Team Notebook

November 13, 2024

Contents

1 BasicTemplate

1.1 Sagor

```
#include<bits/stdc++.h>
#define ll long long
#define yes cout << "YES" << endl
#define no cout << "NO" << endl
#define testing cout << "testing ";
#define mod 1000000007
#define optimize() ios_base::sync_with_stdio(0);cin.tie(0);
     cout.tie(0);
using namespace std;


void do_the_honour(){


}

int main(){
    optimize();
    int t=1;
    cin>>t;
    for(int z=1;z<=t;z++){
    do_the_honour();
}
    return 0;
}
```

2 Data Structures

2.1 2DBit

```
int bit[1001][1001];
int n;

void update(int x, int y, int val) {
    for (; x <= n; x += (x & (-x))) {
        for (int i = y; i <= n; i += (i & (-i))) { bit[x][i]
             += val; }
    }
}

int query(int x1, int y1, int x2, int y2) {
    int ans = 0;
    for (int i = x2; i; i -= (i & (-i))) {
        for (int j = y2; j; j -= (j & (-j))) { ans += bit[i][
            j]; }
    }
    for (int i = x2; i; i -= (i & (-i))) {
        for (int j = y1 - 1; j; j -= (j & (-j))) { ans -= bit
            [i][j]; }
    }
    for (int i = x1 - 1; i; i -= (i & (-i))) {
        for (int j = y2; j; j -= (j & (-j))) { ans -= bit[i][
            j]; }
    }
    for (int i = x1 - 1; i; i -= (i & (-i))) {
        for (int j = y1 - 1; j; j -= (j & (-j))) { ans += bit
            [i][j]; }
    }
    return ans;
}

int main() {
    iostream::sync_with_stdio(false);
    cin.tie(0);
    int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) {
            char c;
            cin >> c;
            if (c == '*') update(j, i, 1);
        }
    while (q--) {
        int t;
        cin >> t;
        if (t == 1) {
            int x, y;
            cin >> y >> x;
            if (query(x, y, x, y)) update(x, y, -1);
            else update(x, y, 1);
        } else {
            int y1, x1, y2, x2;
            cin >> y1 >> x1 >> y2 >> x2;
            cout << query(x1, y1, x2, y2) << '\n';
        }
    }
    return 0;
}
```

2.2 Bit

```
int bit[100005];
void update(int i,int x){
    for(;i<=n;i+=(i&-i)){
        bit[i]+=x;
    }
}

int sum(int i){

    long ans=0;
    for(;i>0;i-=(i&-i)) ans+=bit[i];
    return ans;
}
```

2.3 kadane

```
sum = ans = ara[0];
int u = 0, v = 0;
for (int K = 1; K < n; K++) {
 if (sum + ara[K] >= ara[K]) {
        sum += ara[K];
        v++;
 } else {
        sum = ara[K];
        u = v = K;
 }
 if (sum > ans) {
        ans = sum;
        ans_l = u;
        ans_r = v;
 }
}
```

2.4 mergesortTree

```


vector<int> treee[800008];

int arr[200002];
//give call init(1,1,n);



void init(int node,int b,int e){

    if(b==e){
        int x=arr[b];
    treee[node].push_back(x);
        return;
    }
    int left=node*2;
    int right =node*2+1;
    int mid=(b+e)/2;

    init(left,b,mid);
    init(right,mid+1,e);

    int i=0;
    int j=0;
    while(i<treee[left].size() and j<treee[right].size()){
        int x=treee[left][i];
        int y=treee[right][j];
        if(x<=y){
            treee[node].push_back(x);
            i++;
        }
        else treee[node].push_back(y),j++;
    }

    while(i<treee[left].size()){
        int x=treee[left][i];
        treee[node].push_back(x);
            i++;
    }

    while(j<treee[right].size()){
        int x=treee[right][j];
        treee[node].push_back(x);
            j++;
    }

}

int query(int node,int b,int e,int i, int j){

    if(i>e or j<b) return 0;
    if(b>=i and e<=j) {
    return treee[node].end()-(upper_bound(treee[node].begin()
         ,treee[node].end(),j));
    }

    int left=node*2;
    int right =node*2+1;
    int mid=(b+e)/2;
    int p1=query(left,b,mid,i,j);
    int p2=query(right,mid+1,e,i,j);


    return p1+p2;
}




void do_the_honour(){

    int n,k;cin >> n >> k;

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

    unordered_map<int,int>mp;
    for(int i=n;i>=1;i--){
        if(mp[a[i]]==0){
           arr[i]=n+1;
           mp[a[i]]=i;
        }
        else{
            arr[i]=mp[a[i]];
            mp[a[i]]=i;
        }
    }

    init(1,1,n);

    while(k--){
        int l,r;cin >> l >> r;
        cout << query(1,1,n,l,r) << endl;
    }

}
```

