFlow Algorithm

```
const int MAX = 105;

int residualCapacity[MAX][MAX] , capacity[MAX][MAX] , par[MAX];
bool vis[MAX];

bool BFS(int s,int t)
{
    for(int i=1;i<MAX;i++)
        par[i] = i;
    queue<int>q;
    memset(vis,0,sizeof vis);
    q.push(s);
    vis[s] = 1;

    while(q.size())
    {
        int u = q.front();
        q.pop();
        for(int v=1;v<MAX;v++)
        {
            if(!vis[v] && residualCapacity[u][v])
            {
                par[v] = u;
                q.push(v);
                vis[v] = 1;
            }
        }
    }
    return vis[t];
}
int fordFulkerson(int s,int e)
{
    for(int i=1;i<MAX;i++)
        for(int j=1;j<MAX;j++)
            residualCapacity[i][j] = capacity[i][j];

    int maxFlow = 0;
    while(BFS(s,e))
    {
        int flow = INT_MAX , cur = e;
        while(cur != s)
        {
            flow = min(flow,residualCapacity[par[cur]][cur]);
            cur = par[cur];
        }

        maxFlow += flow;

        cur = e;
        while(cur != s)
        {
            residualCapacity[par[cur]][cur] -= flow;
            residualCapacity[cur][par[cur]] += flow;
            cur = par[cur];
        }
    }

    return maxFlow;
}
int main()
{
    FastRead

    int t,cas=1;

    cin >> t;
    while(t--)
    {
        int n,m,u,v,c,s,e;

        cin >> n >> s >> e >> m;

        memset(capacity,0,sizeof capacity);

        while(m--)
        {
            cin >> u >> v >> c;
            capacity[u][v] += c;
            capacity[v][u] += c;
        }

        cout << "Case " << cas++ << ": " << fordFulkerson(s,e) << endl;
    }
}
```

## 3.7 : Johnson’s Algorithm

```
const int MAX = 1e3+10;

struct Info
{
    int v,w;
    Info(){}
    Info(int _v,int _w) { v = _v ; w = _w; }
};

struct Edge
{
    int u,v,w;
    Edge(){}
    Edge(int _u,int _v,int _w){ u = _u; v = _v; w = _w; }
};

vector<Info>adj[MAX];
vector<Edge>edges;

int n,m,h[MAX] , dist[MAX][MAX];

void init()
{
    for(int i=0;i<MAX;i++)
        adj[i].clear();
}

bool bellman_ford(int src)
{
    fill(h,h+MAX,1e8);
    h[src] = 0;
    for(int i=1;i<=n;i++)
    {
        for(auto j : edges)
        {
            if(h[j.v] > h[j.u]+j.w)
                h[j.v] = h[j.u] + j.w;
        }
    }
    for(auto j : edges)
        if(h[j.v] > h[j.u]+j.w)
            return 1;
    return 0;
}
void dijkstra(int src)
{
    fill(dist[src],dist[src]+MAX,1e8);
    priority_queue< pii,vector<pii>,greater<pii> > pq;

    pq.push(pii(0,src));
    dist[src][src] = 0;

    while(pq.size())
    {
        int u = pq.top().second;
        pq.pop();

        for(auto j : adj[u])
        {
            int v = j.v , w = j.w;

            if(dist[src][v] > dist[src][u]+w)
            {
                dist[src][v] = dist[src][u]+w;
                pq.push(pii(dist[src][v],v));
            }
        }
    }
}
void modifyGraph(int sign)
{
    for(int i=1;i<=n;i++)
    {
        for(auto &j : adj[i])
            j.w += sign*(h[i]-h[j.v]);
    }
}
bool johnson_algorithm()
{
    /// Adding source node to calculate h[]
    int src = 0;
    edges.clear();
    for(int i=1;i<=n;i++)
    {
        for(auto j : adj[i])
            edges.push_back(Edge(i,j.v,j.w));
        adj[src].push_back(Info(i,0));
        edges.push_back(Edge(src,i,0));
    }

    /// Modifying edges to avoid negative weight edges
    if(bellman_ford(src))
        return 0;
    modifyGraph(1);

    /// Running Dijkstra for each node
    for(int i=1;i<=n;i++)
        dijkstra(i);
    modifyGraph(-1);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(dist[i][j] < 1e8)
                dist[i][j] -= (h[i]-h[j]);
        }
    }
    adj[src].clear();
    return 1;
}
```

