# Team hhAhaHAHHAHahHAhaaHaHaHAHaHah

## Nanyang Technological university

# Table of Contents



# String Algorithms

## KMP

```
vll kmp(string s) {
    int n = s.size();
    vll kmp(n);
    for (int i=1;i<n;i++) {
        int j = kmp[i-1];
        while (j&&s[i]!=s[j]) j = kmp[j-1];
        kmp[i] = j+(s[i] == s[j]);
    }
    return kmp;
}
```

## Z Function

> Longest string that is a prefix of S and a prefix of the suffix of S in $O(n)$

```
vll z_function(string s) {
    int n = s.size();
    vll z(n);
    z[0] = n;
    for (int i=1,l=0,r=0;i<n;i++) {
        if (i<=r && z[i-l] < r-i+1)
            z[i] = z[i-l];
```

```
        else {
            z[i] = max(0, r-i+1);
            while (i+z[i] < n && s[z[i]] == s[i + z[i]])
                z[i]++;
        }
        if (i+z[i]-1 > r)
            l=i, r=i + z[i]-1;
    }
    return z;
}
```

## Aho Corasick Algorithm

> For multiple string KMP in $O(n+m+z)$

```
int ans[N]

struct Aho_Corasick {
    vll id[N];
    int son[N][26];
    int fail[N];
    int val[N];
    int cnt;

    Aho_Corasick() {
        cnt = 0;
        memset(son, 0, sizeof(son));
        memset(fail, 0, sizeof(fail));
        memset(val, 0, sizeof(val));
    }

    void insert(string s, int _id) {
        int now = 0;
        for (auto c: s) {
            const int x = c - 'a';
            if (!son[now][x]) {
                son[now][x] = ++cnt;
            }
            now = son[now][x];
        }
        id[now].push_back(_id);
    }

    vll fas[N];

    void build() {
        queue<ll> q;
        for (int i=0;i<26;i++) if (son[0][i])
            q.push(son[0][i]);
        while (!q.empty()) {
            int now = q.front(); q.pop();
            for (int i=0;i<26;i++) {
                if (son[now][i]) {
                    fail[son[now][i]] = son[fail[now]][i];
                    q.push(son[now][i]);
                } else {
                    son[now][i] = son[fail[now]][i];
                }
            }
        }
    }

    void getval(string s) {
        int now = 0;
        for (auto c: s) {
            now = son[now][c-'a'];
            val[now]++;
        }
    }

    void build_fail_tree() {
        for (int i=1;i<=cnt;i++) {
            fas[fail[i]].push_back(i);
        }
    }

    void dfs(int now=0) {
        for (auto x: fas[now]) {
            dfs(x);
            val[now] += val[x];
        }
        if (!id[now].empty()) {
            for (auto x: id[now])
                ans[x] = val[now];
        }
    }
};

Aho_Corasick ac;

int n;

int main() {
    cin>>n;
    for (int i=1;i<=n;i++) {
        string s;
        cin>>s;
        ac.insert(s);
    }
    ac.build();
    string s;
    cin>>s;
    ac.getval(s);
    ac.build_fail_tree();
    ac.dfs();
    for (int i=1;i<=n;i++)
        cout<<ans[i]<<"\n";
}
```

## Manacher's Algorithm

```
int manacher() {
    int i, p, ans = 0;
    r[1] = 0, p=1;
    for (i=2;i<=n;i++) {
```

```
        if (i<=p+r[p])
            r[i] = min(r[2*p-i], p + r[p]-i);
        else r[i] = 1;
        while (st[i-r[i]] == st[i+r[i]]) r[i]++;
        --r[i];
        if (i+r[i]>p + r[p]) p=i;
        ans = max(ans, r[i]);
    }
    return ans;
}
```

# Number Theory

## GCD - Extended Euclidean Algorithm

```
void ex_gcd(ll a, ll b, ll &d, ll &x, ll &y){
    if(b == 0) y = 0, x = 1, d = a;
    else ex_gcd(b, a % b, d, y, x), y -= a / b * x;
}
```

## Miller-Rabin Primality Test

```
bool miller_rabin(n) {
    if (n<3 || !(n&1)) return n==2;
    ll a = n-1, b=0;
    while (!(a&1)) a>>=1, b++;

    // Increase iterations for more accuracy
    for (int i=0;i<8;i++) {
        int x = rand() % (n-2) + 2;
        int v = modPow(x, a, n);
        if (v==1) continue;
        ll j = 0;
        while (j < b) {
            if (v==n-1) break;
            v = (v*v) %n;
            j++;
        }
        if (j>=b) return false;
    }
    return true;
}
```