2.5 MO

```
//1-mo's algo

int blk=700;
int cnt[1000001];
int val=0;
int a[200001];

int lb=0,rb=-1;

bool cmp(tuple<int,int,int>&a,tuple<int,int,int>&b){
    if(get<0>(a)/blk != get<0>(b)/blk)
        return get<0>(a)/blk <= get<0>(b)/blk;
    else return get<1>(a) < get<1>(b);


}

void add(int i){


     val-=cnt[a[i]]*cnt[a[i]]*a[i];
     cnt[a[i]]++;
     val+=cnt[a[i]]*cnt[a[i]]*a[i];


}

void remove(int i){


    val-=cnt[a[i]]*cnt[a[i]]*a[i];
     cnt[a[i]]--;
     val+=cnt[a[i]]*cnt[a[i]]*a[i];
}

void do_the_honour(){

    int n;cin >> n;

    vector<tuple<int,int,int>>v;
int q;cin >> q;

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




    for(int i=0;i<q;i++){
        int l,r;cin >> l >> r;
        l--,r--;
        v.push_back({l,r,i});
    }

    sort(v.begin(),v.end(),cmp);



    int ans[q];

    for(int i=0;i<q;i++){
        auto [l,r,id]=v[i];
        while(lb>l) lb--,add(lb);
        while(rb<r) rb++,add(rb);

        while(lb<l) remove(lb),lb++;
        while(rb>r) remove(rb),rb--;
        ans[id]=val;
    }

    for(auto u:ans) cout << u << endl;

}
```

2.6 OrderedSet

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<class T> using oset=tree<T,null_type,less<T>,
     rb_tree_tag,tree_order_statistics_node_update>;
//oset<int>s;
//*s.find_by_order(x) //(x-1)th largest element;
//s.order_of_key(x) //num of elements strictly less than x
```

2.7 polynomialqueries

```
#define MP make_pair
using ap = pair<int,int>;

struct node{
    ap lazy;
    int sum;
    node(){
        lazy ={0,0};
        sum=0;
    }
};

node merge(node a,node b){

    node ans;
    ans.sum=a.sum+b.sum;
    return ans;

}
node t[800033];
int arr[200004];

void push_down(ap cur, int child){

    t[child].lazy.first += cur.first;
    t[child].lazy.second += cur.second;


}

inline int getnth(ap cur,int n){

    return (cur.first+(n-1)*cur.second);
}

inline int getsum(ap cur,int n){

    return (n*(2*cur.first+(n-1)*cur.second))/2;
}

void push(int id,int l,int r){

        if(t[id].lazy==ap{0,0}) return;

        if(l!=r){
            push_down(t[id].lazy,id<<1);
            int mid=(l+r)/2;
            int n=mid+1-(l-1);
            int newa = getnth(t[id].lazy,n);
            push_down({newa,t[id].lazy.second},id<<1|1);
        }


        t[id].sum += getsum(t[id].lazy,r-l+1);
        t[id].lazy={0,0};



}


void update(int id,int l,int r,int lq,int rq,int a,int d){
    push(id,l,r);
    if(lq>r or rq<l) return;
    if(lq<=l and r<=rq){
        t[id].lazy={(l-lq+1),d};
        push(id,l,r);
        return;
    }
    int mid=(l+r)/2;
    update(2*id,l,mid,lq,rq,a,d);
    update(2*id+1,mid+1,r,lq,rq,a,d);
    t[id]=merge(t[id<<1],t[id<<1|1]);

}


int query(int id,int l,int r, int lq,int rq){


    push(id,l,r);
    if(lq>r or l>rq) return 0;
    if(lq<=l and r<=rq) return t[id].sum;

    int mid = (l + r) / 2;
    return (query(2*id,l,mid,lq,rq)+query(2*id+1,mid+1,r,lq,
         rq));

}


void build(int id,int l,int r){

    if(l==r){
        t[id].sum=arr[l];
        t[id].lazy={0,0};
        return;
    }
    int mid=(l+r)/2;
    build(2*id,l,mid);
    build(2*id+1,mid+1,r);
    t[id]=merge(t[2*id],t[2*id+1]);

}


void do_the_honour(){

    int n;cin >> n;
   int q;cin >> q;

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

    build(1,1,n);
   // cout << query(1,1,n,3,5) << endl;

    while(q--){
        int c,a,b,x; cin >> c;
        if(c==1){
            cin >> a >> b;
            update(1,1,n,a,b,1,1);
        }
        else{
                cin >> a >> b;
            int ans=query(1,1,n,a,b);
            cout << ans << endl;
        }
    }
}
```

2.8 RangeUpdateSetSum

```
struct node{
    int lazy_add;
    int lazy_set;
    int sum;
    node(){
        lazy_add=0;
        lazy_set=0;
        sum=0;
    }
};

node merge(node a,node b){

    node ans;
    ans.sum=a.sum+b.sum;
    return ans;

}

node t[800033];
int arr[200004];

void push_down(int cur, int child){

    if(t[cur].lazy_set!=0){
        t[child].lazy_set=t[cur].lazy_set;
        t[child].lazy_add=0;
    }
    else{
        if(t[child].lazy_set!=0){
            t[child].lazy_set+=t[cur].lazy_add;

        }
        else{
            t[child].lazy_add+=t[cur].lazy_add;
        }
    }

}

void push(int id,int l,int r){

        if(t[id].lazy_add==0 and t[id].lazy_set==0) return;

        if(l!=r){
            push_down(id,id<<1);
            push_down(id,id<<1|1);
        }

        if(t[id].lazy_add!=0){
            t[id].sum += (r-l+1)*t[id].lazy_add;t[id].lazy_add=0;
        }
        else if(t[id].lazy_set!=0){
            t[id].sum = (r-l+1)*t[id].lazy_set; t[id].lazy_set=0;
        }