## 3.8 : Ford Fulkerson MaxFlow Algorithm

```
const int MAX = 5e4+10;

struct Edge
{
    int u , v , cap;

    Edge(){}
    Edge(int _u,int _v,int _cap) { u = _u , v = _v , cap = _cap; }
};

vector<Edge>edges;
vector<int>adj[MAX];
bool vis[MAX];
int s,t;

void init(int _s,int _t)
{
    s = _s , t = _t;
    for(int i=0;i<MAX;i++)
        adj[i].clear();
    edges.clear();
}
```

```
void addEdge(int u,int v,int w)
{
    edges.push_back(Edge(u,v,w));
    edges.push_back(Edge(v,u,0));

    adj[u].push_back(edges.size()-2);
    adj[v].push_back(edges.size()-1);
}

int pushFlow(int u,int flow = 1e9)
{
    vis[u] = 1;
    if(u == t)
        return flow;

    int ret = 0;

    for(int i=0;i<adj[u].size();i++)
    {
        int idx = adj[u][i];
        Edge &e = edges[idx];

        if(!e.cap || vis[e.v])
            continue;

        ret = pushFlow(e.v,min(flow,e.cap));


        if(ret)
        {
            Edge &rev = edges[idx^1];

            e.cap -= ret;
            rev.cap += ret;

            return ret;
        }
    }
    return ret;
}
int maxFlow()
{
    int ans = 0;
    while(1)
    {
        memset(vis,0,sizeof vis);

        int flow = pushFlow(s);
        if(flow == 0)
            break;
        ans += flow;
    }
    return ans;
}
```

```
int main()
{
    /// int source , sink;

    /// init(source,sink);

    /// addEdge(u,v,w)

    /// int maximumFlow = maxFlow();

}
```

## 3.9 : Centroid Decomposition

```
const int MAX = 1e5+10;

vector<int>adj[MAX];
int n , dep[MAX] , T[MAX] , P[MAX][30];

void DFS(int src,int par,int lev)
{
    dep[src] = lev;
    T[src] = par;

    for(int i=0;i<adj[src].size();i++)
    {
        int x = adj[src][i];

        if(x == par)
            continue;
        DFS(x,src,lev+1);
    }
}
void initLCA()
{
    memset(P,-1,sizeof P);

    for(int i=1;i<=n;i++)
        P[i][0] = T[i];
    for(int j=1; 1<<j <n;j++)
    {
        for(int i=1;i<=n;i++)
        {
            if(P[i][j-1] != -1)
                P[i][j] = P[P[i][j-1]][j-1];
        }
    }
}
int lca_query(int u,int v)
{
    if(dep[u] < dep[v])
        swap(u,v);
    int log = log2(n);
    for(int i=log;i>=0;i--)
    {
        if(dep[u]-(1<<i) >= dep[v])
            u = P[u][i];
    }
    if(u == v)
        return u;
    for(int i=log;i>=0;i--)
    {
        if(P[u][i] != -1 && P[u][i] != P[v][i])
        {
            u = P[u][i];
            v = P[v][i];
        }
    }
    return T[u];
}
int dist(int u,int v)
{
    int lca = lca_query(u,v);
    return dep[u] + dep[v] - 2*dep[lca];
}

struct CentroidDecomposition
{
    int path[MAX] , sub[MAX];
    bool vis[MAX];

    CentroidDecomposition()
    {
        memset(vis,0,sizeof vis);
        memset(path,0,sizeof path);
    }

    void subDFS(int src,int par)
    {
        sub[src] = 1;

        for(auto i : adj[src])
        {
            if(i == par || vis[i])
                continue;

            subDFS(i,src);
            sub[src] += sub[i];
        }
    }
    int centroid(int src,int par,int sz)
    {
        for(auto i : adj[src])
        {
            if(i == par || vis[i])
                continue;
            else if(sub[i] > sz)
                return centroid(i,src,sz);
        }
        return src;
    }
    void decompose(int src,int par)
    {
        subDFS(src,-1);
        int c = centroid(src,-1,sub[src]/2);
        vis[c] = 1;
        path[c] = par;

        for(auto i : adj[c])
        {
            if(!vis[i])
                decompose(i,c);
        }
    }
} tree;

bool color[MAX];
multiset<int>storage[MAX];

struct QueryHandler
{
    void update(int u)
    {
        if(color[u])
            return;

        color[u] = 1;
        int cur = u;

        while(cur != -1)
        {
            if(color[u])
                storage[cur].insert(dist(u,cur));

            cur = tree.path[cur];
        }
    }
    int query(int u)
    {
        int cur = u , ret = 1e9;

        while(cur != -1)
        {
            if(storage[cur].size())
                ret = min(ret , *storage[cur].begin() + dist(u,cur) );

            cur = tree.path[cur];
        }
        return ret;
    }
} ds;
```

```
int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007

    FastRead

    int u,v,q,type;

    cin >> n >> q;

    for(int i=0;i<n-1;i++)
    {
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    DFS(1,1,0);
    initLCA();

    tree.decompose(1,-1);
    ds.update(1);

    while(q--)
    {
        cin >> type >> u;

        if(type == 1)
            ds.update(u);
        else
            cout << ds.query(u) << endl;
    }
}
```