## Sieve of Euler

```
void Euler(const int n = 100000) {
    np[1] = true;
    int cnt = 0;
    for (int i = 2; i <= n; ++i) {
        if (!np[i]) {
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt && (LL) i * prime[j] <= n; ++j) {
            np[i * prime[j]] = true;
            if (!(i % prime[j])) {
                break;
            }
        }
    }
}
```

## Totient Function

```
void pre() { 263
    for (int i = 1; i <= 5000000; ++i) {
        is_prime[i] = 1;
    }
    int cnt = 0;
    is_prime[1] = 0;
    phi[1] = 1;
    for (int i = 2; i <= 5000000; ++i) {
        if (is_prime[i]) {
            prime[++cnt] = i;
            phi[i] = i - 1;
        }
        for (int j = 1; j <= cnt && i * prime[j] <= 5000000; j++) {
            is_prime[i * prime[j]] = 0;
            if (i % prime[j])
                phi[i * prime[j]] = phi[i] * phi[prime[j]];
            else {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
        }
    }
}
```

## Matrix Modular Power

```
struct Matrix {
    ll mat[N][N]{};

    friend Matrix operator * (Matrix a, Matrix b) {
        Matrix ans;
        for (int i=0;i<N;i++)
            for (int k=0;k<N;k++)
                if (a.mat[i][k]) for (int j=0;j<N;j++)
                    ans.mat[i][j] = (((a.mat[i][k] * b.mat[k][j]) %M + M)%M
                                     + ans.mat[i][j]) %M;
        return ans;
    }
};

Matrix matPow(Matrix v, ll p) {
    if (!p) {
        Matrix ans;
        for(int i=0;i<N;i++) ans.mat[i][i]=1;
```

```
        return ans;
    }
    Matrix ans = matPow(v, p>>1);
    ans = ans*ans;
    if (p&1) ans = ans*v;
    return ans;
}
```

## CRT - Chinese Remainder Theorem

```
ll crt(int k, vll &a, vll &r) {
    ll n = 1, ans = 0;
    for (int i=1;i<=k;i++) n *= r[i];
    for (int i=1;i<=k;i++) {
        ll m = n/r[i], b, y;
        extended_gcd(m, r[i], b, y);
        ans = (ans + a[i] * m * b % n) %n;
    }
    return (ans%n + n) %n;
}
```

## Matrix Exponential

```
int n,q;
int mask[30005] = {0};
int arr[8][8] = {0};
int MOD = 1e9+7;

int mul(int a, int b)
{
        ll multi = (ll) a * (ll) b % (ll) MOD;
        return (int) multi;
}

int add(int a,int b) { return (a+b)%MOD;}

struct matrix
{
        int e[8][8];
        inline void init(int i)
        {
                for(int curr=0;curr<8;curr++)
                {
                        for(int prv=0;prv<8;prv++)
                        {
                                if(curr>mask[i]) e[prv][curr] = 0;
                                if((mask[i]-curr)&(7-mask[i])) e[prv][curr] = 0;
                else e[prv][curr] = arr[prv][mask[i]-curr];
                        }
                }
                return;
        }

        matrix operator * (matrix& other) const
        {
                matrix ret;

                for(int i = 0; i < 8; i++){
                        for(int j = 0; j < 8; j++){
                                ret.e[i][j] = 0;
                                for(int k = 0; k < 8; k++){
                                        ret.e[i][j] = add(ret.e[i][j], mul(e[i][k], other.e[k][j]));
                                }
                        }
                }

        return ret;
        }
};
matrix iden;

vector<ll> ans;
vector<ll> vtmp;
inline void apply(matrix &a)
{
        vtmp.clear();
        vtmp.assign(8,0);
        for(int i=0;i<8;i++)
        {
                for(int j=0;j<8;j++)
                {
                        vtmp[i] = (vtmp[i]+ans[j]*a.e[j][i])%MOD;
                }
        }
        ans.clear();
        ans = vtmp;
        return;
}

int N = 1<<15;
struct segtree
{
        vector<matrix> v;

        void init()
        {
                v.assign(2*N,iden);
                return;
        }

        void build()
        {
                for(int i=N-1;i>0;--i) v[i] = v[i<<1]*v[i<<1|1];
        }

        inline void upd(int pos,int node,int lx,int rx)
        {
                if(rx-lx==1)
                {
                        v[node].init(pos);
                        return;
                }
                int mid = (lx+rx)>>1;
                if(pos<mid) upd(pos,node<<1,lx,mid);
                else upd(pos,(node<<1)|1,mid,rx);
```

```
                v[node] = v[node<<1]*v[(node<<1)|1];
                return;
        }

        inline void query(int l,int r,int node,int lx,int rx)
        {
                if(l>=rx || lx>=r) return;
                if(l<=lx && rx<=r)
                {
                        apply(v[node]);
                        return;
                }
                int mid = (lx+rx)/2;
                if(r<=mid)
                {
                        query(l,r,2*node,lx,mid);
                        return;
                }
                else if(l>=mid)
                {
                        query(l,r,2*node+1,mid,rx);
                        return;
                }
                else
                {
                        query(l,r,2*node,lx,mid);
                        query(l,r,2*node+1,mid,rx);
                }
        }
};

void solve()
{
        segtree st;
        st.init();
        for(int i=1;i<=n;i++)
        {
                matrix tmp;
                tmp.init(i);
                st.v[N+i] = tmp;
        }
        st.build();

        int type,l,r;
        for(int i=1;i<=q;i++)
        {
                scanf("%d %d %d",&type,&l,&r);
                if(type==2)
                {
                        ans.clear();
                        ans.assign(8,0); ans[0] = 1;
                        st.query(l,r+1,1,0,N);

                        ll ret = 0;
                        for(int i=0;i<8;i++)
                        {
                                ret += ans[i];
                                ret %= MOD;
                        }
                        printf("%lld\n",ret);
                }
                else
                {
                        mask[r]^=(1<<(3-l));
                        st.upd(r,1,0,N);
                }
        }
        return;
}

void brute()
{
        for(int prv=0;prv<8;prv++)
    {
        for(int curr=0;curr<8;curr++)
        {
            int mask = curr;
            int cnt = 0;
            if((prv&mask)==mask) cnt = 1;
            else cnt = 0;
            if(mask==7)
            {
                                cnt = 0;
                if(prv&1) cnt++;
                                if(prv&4) cnt++;
                if(prv==7) cnt++;
            }
            else if(mask==3 || mask==6) cnt++;

            arr[prv][curr] = cnt;
        }
    }

        for(int i=0;i<8;i++)
        {
                iden.e[i][i] = 1;
        }
        return;
}

void input()
{
        scanf("%d %d",&n,&q);
        char a[3][30005];
        for(int i=0;i<3;i++) {scanf("%s",&a[i]);}
        for(int i=1;i<=n;i++)
        {
                int t[3];
                for(int j=0;j<3;j++)
                {
                        if(a[j][i-1]=='.') t[j] = 1;
                        else t[j] = 0;
                }
                mask[i] = t[0]*4+t[1]*2+t[2];
                //cout << mask[i] << endl;
        }
        return;
}

int main()
```

```
{
        brute();
        input();
        solve();
        return 0;
}
```