}


void update(int id,int l,int r,int lq,int rq,int val,int utype){
    push(id,l,r);
    if(lq>r or rq<l) return;
    if(lq<=l and r<=rq){
        if(utype==0){
            t[id].lazy_set=val;
        }
        else{
            t[id].lazy_add+=val;
        }
        push(id,l,r);
        return;
    }

    int mid=(l+r)/2;
    update(2*id,l,mid,lq,rq,val,utype);
    update(2*id+1,mid+1,r,lq,rq,val,utype);
    t[id]=merge(t[id<<1],t[(id<<1)|1]);

}


int query(int id,int l,int r, int lq,int rq){


   push(id,l,r);
    if(lq>r or l>rq) return 0;
    if(lq<=l and r<=rq) return t[id].sum;

    int mid = (l + r) / 2;
    return (query(2*id,l,mid,lq,rq)+query(2*id+1,mid+1,r,lq,rq));

}


void build(int id,int l,int r){

    if(l==r){
        t[id].sum=arr[l];
        t[id].lazy_add=0;
        t[id].lazy_set=0;
        return;
    }
    int mid=(l+r)/2;
    build(2*id,l,mid);
    build(2*id+1,mid+1,r);
    t[id]=merge(t[2*id],t[2*id+1]);

}


void do_the_honour(){

    int n;cin >> n;
    int q;cin >> q;

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

    build(1,1,n);
   // cout << query(1,1,n,3,5) << endl;

    while(q--){
        int c,a,b,x; cin >> c;
        if(c==1){
            cin >> a >> b >> x;
            update(1,1,n,a,b,x,1);
        }
        else if(c==2){
            cin >> a >> b >> x;
            update(1,1,n,a,b,x,0);
        }
        else{
                cin >> a >> b;
            int ans=query(1,1,n,a,b);
            cout << ans << endl;
        }
    }
}

‘‘‘
range update set and sum
```

2.9 Segmenttree

```
int tree[400004];
int arr[100001];
//give call init(1,1,n);
void init(int node,int b,int e){

    if(b==e){
        tree[node]=arr[b];
        return;
    }
    int left=node*2;
    int right =node*2+1;
    int mid=(b+e)/2;

    init(left,b,mid);
    init(right,mid+1,e);
    tree[node]=tree[left]+tree[right];

}

int query(int node,int b,int e,int i, int j){

    if(i>e or j<b) return 0;
    if(b>=i and e<=j) return tree[node];

    int left=node*2;
    int right =node*2+1;
    int mid=(b+e)/2;
    int p1=query(left,b,mid,i,j);
    int p2=query(right,mid+1,e,i,j);
    return p1+p2;
}

void update(int node,int b,int e,int i,int newvalue){
    if(i>e or i<b) return;
    if(b>=i and e<=i){
        tree[node]=newvalue;
        return;
    }
    int left=2*node;
    int right=node*2+1;
    int mid=(b+e)/2;
    update(left,b,mid,i,newvalue);
    update(right,mid+1,e,i,newvalue);
    tree[node]=tree[left]+tree[right];
}

////------lazy propagation-----

struct info{
    ll prop,sum;
}tree[400001];

int query(int node,int b, int e, int i, int j,int carry=0){
    if(i>e or j<b) return 0;
    if(b>=i and e<=j){
        return tree[node].sum+carry*(e-b+1);
    }

    int left=node*2;
    int right=node*2+1;
    int mid=(b+e)/2;

    int p1=query( left, b, mid, i, j,carry+tree[node].prop);
    int p2=query( right, mid+1, e, i,j,carry+tree[node].prop);
    return p1+p2;

}

void update(int node,int b,int e,int i,int j,ll x){
    if(i>e or j<b) return;
    if(b>=i and e<=j){

        tree[node].sum+=((e-b+1)*x);
        tree[node].prop+=x;
        return;
    }

    int left=node*2;
    int right =node*2+1;
    int mid=(b+e)/2;
    update(left,b,mid,i,j,x);
    update(right,mid+1,e,i,j,x);
    tree[node].sum=tree[left].sum+tree[right].sum+(e-b+1)*tree[node].prop;
}
```

2.10 SqrtDecomp

```
int n;
int a[500002];
int blk = 800;
int F[801];

// don't forget to call preprocess();
// static range min query

int getmin(int l, int r)
{

    int lb = l / blk;
    int rb = r / blk;

    int mn = 0;
    if (lb == rb)
    {
        for (int i = l; i <= r; i++)
              mn += a[i];
        return mn;
    }

    for (int i = l; i < min(r, (lb + 1) blk); i++)
        mn += a[i];
    for (int i = lb + 1; i < rb; i++)
        mn += F[i];
    for (int i = blkrb; i <= r; i++)
        mn += a[i];

    return mn;
}

void do_the_honour()
{
    for (int i = 0; i < 600; i++)
        F[i] = 0;
    cin >> n;
    int k;
    cin >> k;
    for (int i = 0; i < n; i++)
    {

        cin >> a[i];
        F[i / blk] += a[i];
    }

    while (k--)
    {
        int l, r;
        cin >> l >> r;
        l--;
        r--;
        cout << getmin(l, r) << endl;
    }
}
```

3 DP

3.1 BitmaskDp