# Matrix

## 4.1 : Matrix Exponentiation

```
const int MAX = 1e6;

struct Matrix
{
    int n,m;
    vector< vector<int> > mat;

    Matrix() {}
    Matrix(int _n,int _m)
    {
        n = _n, m = _m;
        mat = vector< vector<int> > (n,vector<int>(m));
    }
};

Matrix multiply(Matrix a,Matrix b,int MOD)
{
    Matrix c = Matrix(a.n,b.m);

    for(int i=0; i<a.n; i++)
    {
        for(int j=0; j<b.m; j++)
        {
            c.mat[i][j] = 0;
            for(int k=0; k<a.m; k++)
            {
                c.mat[i][j] += (1LL * a.mat[i][k] * b.mat[k][j])%MOD;
                if(c.mat[i][j] >= MOD)
                    c.mat[i][j] -= MOD;
            }
        }
    }
    return c;
}
Matrix pow(Matrix a,ll p,int MOD)
{
    if(p == 1)
        return a;

    Matrix x = pow(a,p/2,MOD);
    x = multiply(x,x,MOD);

    if(p&1)
        x = multiply(x,a,MOD);
    return x;
}
```

```
Matrix createA(int a1,int a2,int b1,int b2,int c1,int c2)
{
    Matrix a(6,6);

    a.mat[0][0] = a1, a.mat[0][1] = b1, a.mat[0][5] = c1;
    a.mat[3][3] = a2, a.mat[3][4] = b2, a.mat[3][2] = c2;
    a.mat[1][0] = a.mat[2][1] = a.mat[4][3] = a.mat[5][4] = 1;

    return a;
}
Matrix createB(int f[],int g[])
{
    Matrix a(6,1);

    a.mat[0][0] = f[2];
    a.mat[1][0] = f[1];
    a.mat[2][0] = f[0];
    a.mat[3][0] = g[2];
    a.mat[4][0] = g[1];
    a.mat[5][0] = g[0];

    return a;
}
void print(Matrix A)
{
    for(int i=0; i<A.n; i++)
        for(int j=0; j<A.m; j++)
            cout << A.mat[i][j] << " \n"[j ==A.m-1];
}
int main()
{
    FastRead

    int t,cas=1;

    cin >> t;

    while(t--)
    {
        int a1,b1,c1,a2,b2,c2,n,q;

        cin >> a1 >> b1 >> c1 >> a2 >> b2 >> c2;

        int f[3], g[3];

        for(int i=0; i<3; i++)
            cin >> f[i];
        for(int i=0; i<3; i++)
            cin >> g[i];

        int MOD,ans1,ans2;

        cin >> MOD;

        Matrix A = createB(f,g);
        cin >> q;

        cout << "Case " << cas++ << ":\n";
        while(q--)
        {
            cin >> n;

            if(n <= 2)
            {
                ans1 = f[n]%MOD;
                ans2 = g[n]%MOD;
            }
            else
            {
                Matrix M = pow(createA(a1,a2,b1,b2,c1,c2),n-2,MOD);

                ans1 = multiply(M,createB(f,g),MOD).mat[0][0];
                ans2 = multiply(M,createB(f,g),MOD).mat[3][0];
            }

            cout << ans1 << " " << ans2 << endl;
        }
    }
}
```