# Integer Sequences

## Fibonnacci Numbers

$f_n = f_{n-1} + f_{n-2}$

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418
```

## Catalan Numbers

$C_n = \sum_{i=1}^{n} H_{i-1}H_{n-i} = \frac{\binom{2n}{n}}{n+1} = \binom{2n}{n} - \binom{2n}{n-1}$

```
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440
```

## Bell or Exponential Numbers

$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$

Number of ways to partition a set of n labeled elements

```
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597
```

## Lucas Numbers

$L_n = L_{n-1} + L_{n-2}, L_0 = 2, L_1 = 1$

```
2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571,
5778, 9349, 15127, 24476, 39603, 64079, 103682, 167761, 271443, 439204
```

## Derangement

$D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$

Number of permutations of n elements with no fixed points

```
1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961
```

## Prufer

$n^{n-1}$

Number of labelled rooted trees with n nodes

```
1, 2, 9, 64, 625, 7776, 117649, 2097152, 43046721
```

# Data Structures

## CDQ - DP DNC

```
int n,k;
ll arr[4005][4005] = {0};
ll cost[4005][4005] = {0};
ll dp[805][4005] = {0};
ll INF = 1e9;

ll calc(int prv,int ind)
{
    return cost[prv][ind];
}

void dpdnc(int i,int l,int r,int optl,int optr)
{
    if(l>r || optl>optr) return;
    int mid = (l+r)/2;

    int opt = optl;
    dp[i][mid] = INF;
    for(int k=optl;k<=min(mid-1,optr);k++)
    {
        ll val = dp[i-1][k]+calc(k+1,mid);
        if(dp[i][mid]>val)
        {
            dp[i][mid] = val;
            opt = k;
        }
    }
    dpdnc(i,l,mid-1,optl,opt);
    dpdnc(i,mid+1,r,opt,optr);
}

const int bufsz = 40101010;
struct fastio{
        char buf[bufsz];
        int cur;
        fastio(){
                cur = bufsz;
        }
        inline char nextchar(){
                if(cur==bufsz){
                        fread(buf, bufsz, 1, stdin);
                        cur=0;
                }
```