```
//bitmask DP: number of permutaton of length n such that no
     two adjacent elements diff are greater than k

ll int n,k;
vector<ll int>v;

ll dp[17][(1<<17)];
ll func(int id,int mask){

    if(mask==0) return 1;
    if(dp[id][mask]!=-1) return dp[id][mask];
    ll int ans=0;
    for(ll int i=0;i<=(n-1);i++){
        if(abs(i-id)>k) continue;
        if(mask&(1<<i)){
            ans+=func(i,mask^(1<<i));
        }
    }
    return dp[id][mask]=ans;
}

void do_the_honour(){

    cin >> n >> k;
    memset(dp,-1,sizeof dp);

    for(int i=0;i<n;i++) v.push_back(i);

    ll ans=0;
    for(int i=0;i<n;i++) ans+=func(i,((1<<n)-1)^(1<<i)) ;
    cout << ans << endl;
}
```

3.2 DigitDp

```
//digit dp template: find number from a to b such that that
     number and sum of digits of that number are divisible
     by k
ll dp[10][82][2][1000];
int k;
ll func(string &s,int pos=0,int sum=0,int tight=1,int x=0){

    if(pos==s.size()){
        if(x%k==0 and sum%k==0) return 1;
        else return 0;
    }

    if(dp[pos][sum][tight][x]!=-1) return dp[pos][sum][tight
        ][x];
    ll res=0;
    if(tight==1){

        for(int i=0;i<(s[pos]-’0’);i++){
            res+=func(s,pos+1,sum+i,0,(x*10+i)%k);
        }
        res+=func(s,pos+1,sum+(s[pos]-’0’),1,(x*10+(s[pos
            ]-’0’))%k);
    }
    else{
        for(int i=0;i<=9;i++){
            res+=func(s,pos+1,sum+i,0,(x*10+i)%k);
        }
    }

    return dp[pos][sum][tight][x]=res;
}

void do_the_honour(int t){

    memset(dp,-1,sizeof(dp));
    int a,b;cin >> a >> b >>k;
    a--;
    string x=to_string(a);
    string y=to_string(b);

    ll cnt1=func(x);
    memset(dp,-1,sizeof(dp));
    ll cnt2=func(y);
    cout << "Case " << t << ": " << cnt2 - cnt1 << endl;
}
```

4 GeoMetry

4.1 closestpair

```
// To find the closest pair of points
long long
closestPair(vector<pair<long long, long long> > coordinates,
            int n)
{
    // Sort points according to x-coordinates
    sort(coordinates.begin(), coordinates.end());

    // Set to store already processed points whose distance
    // from the current points is less than the smaller
    // distance so far
    set<pair<long long, long long> > s;

    long long squaredDistance = LLONG_MAX;
    long long j = 0;

    for (long long i = 0; i < n; ++i) {
        // Find the value of D
        long long D = ceil(sqrtl(squaredDistance));
        while (coordinates[i].first - coordinates[j].first >=
               D) {
            s.erase({ coordinates[j].second, coordinates[j].
                first });
            j += 1;
        }

        // Find the first point in the set whose y-
             coordinate is less than D distance from ith
             point
        auto start
            = s.lower_bound({ coordinates[i].second - D,
                              coordinates[i].first });
        // Find the last point in the set whose y-
             coordinate is less than D distance from ith
             point
        auto end
            = s.upper_bound({ coordinates[i].second + D,
                              coordinates[i].first });

        // Iterate over all such points and update the
             minimum distance
        for (auto it = start; it != end; ++it) {
            long long dx = coordinates[i].first - it->second;
            long long dy = coordinates[i].second - it->first;
            squaredDistance = min(squaredDistance, 1LL * dx *
                  dx + 1LL * dy * dy);
        }

          // Insert the point as {y-coordinate, x-coordinate}
        s.insert({ coordinates[i].second,
                   coordinates[i].first });
    }
    return squaredDistance;
}
closestPair(v,v.size());
```

4.2 convexhull

```
typedef pair<int, int> Point;

// Function to determine the orientation of the triplet (p,
     q, r).
int orientation(const Point& p, const Point& q, const Point&
      r) {
    int val = (q.second - p.second) * (r.first - q.first) - (
         q.first - p.first) * (r.second - q.second);
    if (val == 0) return -1;         // Collinear
    return (val > 0) ? 1 : -1;       // Clockwise or
         Counterclockwise
}

// Function to compute the convex hull using Andrew's
     monotone chain algorithm
vector<Point> convexHull(vector<Point>& points) {
    int n = points.size();
    if (n < 3) return points;

    // Sort points lexicographically (by x first, then by y)
    sort(points.begin(), points.end());

    vector<Point> hull;

    // Build the lower hull
    for (int i = 0; i < n; ++i) {
        while (hull.size() >= 2 && orientation(hull[hull.size
             () - 2], hull.back(), points[i]) != -1) {
            hull.pop_back();
        }
        hull.push_back(points[i]);
    }

    // Build the upper hull, avoiding the last point of the
         lower hull
    for (int i = n - 2, t = hull.size() + 1; i >= 0; --i) {
        while (hull.size() >= t && orientation(hull[hull.size
             () - 2], hull.back(), points[i]) != -1) {
            hull.pop_back();
        }
        hull.push_back(points[i]);
    }

    hull.pop_back(); // Remove the last point because it is
         the same as the first one

    return hull;
}
```

4.3 Linesegment

```
// Function to find the orientation of the ordered triplet (
     p, q, r)
// The function returns:
// 0 -> p, q and r are collinear
// 1 -> Clockwise
// 2 -> Counterclockwise
int orientation(pair<int, int> p, pair<int, int> q, pair<int
     , int> r) {
    int val = (q.second - p.second) * (r.first - q.first) - (
         q.first - p.first) * (r.second - q.second);
    if (val == 0) return 0;
    return (val > 0) ? 1 : 2;
}