# Number Theory

## 5.1 : Sum of divisors of n^m

```
const int MAX = 1e5+10;
const int MOD = 1e9+7;

bool prime[MAX];
vector<int>primes;

ll bigMod(ll a,ll b,ll M)
{
    if(b==0) return 1;
    ll x = bigMod(a,b/2,M);
    x = (x*x)%M;
    if(b&1)
        x = (x*a)%M;
    return x;
}
void sieve()
{
    fill(prime,prime+MAX,1);

    prime[2] = 1;
    primes.push_back(2);
    for(int i=4; i<MAX; i+=2)
        prime[i] = 0;
    for(int i=3; i<MAX; i+=2)
    {
        if(prime[i])
        {
            primes.push_back(i);
            for(int j=i+i; j<MAX; j+=i)
                prime[j] = 0;
        }
    }
}
ll primeFact(ll n,int m)
{
    ll sum = 1;
    for(int i=0; i<primes.size() && primes[i]<=n; i++)
    {
        ll cnt = 0, p = primes[i];

        if(n%p == 0)
        {
            while(n%p == 0)
                cnt++ , n /= p;
            cnt = cnt*m+1;

            ll calc = (bigMod(p,cnt,MOD)+MOD-1)%MOD;
            calc *= bigMod(p-1,MOD-2,MOD);
            calc %= MOD;
            sum = (sum*calc)%MOD;

        }
    }
    if(n > 1)
    {
        ll calc = (bigMod(n,1+m,MOD)+MOD-1)%MOD;
        calc *= bigMod(n-1,MOD-2,MOD);
        calc %= MOD;

        sum = (sum*calc)%MOD;
    }
    return sum;
}

int main()
{
    FastRead

    sieve();

    int t,cas=1;

    cin >> t;

    while(t--)
    {
        int n,m;

        cin >> n >> m;

        cout << "Case " << cas++ << ": " << primeFact(n,m) << endl;
    }
}
```

## 5.2 : Trailing Zeroes of nCr * p^q

```
const int MAX = 1e5+10;

ll calcX(int n,int x,ll p)
{
    ll cnt = 0;

    while(n%x == 0)
    {
        cnt++;
        n /= x;
    }
    return cnt*p;
}
ll calcfactX(int n,int p)
{
    ll cnt = 0 , x = n;
    while(x/p)
    {
        cnt += x/p;
        x /= p;
    }
    return cnt;
}
int main()
{
    FastRead

    int t,n,r,p,q,cas=1;

    cin >> t;

    while(t--)
    {
        cin >> n >> r >> p >> q;

        ll c1 = calcX(p,2,q) , c2 = calcX(p,5,q);

        ll c3 = calcfactX(n,2)-calcfactX(r,2)-calcfactX(n-r,2) , c4 = calcfactX(n,5)-
calcfactX(r,5)-calcfactX(n-r,5) ;

        cout << "Case " << cas++ << ": " << min(c1+c3,c2+c4) << endl;
    }
}
```

## 5.3 : Prime Factorization of N!

```
const int MAX = 1e5+10;

bool prime[MAX];
vector<int>primes;
void sieve()
{
    fill(prime,prime+MAX,1);

    prime[2] = 1;
    primes.push_back(2);
    for(int i=4; i<MAX; i+=2)
        prime[i] = 0;
    for(int i=3; i<MAX; i+=2)
    {
        if(prime[i])
        {
            primes.push_back(i);
            for(int j=i+i; j<MAX; j+=i)
                prime[j] = 0;
        }
    }
}
vector<pii> factorialPrimeFact(int n)
{
    vector<pii> ans;
    for(int i=0; i<primes.size() && primes[i]<=n; i++)
    {
        int cnt = 0, p = primes[i] , x = n;
        while(x/p)
        {
            cnt += x/p;
            x /= p;
        }
        if(cnt)
            ans.push_back({primes[i],cnt});
    }
    return ans;
}
void solve(int n)
{
    vector<pii> ans = factorialPrimeFact(n);
    cout << n << " = ";
    for(int i=0; i<ans.size(); i++)
    {
        if(i)
            cout << " * ";
        cout << ans[i].first << " (" << ans[i].second << ")";
    }
    cout << endl;
}
```

```
int main()
{
    FastRead

    sieve();

    int t,n,cas=1;

    cin >> t;

    while(t--)
    {
        cin >> n;

        cout << "Case " << cas++ << ": ";

        solve(n);
    }
}
```