```
                return buf[cur++];
        }
        inline int nextint(){
                int x = 0;
                char c = nextchar();
                while(!('0' <= c && c <= '9')){
                        c = nextchar();
                }
                while('0' <= c && c <= '9'){
                        x = x*10+c-'0';
                        c = nextchar();
                }
                return x;
        }
}io;

void solve()
{
    n = io.nextint(); k = io.nextint();
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            arr[i][j] = io.nextint();
            arr[i][j] += arr[i][j-1];
        }
    }

    for(int i=1;i<=n;i++)
    {
        cost[i][i] = 0;
        for(int j=i+1;j<=n;j++)
        {
            cost[i][j] = cost[i][j-1]+arr[j][j]-arr[j][i-1];
        }
    }

    for(int i=0;i<=k;i++)
    {
        for(int j=0;j<=n;j++)
        {
            dp[i][j] = INF;
        }
    }
    dp[0][0] = 0;

    for(int i=1;i<=k;i++)
    {
        dpdnc(i,0,n,0,n);
    }
    cout << dp[k][n] << endl;
        return;
}
```

## Persistent Segment Tree

```
struct Node
{
    int val;
    Node *left,*right;

    Node(Node* l, Node* r, int v)
    {
        left = l;
        right = r;
        val = v;
    }
};

int n,q;
int N = 1<<17;

pair<int,int> arr[100005]; //val,index
int v[100005] = {0};
int pos[100005] = {0};
Node* version[100005];

struct segtree
{
    void build(Node* node,int lx,int rx) //node start with root
    {
        if (rx-lx==1) return;
        int mid = (lx+rx)/2;
        node->left = new Node(NULL, NULL, 0);
        node->right = new Node(NULL, NULL, 0);
        build(node->left,lx,mid);
        build(node->right,mid,rx);
        node->val = node->left->val + node->right->val;
        return;
    }

    void upd(int pos,Node* prev,Node* curr,int lx,int rx) //prev start with root
    {
        if(rx-lx==1)
        {
            curr->val++;
            return;
        }
        int mid = (lx+rx)/2;
        if(pos<mid)
        {
            curr->right = prev->right;
            curr->left = new Node(NULL, NULL, 0);
            upd(pos,prev->left,curr->left,lx,mid);
        }
        else
        {
            curr->left = prev->left;
            curr->right = new Node(NULL, NULL, 0);
            upd(pos,prev->right,curr->right,mid,rx);
        }
        curr->val = curr->left->val + curr->right->val;
        return;
    }

    ll query(int k,Node* node1,Node* node2,int lx,int rx)
    {
        if(rx-lx==1) return lx;
```

```
        int mid = (lx+rx)/2;
        int leftval = node2->left->val - node1->left->val;
        if(leftval>=k) return query(k,node1->left,node2->left,lx,mid);
        else query(k-leftval,node1->right,node2->right,mid,rx);
    }
};

void solve()
{
    cin >> n >> q;
    for(int i=1;i<=n;i++)
    {
        cin >> arr[i].fir;
        arr[i].sec = i;
    }
    sort(arr+1,arr+n+1);
    for(int i=1;i<=n;i++)
    {
        pos[arr[i].sec]=i;
    }

    Node *root = new Node(NULL, NULL, 0);
    segtree st;
    st.build(root,0,N);
    version[0] = root;

    for(int i=1;i<=n;i++)
    {
        version[i] = new Node(NULL, NULL, 0);
        st.upd(pos[i],version[i-1],version[i],0,N);
    }

    for(int i=0;i<q;i++)
    {
        int l,r,k;
        cin >> l >> r >> k;
        ll ind = st.query(k,version[l-1],version[r],0,N);
        cout << arr[ind].first << endl;
    }
    return;
}
```

## Mo's Algorithm

```
ll n,sqrtn,m,k;
ll val[200005] = {0};
ll cnt[1800005] = {0};
ll ans[200005] = {0};
struct query
{
    int l,r,idx;
};
bool cmpMo(query lhs,query rhs)
{
    int l1 = lhs.l/sqrtn; int l2 = rhs.l/sqrtn;
    if(l1!=l2) return l1<l2;
    else return lhs.r<rhs.r;
    //return l1 != l2 ? l1 < l2 : lhs.r < rhs.r;
}

vector<query> queries;

void solve()
{
    cin >> n >> m >> k;
    sqrtn = sqrt(2*n);
    for(int i=1;i<=n;i++)
    {
        cin >> val[i];
        val[i] ^= val[i-1];
    }
    for(int i=1;i<=m;i++)
    {
        query tmp;
        cin >> tmp.l >> tmp.r;
        tmp.idx = i;
        queries.pb(tmp);
    }
    sort(queries.begin(),queries.end(),cmpMo);