// Function to check if point q lies on segment pr
bool onSegment(pair<int, int> p, pair<int, int> q, pair<int,
      int> r) {
    if (q.first <= max(p.first, r.first) && q.first >= min(p.
         first, r.first) &&
        q.second <= max(p.second, r.second) && q.second >=
             min(p.second, r.second))
        return true;
    return false;
}

// Function to check if the line segments 'p1q1' and 'p2q2'
     intersect
bool doIntersect(pair<int, int> p1, pair<int, int> q1, pair<
     int, int> p2, pair<int, int> q2) {
    // Find the four orientations needed for general and
         special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are collinear and p2 lies on segment
         p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and q2 are collinear and q2 lies on segment
         p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are collinear and p1 lies on segment
         p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;

    // p2, q2 and q1 are collinear and q1 lies on segment
         p2q2
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;

    // Doesn't fall in any of the above cases
    return false;
}

void do_the_honour() {
    int x1, y1, x2, y2, x3, y3, x4, y4;
    cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;

    pair<int, int> p1 = {x1, y1}, q1 = {x2, y2}, p2 = {x3, y3
         }, q2 = {x4, y4};

    if (doIntersect(p1, q1, p2, q2)) yes;
    else no;
}
```

4.4 pointlocation

```
void do_the_honour() {
    int x1, y1, x2, y2, x3, y3;
    cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

    int cross_product = (x2 - x1) * (y3 - y2) - (y2 - y1) * (
         x3 - x2);

    if (cross_product == 0) {
        cout << "TOUCH" << endl;
    } else if (cross_product > 0) {
        cout << "LEFT" << endl;
    } else {
        cout << "RIGHT" << endl;
    }
}
```

4.5 polygonLatticepoints

```
int area(vector<pair<int, int>> vertices){


    int n=vertices.size();



    int area = 0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        area += vertices[i].first * vertices[j].second;
        area -= vertices[i].second * vertices[j].first;
    }

    return abs(area);


}


int latticeonboundary(pair<int,int>p1,pair<int,int>p2){

    int a=abs(p1.first-p2.first);
    int b=abs(p1.second-p2.second);

    return (__gcd(a,b)-1);

}



//A=i+b/2-1
void do_the_honour(){
    int n;cin >> n;
    vector<pair<int,int>>vp;
    for(int i=0;i<n;i++){
        int x,y; cin >> x >> y;
        vp.push_back({x,y});
    }

    int total=area(vp)/2;

    int boundary=0;

    for(int i=0;i<n;i++){
        boundary+= latticeonboundary(vp[i],vp[(i+1)%n]);
    }

    boundary+=n;

    cout << total+1-(boundary/2) << " " << boundary << endl;

}
```

5 Graph

5.1 BeckedgeDetection

```
void Graph::dfs(int v, int parent) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    subtreeSize[v] = 1;

    for (int to : adj[v]) {
        if (to == parent) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v]) {
                // Back edge detected between v and to
                int x = subtreeSize[to];
                int y = vertices - x;
                totalPairs = min(totalPairs, calculatePairs(x)
                      + calculatePairs(y));
            }
            subtreeSize[v] += subtreeSize[to];
        }
    }
}

void Graph::countSubtreeSizes(int v, int parent) {
    visited[v] = true;
    subtreeSize[v] = 1;
    for (int to : adj[v]) {
        if (to != parent && !visited[to]) {
            countSubtreeSizes(to, v);
            subtreeSize[v] += subtreeSize[to];
        }
    }
}
```

5.2 BellMenFord

```
//bellmen ford for negative cycle and printing suc a cycle
class triplet{
public:
    int first;
    int second;
    int third;

};
int n, m;
vector<triplet> edges;
vector<int> dist;
vector<int> relaxant;
void bellman_ford()
{
    int x;
    for(int i = 1; i <= n; ++i)
    {
        x = -1;
        for(auto e: edges)
        {

            int u = e.first;
            int v = e.second;
            int d = e.third;
            if(dist[u]+d < dist[v])
            {
                dist[v] = d+dist[u];
                relaxant[v] = u;
                x = v;
            }
        }
    } // n relaxations

    if(x == -1)
```

```
        cout << "NO" << endl;

    else
    {
        for(int i = 1; i <= n; ++i)
        {
            x = relaxant[x];
        }

        vector<int> cycle;

        for(int v = x; ; v = relaxant[v])
        {
            cycle.push_back(v);
            if(v == x and cycle.size() > 1)
                break;
        }

        reverse(cycle.begin(), cycle.end());

        cout << "YES" << endl;
        for(auto v: cycle)
        {
            cout << v << " ";
        }

        cout << endl;
    }


}
int32_t main()
{

    cin >> n >> m;
    dist.resize(n+1);
    relaxant.resize(n+1);
    edges.resize(m);

    for(int i = 0; i < m; ++i)
    {
        struct triplet inp;
        cin >> inp.first >> inp.second >> inp.third;
        edges[i] = inp;
    }

    for(int i = 1; i <= n; ++i)
    {
        relaxant[i] = -1;
    }
```

```
    bellman_ford();
}
```

5.3 BipartiteMatching

```
 int k;
    int m; cin >> n >> m >>k;
    int src=0,dest=n+m+1;
    for (int i = 0; i < k; i++) {
        int x, y, w = 1; cin >> x >> y;
        capacity[x][y+n] = w;


        adj[x].push_back(y+n);
        adj[y+n].push_back(x);
        adj2[x][y+n] = 1;