## 5.4 : Sum of SODs of all number in range 1 to N

```
const int MAX = 1e6;

ll solve(int n)
{
    int sq = sqrt(n);
    ll sum = 0;

    for(int i=2;i<=sq;i++)
    {
        int j = n/i;
        sum += (j+i)*1LL*(j-i+1)/2;
        sum += (j-i)*1LL*i;
    }
    return sum;
}
int main()
{
    FastRead

    int t,n,cas=1;

    cin >> t;

    while(t--)
    {
        cin >> n;

        cout << "Case " << cas++ << ": " << solve(n) << endl;
    }
}
```

## 5.5 : Modular Multiplicative Inverse

```
const int MAX = 1e6;

ll extendedGCD(ll a,ll b,ll *x,ll *y)
{
    if(a == 0)
    {
        *x = 0 , *y = 1;
        return b;
    }

    ll x1 , y1;
    ll gcd = extendedGCD(b%a,a,&x1,&y1);

    *x = y1 - (b/a)*x1;
    *y = x1;

    return gcd;
}
ll modInverse(ll a,ll M)
{
    if(__gcd(a,M) > 1)
        return -1;

    ll x , y;
    ll gcd = extendedGCD(a,M,&x,&y);

    return (x+M)%M;
}
int main()
{
    cout << modInverse(3,11) << endl;

    /// ans = 4 , because (4*3)%11 = 12%11 = 1
}
```

## 5.6 : Chinese Remainder Theorem

```
int n;
ll num[MAX], rem[MAX];

ll chineseRemainderTheorem()
{
    ll res = 0, prod = 1;

    for(int i=0; i<n; i++)
        prod *= num[i];

    for(int i=0; i<n; i++)
    {
        ll pp = prod/num[i];

        res += rem[i] * pp * modInverse(pp,num[i]);
        res %= prod;
    }

    return res;
}
int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007


    cin >> n;

    for(int i=0; i<n; i++)
        cin >> num[i] >> rem[i];

    cout << chineseRemainderTheorem() << endl;
}
```

## 5.7 : Segmented Sieve

```
const int MAX = 1e5+10;

bool status[MAX] , mark[MAX];
vector<ll>primes;

void sieve()
{
    status[1] = false;
    status[2] = true;
    for(int i=3;i<MAX;i++)
    {
        if(i&1)
            status[i] = true;
        else
            status[i] = false;
    }

    for(int i=3;i*i<MAX;i+=2)
    {
        if(status[i])
        {
            for(int j=i*i;j<MAX;j+=i)
                status[j] = false;
        }
    }
    for(int i=2;i<MAX;i++)
    {
        if(status[i])
            primes.push_back(i);
    }
}
void segmented_sieve(ll l,ll r)
{
    memset(mark,true,sizeof mark);
    if(l == 1)
        mark[0] = false;

    for(int i=0;primes[i]*primes[i]<=r;i++)
    {
        ll base = primes[i]*primes[i];

        if(base < l)
            base = (l+primes[i]-1)/primes[i] * primes[i];

        for(int j=base;j<=r;j+=primes[i])
            mark[j-l] = false;

    }
}

int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007

    FastRead

    sieve();

    int t;
    ll l,r;

    cin >> t;
    while(t--)
    {
        cin >> l >> r;

        segmented_sieve(l,r);

        for(ll i=0;i<=r-l;i++)
        {
            if(mark[i])
                cout << l+i << endl;
        }
        cout << endl;
    }
}
```

# String Matching

## 6.1 : Z Algorithm

```
const int MAX = 2e6+10;

string S;
int z[MAX];
void zFunction()
{
    int left, right;
    left = right = z[0] = 0;
    for(int i=1; i<S.size(); i++)
    {
        if(i <= right)
            z[i] = min(z[i-left],right-i+1);

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

        if(i+z[i]-1 > right)
            left = i, right = i+z[i]-1;
    }
}
bool isSubstr(string t,string p)
{
    S = p + "#" + t;
    zFunction();

    for(int i=p.size()+1; i<S.size(); i++)
    {
        if(z[i] == p.size())
            return true;
    }
    return false;
}
```

```
int countSubstr(string t,string p)
{
    S = p + "#" + t;
    memset(z,0,sizeof z);
    zFunction();
    int cnt = 0;
    for(int i=p.size()+1; i<S.size(); i++)
    {
        if(z[i] == p.size())
            cnt++;
    }
    return cnt;
}

int countNonOverlappingSubstr(string t,string p)
{
    S = p + "#" + t;
    memset(z,0,sizeof z);
    zFunction();

    int cnt = 0;
    for(int i=p.size()+1; i<S.size(); i++)
    {
        if(z[i] == p.size())
            cnt++ , i = i+z[i]-1;
    }
    return cnt;
}
int main()
{
    string txt,p;

    cin >> txt >> p;

    cout << countNonOverlappingSubstr(txt,p) << endl;
}
```