    int l = 0, r = -1;
    ll currcnt = 0;
    for(auto q:queries)
    {
        q.l--;
        while(r<q.r)
        {
            ++r;
            currcnt += cnt[val[r]^k];
            cnt[val[r]]++;
        }
        while(r>q.r)
        {
            cnt[val[r]]--;
            currcnt -= cnt[val[r]^k];
            r--;
        }
        while (l < q.l)
        {
            cnt[val[l]]--;
            currcnt -= cnt[val[l]^k];
            l++;
        }
        while (l > q.l)
        {
            --l;
            currcnt += cnt[val[l]^k];
            cnt[val[l]]++;
        }
        ans[q.idx] = currcnt;
    }

    for(int i=1;i<=m;i++)
    {
        cout << ans[i] << endl;
    }
    return;
}
```

# Graph Theory

## SCC - Tarjan's Algorithm

```
int cnt, scc;
vll dfs_num, dfs_low, visited;
stack<ll> St;

void tarjanSCC(int u) {
    dfs_low[u] = dfs_num[u] = cnt++;
    St.push(u);
    visited[u] = 1;
    for (auto v: adj[u]) {
        if (dfs_num[v]==UNVISITED) tarjanSCC(v);
        if (visited[v]) dfs_low[u] = min(dfs_low[u], dfs_low[v]);
    }
}
```

## Max Flow - Dinic's Algorithm

```
int gn,gm;
int grid[305][305] = {0};
int curr = 1;
int cnt = 0;
int noden[305][305][4] = {0};
set<int> hori,verti;
vector<int> conn[200005];
int src,sink;

bool black(int i,int j)
{
    return grid[i][j];
}

struct Flow_Edge
{
    int v,u;
    ll cap,flow = 0;
    Flow_Edge(int v, int u, ll cap) : v(v), u(u), cap(cap) {}
};

struct Dinic
{
    const ll flow_inf = 1e18;
    int n, s, t;
    int m = 0;
    vector<Flow_Edge> edges;
    vector<vector<int>> conn;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t)
    {
        conn.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, ll cap)
    {
        edges.push_back({v,u,cap});
        edges.push_back({u,v,0});
        conn[v].push_back(m);
        conn[u].push_back(m+1);
        m += 2;
    }

    bool bfs()
    {
        while (!q.empty())
        {
            int v = q.front();
            q.pop();
            for (int id:conn[v])
            {
                if (edges[id].cap-edges[id].flow<1 || level[edges[id].u]!=-1) continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    ll dfs(int v, ll pushed)
    {
        if (pushed == 0) return 0;
        if (v == t) return pushed;
        for (int& cid = ptr[v]; cid < conn[v].size(); cid++)
        {
            int id = conn[v][cid];
            int u = edges[id].u;
            if (level[v]+1!=level[u] || edges[id].cap-edges[id].flow<1) continue;
            ll tr = dfs(u, min(pushed,edges[id].cap-edges[id].flow));
            if (tr == 0) continue;
            edges[id].flow += tr;
            edges[id^1].flow -= tr;
            return tr;
        }
        return 0;
    }

    ll flow()
    {
        ll f = 0;
        while (true)
        {
            fill(level.begin(),level.end(),-1);
            q.push(s);
            level[s] = 0;
            if (!bfs()) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (ll pushed = dfs(s, flow_inf)) f += pushed;
        }
        return f;
    }
```

```
};

void solve()
{
    cin >> gn >> gm;
    for(int i=1;i<=gn;i++)
    {
        string s;
        cin >> s;
        for(int j=1;j<=gm;j++)
        {
            if(s[j-1]=='#')
            {
                grid[i][j] = 1;
                cnt++;
            }
        }
    }

    for(int i=1;i<=gn;i++)
    {
        for(int j=1;j<=gm;j++)
        {
            int up = -1;
            int down = -1;
            int left = -1;
            int right = -1;

            if(black(i,j) && black(i-1,j))
            {
                int tmp = curr;
                if(noden[i][j][0]!=0) tmp = noden[i][j][0];
                else if(noden[i-1][j][2]!=0) tmp = noden[i-1][j][2];
                noden[i][j][0] = tmp;
                noden[i-1][j][2] = tmp;
                up = tmp;
                verti.insert(tmp);
                curr++;
            }

            if(black(i,j) && black(i+1,j))
            {
                int tmp = curr;
                if(noden[i][j][2]!=0) tmp = noden[i][j][2];
                else if(noden[i+1][j][0]!=0) tmp = noden[i+1][j][0];
                noden[i][j][2] = curr;
                noden[i+1][j][0] = curr;
                down = tmp;
                verti.insert(tmp);
                curr++;
            }

            if(black(i,j) && black(i,j-1))
            {
                int tmp = curr;
                if(noden[i][j][3]!=0) tmp = noden[i][j][3];
                else if(noden[i][j-1][1]!=0) tmp = noden[i][j-1][1];
                noden[i][j][3] = curr;
                noden[i][j-1][1] = curr;
                left = tmp;
                hori.insert(tmp);
                curr++;
            }

            if(black(i,j) && black(i,j+1))
            {
                int tmp = curr;
                if(noden[i][j][1]!=0) tmp = noden[i][j][3];
                else if(noden[i][j+1][3]!=0) tmp = noden[i][j+1][3];
                noden[i][j][1] = curr;
                noden[i][j+1][3] = curr;
                right = tmp;
                hori.insert(tmp);
                curr++;
            }