    }

    for(int i=1;i<=n;i++)
    {
        adj[src].push_back(i);
        adj[i].push_back(src);
        capacity[src][i]=1;
        adj2[0][i]=1;
    }
    for(int i=1;i<=m;i++)
    {

        adj[i+n].push_back(dest);
        adj[dest].push_back(i+n);
        capacity[i+n][dest]=1;
        adj2[n+i][dest]=1;
    }

    cout << maxflow(src, dest) << endl;

    fill(vis, vis + 10001, 0); // Reset the visited array
    dfs(0);

    for (int i = 1; i <= n; i++) {
        for (int j = n+1; j <= n+m; j++) {
            if ((capacity[i][j]==0) and (adj2[i][j]==1)) {
                ans.push_back({i, j-n});
            }
        }
    }
```

```
    for (auto u : ans) cout << u.first << " " << u.second <<
         endl;
```

5.4 cycle in directed graph

```
vector<int>adj[200001];
vector<vector<int>>v;
int par[200001];
int color[200001];
bool f=0;
void dfs(int node,int pp){

    if(color[node]==2) return;
    if(f) return;
    if(color[node]==1){
        vector<int>x;
        int p=pp;
        x.push_back(p);
        while(p!=node){          p=par[p];

            x.push_back(p);
        }
        reverse(x.begin(),x.end());
        v.push_back(x);
        if(x.size()>=2) f=1;
        x.clear();
        return ;
    }
    color[node]=1;
    par[node]=pp;

    for(auto u:adj[node]){
        if(u==pp) continue;
        dfs(u,node);
    }
    color[node]=2;

}
void do_the_honour(){

    int n,m;cin >> n >> m;

    map<pair<int,int>,bool>mp;

    int b=-1,cc=-1;
    for(int i=0;i<m;i++){
        int x,y; cin >> x >> y;
        adj[x].push_back(y);
        mp[{x,y}]=1;
        if(mp[{y,x}]){
            b=x,cc=y;
        }
        //adj[y].push_back(x);
    }
    if(b!=-1){
        cout << 3 << endl;
        cout << b << " " << cc << " " << b << endl;
        return;
    }

    for(int i=1;i<=n;i++){
        if(color[i]!=0) continue;
        dfs(i,0);
    }

    int sz=0;
    vector<int>ans;
    for(auto u:v){
        if(sz<u.size()){
            sz=u.size();
            ans=u;
        }
    }
    if(sz>=3){
        cout << ans.size()+1 << endl;
        for(auto u:ans) cout << u << " ";
        cout << ans[0] << endl;
        cout << endl;

    }
    else cout << "IMPOSSIBLE" << endl;



}
```

5.5 De Brujin

```
unordered_set<string> seen;
vector<int> edges;
void dfs(string node, int& k, string& A)
{
    for (int i = 0; i < k; ++i) {
        string str = node + A[i];
        if (seen.find(str) == seen.end()) {
            seen.insert(str);
            dfs(str.substr(1), k, A);
            edges.push_back(i);
        }
    }
}
 string deBruijn(int n, int k, string A)
{


    seen.clear();
    edges.clear();

    string startingNode = string(n - 1, A[0]);
    dfs(startingNode, k, A);

    string S;

    // Number of edges
    int l = pow(k, n);
    for (int i = 0; i < l; ++i)
        S += A[edges[i]];
    S += startingNode;

    return S;
}

// Driver code
int main()
{
    int n = 3, k = 2;
    cin >> n;
    string A = "01";
    cout << deBruijn(n, k, A);
    return 0;
}
```

5.6 Dijkstra with kth minimum path

```
const int N=2e5+3;
const int INF=1e15+10;
vector<pair<int,int>>g[N];
priority_queue<int>d[200001];
void dijkstra(int k){

    vector<int>dist(N,INF);
    vector<bool>vis(N,0);

    multiset<pair<int,int>>st;

    st.insert({0,1});

    while(st.size()>0){
        auto node=*st.begin();
        st.erase(st.begin());
        int v=node.second;
        int v_dis=node.first;

        if(d[v].top()<v_dis) continue;

        for(auto child:g[v]){
            int child_v=child.first;
            int wt=child.second;
            if(v_dis+wt<d[child_v].top()){
                d[child_v].pop();
                int l=v_dis+wt;
                d[child_v].push(v_dis+wt);
                st.insert({l,child_v});


            }
        }
    }

}
void do_the_honour(){

    int n,m,k;cin >> n >> m >> k;

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

    for(int i=1;i<=n;i++){
        for(int j=1;j<=k;j++){
            d[i].push(INF);
        }
    }
    d[1].pop();
    d[1].push(0);
    dijkstra(k);
    vector<int>v;
    //cout << d[n].size() << endl;
    while(!d[n].empty()){
        v.push_back(d[n].top());
        d[n].pop();
    }
    while(!v.empty()){
        cout << v.back() << " ";
        v.pop_back();
    }
    cout << endl;