## 6.2 : Aho Corasick Algorithm

```
#define ll          long long
#define pii         pair<ll,ll>
#define bug(a)      cerr << #a << " : " << a << endl;
#define FastRead    ios_base::sync_with_stdio(false);cin.tie(NULL);

const int MAX = 3e5;

int cnt[505];

struct AhoCorasick
{
    int node;
    int trie[MAX][26] , failure[MAX];
```

```
vector<int>mark[MAX];

void init()
{
    fill(trie[0],trie[0]+26,-1);
    mark[0].clear();
    node = 0;
}
int getID(char ch)
{
    int id = ch-'a';
    return id;
}

void Insert(string s,int idx)
{
    int cur = 0 , n = s.size();

    for(int i=0;i<n;i++)
    {
        int id = getID(s[i]);

        if(trie[cur][id] == -1)
        {
            trie[cur][id] = ++node;
            mark[node].clear();
            fill(trie[node],trie[node]+26,-1);
        }
        cur = trie[cur][id];
    }
    mark[cur].push_back(idx);
}

void computeFailure()
{
    queue<int>q;
    failure[0] = 0;
    for(int i=0;i<26;i++)
    {
        if(trie[0][i] != -1)
        {
            failure[trie[0][i]] = 0;
            q.push(trie[0][i]);
        }
        else
            trie[0][i] = 0;
    }
    while(!q.empty())
    {
        int u = q.front();
        q.pop();

        for(int id : mark[failure[u]])
            mark[u].push_back(id);
            for(int i=0;i<26;i++)
            {
                if(trie[u][i] != -1)
                {
                    failure[trie[u][i]] = trie[failure[u]][i];
                    q.push(trie[u][i]);
                }
                else
                    trie[u][i] = trie[failure[u]][i];
            }
        }
    }
} automata;

void countFreq(string s)
{
    int cur = 0 , n = s.size();

    for(int i=0;i<n;i++)
    {
        int id = automata.getID(s[i]);
        cur = automata.trie[cur][id];

        for(int idx : automata.mark[cur])
            cnt[idx]++;
    }
}
int main()
{
#ifdef Aaman007
    freopen("input.txt","r",stdin);
    // freopen("output.txt","w",stdout);
#endif // Aaman007

    FastRead

    int t,cas=1;

    cin >> t;

    while(t--)
    {
        int n;
        string s,p;

        cin >> n >> s;

        automata.init();
        for(int i=0;i<n;i++)
        {
            cin >> p;
            automata.Insert(p,i);
        }
        automata.computeFailure();
        memset(cnt,0,sizeof cnt);
        countFreq(s);

        cout << "Case " << cas++ << ":\n";
        for(int i=0;i<n;i++)
            cout << cnt[i] << endl;
    }
}
```

# Others

## 7.1 : Fast I/O

### Fast Input :

```
#define gc getchar
inline void Cin(int &num)
{
    num = 0;
    char c = gc();
    int flag = 0;
    while(!((c >= '0' & c <= '9') ||
            c == '-'))
    {
        c = gc();
    }
    if(c == '-')
    {
        flag = 1;
        c = gc();
    }
    while(c >= '0' && c <= '9')
    {
        num = (num << 1) + (num << 3)
              + c - '0';
        c = gc();
    }
    if(flag == 1)
    {
        num = 0 - num;
    }
```

### Fast Output :

```
#define pc putchar
inline void Cout(ll n)
{
    ll NN = n, rev;
    int count = 0;
    rev = NN;
    if(NN == 0)
    {
        pc('0');
        pc('\n');
        return ;
    }
    while((rev % 10) == 0)
    {
        count++;
        rev /= 10;
    }
    rev = 0;
    while(NN != 0)
    {
        rev = (rev << 3) + (rev << 1)
              + NN % 10;
        NN /= 10;
    }
    while(rev != 0)
    {
        pc(rev % 10 + '0');
        rev /= 10;
    }
    while(count--)
    {
        pc('0');
    }
    pc('\n');
}
```