            //cout << up << " " << down << " " << left << " " << right << endl;
            if(up!=-1 && right!=-1) conn[up].pb(right);
            if(up!=-1 && left!=-1) conn[up].pb(left);
            if(down!=-1 && right!=-1) conn[down].pb(right);
            if(down!=-1 && left!=-1) conn[down].pb(left);
        }
    }

    src = curr++;
    int cnt2 = 0;
    for(int x:verti)
    {
        conn[src].pb(x);
        cnt2++;
    }

    sink = curr++;
    for(int x:hori)
    {
        conn[x].pb(sink);
        cnt2++;
    }

    //cout << src << endl;
    //cout << sink << endl;

    Dinic D(200000,src,sink);
    for(int i=1;i<=200000;i++)
    {
        for(auto x:conn[i])
        {
            //cout << i << " " << x << endl;
            D.add_edge(i,x,1);
        }
    }
    cout << cnt-(cnt2-D.flow()) << endl;
        return;
}
```

## 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;
}
```

## LCA - Lowest Common Ancestor

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

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
```

```
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

## HLD - Heavy Light Decomposition

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;

int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;
        }
    }
    return size;
}

void decompose(int v, int h, vector<vector<int>> const& adj) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);
    }
}

void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;

    dfs(0, adj);
    decompose(0, 0, adj);
}

int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = segment_tree_query(pos[head[b]], pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = segment_tree_query(pos[a], pos[b]);
    res = max(res, last_heavy_path_max);
    return res;
}
```

## Topological Sorting

```
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());
}
```

## Centroid Decomposition

```
ll n,MOD;
ll sz[200005] = {0};
ll vis[200005] = {0};
ll ans = 0;
vector<pair<ll,ll>> conn[200005];
vector<pair<pair<ll,ll>,ll>> up_down;
ll invpow[200005] = {0};
ll pow10[200005] = {0};

inline ll totient(ll x)
{
        ll res = x;
        for (ll i = 2; i*i <= x; ++i)
        {
                if (x % i == 0)
                {
```

```
                        res = res * (i-1) / i;
                        while (x % i == 0) x /= i;
                }
        }
        if (x > 1) res = res * (x-1) / x;

        return res;
}

ll exp(ll a,ll pow)
{
        if(pow==0) return 1;
        if(pow==1) return a%MOD;
        ll tmp = exp(a,pow/2);
        tmp = (tmp*tmp)%MOD;
        if(pow%2==1) tmp = (tmp*a)%MOD;
        return tmp%MOD;
}

ll inv(ll x)
{
        return exp(x,totient(MOD)-1);
}

void precexp()
{
    ll tmp = inv(10);
    invpow[0] = 1;
    pow10[0] = 1;
    for(ll i=1;i<=200000;i++)
    {
        invpow[i] = (invpow[i-1]*tmp)%MOD;
        pow10[i] = (pow10[i-1]*10)%MOD;
    }
    return;
}

ll find_size(ll node,ll prv)
{
    ll size = 1;
    for(auto x:conn[node])
    {
        if(x.fir==prv) continue;
        size += find_size(x.fir,node);
    }
    return sz[node] = size;
}

ll find_centroid(ll node,ll prv,ll subsize)
{
    ll centroid = -1;
    for(auto x:conn[node])
    {
        if(x.fir==prv || vis[x.fir]==1) continue;
        if(sz[x.fir]>subsize/2)
        {
            sz[node] -= sz[x.fir];
            sz[x.fir] += sz[node];
            centroid = find_centroid(x.fir,node,subsize);
        }
    }
    if(centroid==-1)
    {
        centroid = node;
        vis[node] = 1;
    }
    return centroid;
}

inline void dfs(ll node,ll prv,ll up,ll down,ll dep)
{
    up_down.pb(mp(mp(up,down),dep));
    for(auto x:conn[node])
    {
        if(x.fir==prv || vis[x.fir]) continue;
        dfs(x.fir,node,(up+pow10[dep]*x.sec)%MOD,(10*down+x.sec)%MOD,dep+1);
    }
    return;
}