}
```

5.7 DSU

```
const int N=2e5+10;
int parent[N];
int Size[N];

void make(int v) {parent[v]=v;Size[v]=1;}

int Find(int v){
    if(v==parent[v]) return v;
    return (parent[v]=Find(parent[v]));
}

void Union(int a,int b){
    a=Find(a);
    b=Find(b);
    if(Size[a]<Size[b]) swap(a,b);
    if(a!=b) parent[b]=a;
    Size[a]+=Size[b];
}
```

5.8 Euler Circuit in undirected graph

```
void dfs(int node){

    set<int>&ev=adj[node];
    while(!ev.empty()){
        int u=*ev.begin();
        ev.erase(ev.begin());
        adj[u].erase(node);
        dfs(u);
    }
    path.push_back(node);

}

void do_the_honour(){

    int n,m;cin >> n >> m;
    for(int i=0;i<m;i++){
        int x,y;cin >> x >> y;
        adj[x].insert(y);
        adj[y].insert(x);
    }

    for(int i=1;i<=n;i++){
        if(adj[i].size()&1){
            cout << "IMPOSSIBLE" << endl;//every node must have even degree;
            return;
        }
    }

    dfs(1);
    if(path.size()!=m+1){
        cout << "IMPOSSIBLE" << endl;
    }
    else{
        for(auto u:path) cout << u << " ";
    }
}
```

5.9 Euler tour in directed graph

```
vector<pair<int,int>>vp;
vector<int>adj[200001];
set<int>seen;
vector<int>path;
void dfs(int node){

    while(!adj[node].empty()){
        int x=adj[node].back();
        adj[node].pop_back();
            dfs(x);

    }
    path.push_back(node);
}

void do_the_honour(){

    int n,m;cin >> n >> m;

    int in[n+1]={0},out[n+1]={0};
    for(int i=0;i<m;i++){
        int x,y;cin >> x >> y;
        vp.push_back({x,y});
        in[y]++,out[x]++;
        adj[x].push_back(y);
    }
    int c=0;
    int s,e,ss=0,ee=0;
    for(int i=1;i<=n;i++){
        if(in[i]-out[i]==0) c++;
        if(in[i]-out[i]==1) ee++,e=i;
        if(in[i]-out[i]==-1) ss++,s=i;
    }

    if(c!=n-2 or s!=1 or e!=n or ee!=1 or ss!=1) cout << "IMPOSSIBLE";
    else{
        dfs(1);
        if(path.size()!=m+1) cout << "IMPOSSIBLE";
        else{
            reverse(path.begin(),path.end());
            for(auto u:path) cout << u << " ";
        }
    }
}
```

5.10 HamiltonTour

```
//visits each node exactly once
int n,m;
vector<int>adj[2000];
int dp[22][1<<22];
int dfs(int node,int mask){

    if(mask==(1<<(n))-1) return 1;
    else if(node==n-1) return 0;

    if(~dp[node][mask]) return dp[node][mask];

    int ans=0;
    for(auto u:adj[node]){
        if(mask&(1<<u)) continue;
        ans+=dfs(u,mask|(1<<u));
        ans%=mod;
    }

    return dp[node][mask]=ans;

}
```

5.11 Hopcroft

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

```
vector<int>adj[mx];
map<pair<int,int>,int>cost;
int pairU[mx],pairV[mx],dist[mx],a,b;
bool bfs()
{
    queue<int> Q;
    for (int u=1; u<=a; u++)
    {
        if (pairU[u]==0)
        {
            dist[u] = 0;
            Q.push(u);
        }
        else dist[u] = inf;
    }
    dist[0] = inf;
    while (!Q.empty())
    {
        int u = Q.front();
        Q.pop();
        if (dist[u] < dist[0])
        {
            for (auto it : adj[u])
            {
                int v = it;
                if (dist[pairV[v]] == inf)
                {
                    dist[pairV[v]] = dist[u] + 1;
                    Q.push(pairV[v]);
                }
            }
        }
    }
    return (dist[0] != inf);
}
bool dfs(int u)
{
    if (u != 0)
    {
         for (auto it : adj[u])
            {
                int v = it;
            if (dist[pairV[v]] == dist[u]+1)
            {

                if (dfs(pairV[v]) == true)
                {
                    pairV[v] = u;
                    pairU[u] = v;
                    return true;
                }
            }
        }
        return false;
    }
    return true;
}

int hopcroftKarp()
{
    for (int u=0; u<=a; u++)
        pairU[u] = 0;
    for (int v=0; v<=a; v++)
        pairV[v] = 0;
    int result = 0;
    while (bfs())
    {
        for (int u=1; u<=a; u++)

            if (pairU[u]==0 && dfs(u)){
                result++;
            }
    }
    return result;
}
 main()
{
    cin>>a>>b;
    int x,y;
    for(int i=0;i<b;i++){
        cin>>x>>y;
        adj[y].push_back(x);
        int cost;
        cin>>cost;
        cost[{y,x}]=cost;
    }
    int ans=hopcroftKarp();
}
```

5.12 kosaraju