vector<ll> appear;
inline ll pivot(ll root)
{
    up_down.clear();
    dfs(root,-1,0,0,0);

        appear.clear();
        for(auto x:up_down)
        {
        appear.pb(x.fir.fir);
        }
    sort(appear.begin(),appear.end());

        ll tmp = 0;
        for(auto x:up_down)
        {
                ll val = invpow[x.sec]%MOD;
        val = (val*(MOD-(x.fir.sec)%MOD))%MOD;
                tmp += upper_bound(appear.begin(),appear.end(),val)-upper_bound(appear.begin(),appear.
                if(val==x.fir.fir) tmp--;
        }
    return tmp;
}

vector<pair<pair<ll,ll>,ll>> up_down2[100005];
ll fcnt;
inline void dfs2(ll node,ll prv,ll up,ll down,ll dep,bool isroot,ll cnt)
{
    up_down2[cnt].pb(mp(mp(up,down),dep));
    for(auto x:conn[node])
    {
        if(x.fir==prv || vis[x.fir]) continue;
        if(isroot) dfs2(x.fir,node,(up+pow10[dep]*x.sec)%MOD,(10*down+x.sec)%MOD,dep+1,0,++cnt);
        else dfs2(x.fir,node,(up+pow10[dep]*x.sec)%MOD,(10*down+x.sec)%MOD,dep+1,0,cnt);
    }
    if(isroot) fcnt = cnt;
    return;
}

inline ll dc(ll root)
```

```
{
    dfs2(root,-1,0,0,0,1,0);
    ll tmp = 0;
    for(ll i=1;i<=fcnt;i++)
    {
        appear.clear();
        for(auto x:up_down2[i])
        {
            appear.pb(x.fir.fir%MOD);
        }
        sort(appear.begin(),appear.end());

        for(auto x:up_down2[i])
        {
            ll val = invpow[x.sec]%MOD;
            val = (val*(MOD-(x.fir.sec)%MOD))%MOD;
            tmp += upper_bound(appear.begin(),appear.end(),val)-upper_bound(appear.begin(),appear.end(
            if(val==x.fir.fir) tmp--;
        }

        up_down2[i].clear();
    }
    return tmp;
}

void rec_centroid(ll curr)
{
    ll centroid = find_centroid(curr,-1,sz[curr]);

    ans += pivot(centroid);
    ans -= dc(centroid);

    for(auto x:conn[centroid])
    {
        if(vis[x.fir]) continue;
        rec_centroid(x.fir);
    }
    return;
}

void solve()
{
    cin >> n >> MOD;
    precexp();

    for(ll i=1;i<n;i++)
    {
        ll a,b,w;
        cin >> a >> b >> w;
                a++; b++;
        conn[a].push_back(mp(b,w));
        conn[b].push_back(mp(a,w));
    }

    find_size(1,-1);
    rec_centroid(1);
    cout << ans << endl;
}
```

# Geometry

## DP CHT

```
int n;
ll dp[1000005] = {0};
pair<ll, pair<ll, ll>> arr[1000005];

struct Line
{
        mutable ll m, c, p;
        bool operator<(const Line &o) const { return m < o.m; }
        bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>>
{
        static 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())
                        return x->p = inf, 0;
                if (x->m == y->m)
                        x->p = x->c > y->c ? inf : -inf;
                else
                        x->p = div(y->c - x->c, x->m - y->m);
                return x->p >= y->p;
        }

        void add(ll m, ll c)
        {
                auto z = insert({m, c, 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.m * x + l.c;
        }
}
```

## CHT - Convex Hull

```
struct point
{
    int x,y;
```

```
};

bool sortpoints(const point &lhs, const point &rhs)
{
    return (lhs.x < rhs.x) || (lhs.x==rhs.x && lhs.y < rhs.y);
}

vector<point> v;
vector<point> ans;

int cross(point a,point b){return a.x*b.y-a.y*b.x;}

int ccw(point p,point q,point r){
    point p1; p1.x = q.x-p.x; p1.y = q.y-p.y;
    point p2; p2.x = p.x-r.x; p2.y = p.y-r.y;
    int tmp = cross(p1,p2);
    if (tmp>0) return 1;
    else if(tmp==0) return 0;
    else return -1;
}

vector<point> find_CH(vector<point>v)
{
    int n = v.size();
    vector<point> lh;
    vector<point> uh;
    sort(v.begin(),v.end(),sortpoints);
    lh.push_back(v[0]);
    lh.push_back(v[1]);
    for(int i=2;i<n;i++)
    {
        while(lh.size()>1)
        {
            if(ccw(lh[lh.size()-2],lh[lh.size()-1],v[i])!=-1) lh.pop_back();
            else break;
        }
        lh.push_back(v[i]);
    }

    uh.push_back(v[n-1]);
    uh.push_back(v[n-2]);
    for(int i=2;i<n;i++)
    {
        while(uh.size()>1)
        {
            if(ccw(uh[uh.size()-2],uh[uh.size()-1],v[n-i-1])!=-1) uh.pop_back();
            else break;
        }
        uh.push_back(v[n-i-1]);
    }

    for(int i=1;i<uh.size()-1;i++)
    {
        lh.push_back(uh[i]);
    }
    return lh;
}

void solve()
{
    while(true)
    {
        v.clear();
        int n;
        cin >> n;
        if(n==0) break;
        set<pair<int,int>> s;
        for (int i=0;i<n;i++)
        {
            point tmp;
            cin >> tmp.x >> tmp.y;
            if(s.find(make_pair(tmp.x,tmp.y))!=s.end()) continue;
            s.insert(make_pair(tmp.x,tmp.y));
            v.push_back(tmp);
        }
        if(v.size()==1)
        {
            cout << 1 << endl;
            cout << v[0].x << " " << v[0].y << endl;
            continue;
        }
        ans.clear();
        ans = find_CH(v);
        cout << ans.size() << endl;
        for(int i=0;i<ans.size();i++)
        {
            cout << ans[i].x << " " << ans[i].y << endl;
        }
    }
    return;
}
```

## Nearest Point Pair on a Plane

```
struct pt {
    int x, y, id;
};

struct cmp_x {
    bool operator()(const pt& a, const pt& b) const {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
};

struct cmp_y {
    bool operator()(const pt& a, const pt& b) const { return a.y < b.y; }
};

int n;
vector<pt> a;

double mindist;
int ansa, ansb;

inline void upd_ans(const pt& a, const pt& b) {
    double dist =
        sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + .0);
    if (dist < mindist) mindist = dist, ansa = a.id, ansb = b.id;
```

```
}

void rec(int l, int r) {
    if (r - l <= 3) {
    for (int i = l; i <= r; ++i)
        for (int j = i + 1; j <= r; ++j) upd_ans(a[i], a[j]);
    sort(a + l, a + r + 1, &cmp_y);
    return;
}

int m = (l + r) >> 1;
int midx = a[m].x;
rec(l, m), rec(m + 1, r);
inplace_merge(a + l, a + m + 1, a + r + 1, &cmp_y);

static pt t[MAXN];
int tsz = 0;
for (int i = l; i <= r; ++i)
    if (abs(a[i].x - midx) < mindist) {
        for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y < mindist; --j)
            upd_ans(a[i], t[j]);
        t[tsz++] = a[i];
    }
}

sort(a, a + n, &cmp_x);
mindist = 1E20;
rec(0, n - 1);
```

### Distance Between Point and Plane

```
float shortest_distance(float x1, float y1,
                        float z1, float a,
                        float b, float c,
                        float d)
{
    d = fabs((a * x1 + b * y1 +
              c * z1 + d));
    float e = sqrt(a * a + b *
                   b + c * c);
    return d/e;
}
```

## Template

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef vector<ll> vll;
#define FOR(i, a, b) for (ll i = (a); (i) < (b); (i)++)
#define FORN(i, a, b) for (ll i = (a); (i) <= (b); (i)++)
#define ROF(i, a, b) for (ll i = (a); (i) > (b); (i)--)
#define REP(i, n) FOR(i, 0, n)
#define IO cin.sync_with_stdio(false); cin.tie(0); cout.tie(0);
const ll MOD[] = {999727999, 1070777777, 1000000007, 998244353};
mt19937_64 rng(chrono::system_clock::now().time_since_epoch().count());
const int M = MOD[2];
const int inf = (int)1e9;
const ll INF = 1e18;

void solve() {

}

int main() {
    int t=1;
    cin >> t;
    while (t--) solve();
}
```

## VimConfig

```
syntax on
colorscheme desert
set nocompatible, showmatch, hlsearch, noswapfile, ignorecase, autoindent, tabstop=4,
    expandtab, shiftwidth=4, softtabstop=4, relativenumber, number
inoremap { {}<left>
inoremap {<BS> <nop>
inoremap {} {}
inoremap {<Esc> {<Esc>
inoremap {<Enter> {<CR>}<Esc>ko
nnoremap <silent> <Esc> :noh<cr>
```

### Compile Command

```
g++ -std=c++17 -O2 -Wshadow -Wall -o "${1}.out" "${1}.cpp" -g -fsanitize=address -fsanitize=undefined
-D_GLIBCXX_DEBUG
```

## Additional

### Discrete Fourier Transform

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;

    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2*i];
        a1[i] = a[2*i+1];
    }
```

```
    fft(a0, invert);
    fft(a1, invert);

    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}
```