```
#include<bits/stdc++.h>
#define ll long long
#define yes cout << "YES" << endl
#define no cout << "NO" << endl
#define testing cout << "testing ";
#define mod 1000000007
#define optimize() ios_base::sync_with_stdio(0);cin.tie(0);
     cout.tie(0);
using namespace std;

vector<int>adj[200010];
vector<int>trans[200010];
vector<bool> vis(200010,false);
vector<int>v,x;

void dfs(int node){

    vis[node]=1;
    for(auto u:adj[node]){
        if(vis[u]) continue;
        dfs(u);
    }
    v.push_back(node);
}

void dfs2(int node){


    //cout << node << " ";
    vis[node]=1;
    for(auto u:trans[node]){
        if(vis[u]) continue;
        dfs2(u);
    }
    x.push_back(node);
}

void do_the_honour(){

    int n;cin >> n;
    int m;cin >> m;

    for(int i=0;i<m;i++){
        int x,y; cin >> x >> y;
        adj[x].push_back(y);
        trans[y].push_back(x);
    }



    v.clear();
    for(int i=1;i<=n;i++) if(vis[i]==0) dfs(i);

    for(int i=0;i<200010;i++) vis[i]=false;

    reverse(v.begin(),v.end());

    int cc=0;
    vector<vector<int>>ans;
    int xx=0,xy=0;
    for(auto u:v){
        if(vis[u]) continue;
        if(ans.size()==0) xx=u;
        else xy=u;
        x.clear();
        dfs2(u);
        ans.push_back(x);
        cc++;
       // cout << endl;
    }
    if(ans.size()>1){
        cout << "NO" << endl;
        cout << xy<< " " << xx << endl;
    }
    else{
        cout << "YES" << endl;
    }

}

void reset(){
     for(int i=0;i<200010;i++){
        adj[i].clear();
        vis[i]=0;
        v.clear();
        trans[i].clear();
     }
}

int main(){
    optimize();
    int t=1;
   // cin>>t;
    for(int z=1;z<=t;z++){

    reset();
    do_the_honour();


}
    return 0;
}
```

5.13 LCA and kth ancestor

```
const int MAXN = 200001;
const int LOG = 20;
int dp[LOG][MAXN];
int level[MAXN];
vector<int> adj[MAXN];
void dfs(int node, int par) {
    dp[0][node] = par;
    for (int i = 1; i < LOG; i++) { // Fixed boundary to LOG
        dp[i][node] = dp[i - 1][dp[i - 1][node]];
    }
    for (auto u : adj[node]) {
        if (u == par) continue;
        level[u] = level[node] + 1;
        dfs(u, node);
    }
}
adj[x].push_back(y);
adj[y].push_back(x);
level[1] = 0;
dfs(1, 0);
dp[0][1] = 0;

if (level[a] < level[b]) swap(a, b), swap(x, y);
int diff = level[a] - level[b];

for (int i = 0; i < LOG; i++) { // Fixed boundary to LOG
if (diff & (1 << i)) {
a = dp[i][a];
}
}
if (a == b) {
node = a;
} else {
for (int ii = LOG - 1; ii >= 0; ii--) { // Fixed boundary to
      LOG
if (dp[ii][a] != dp[ii][b]) {
a = dp[ii][a];
b = dp[ii][b];
}
}
node = dp[0][a];
}
```

5.14 MaxFlow

```
const int INF=1e18;
int n;
vector<vector<int>> capacity(1001,vector<int>(1001,0));
vector<vector<int>> adj(1001);
vector<vector<int>> adj3(1001);
int adj2[501][501];
int vis[10001];
int vis2[1001];
vector<pair<int,int>>ans;
void dfs(int node){

    vis[node]=1;
    for(auto u:adj[node]){
        if(vis[u] or capacity[node][u]<=0) continue;
        dfs(u);
    }
}
int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        int flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(n+1);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}



void do_the_honour(){

    int m;cin >> n >> m;
    for(int i=0;i<m;i++){
        int x,y,w=1; cin >> x >> y ;
        capacity[x][y]+=w;
        capacity[y][x]+=w;
        adj[x].push_back(y);
        adj[y].push_back(x);
        adj2[x][y]=1;
        adj2[y][x]=1;


        adj3[x].push_back(y);
    }





    cout << maxflow(1,n) << endl;
    dfs(1);
    for(int i=1;i<=n;i++) for(int j=1;j<=n;j++) if(adj2[i][j]
          and vis[i] and !vis[j]) ans.push_back({i,j});

    //cout << ans.size() << endl;
    for(auto u:ans) cout << u.first << " " << u.second <<
         endl;

‘‘‘

finding max flow and cut edges for undirected graph
```

5.15 num of path with minimum weight

```
 if(dist[v]+wt<dist[child_v]){
                dist[child_v]=dist[v]+wt;
                route[child_v]=route[v];
                min_f[child_v]=min_f[v]+1;
                max_f[child_v]=max_f[v]+1;
                st.push({dist[child_v],child_v});
            }
            else if(dist[v]+wt==dist[child_v]){
                route[child_v]+=route[v];
                route[child_v]%=mod;
                min_f[child_v]=min(min_f[v]+1,min_f[child_v]);
                max_f[child_v]=max(max_f[v]+1,max_f[child_v]);

            }
```

6 Math

6.1 bineexp

```
ll binexp(ll int a,ll int b,ll int m){
    ll res=1;
    while(b>0){
        if(b&1){
            res=(res*1LL*a)%m;
        }
        a=(a*1LL*a)%m;
        b>>=1;
    }
    return res;
}
```

6.2 BitwiseSieve

```
// Bitwise - sieve :- prime number generator;
// 0 for prime, 1 for not prime; >> Not sure ... #define MX
     10000000
int marked[MX / 64 + 2];
vector<int> primes;
#define mark(x) marked[x >> 6] |= (1 << ((x & 63) >> 1))
#define check(x)(marked[x >> 6] & (1 << ((x & 63) >> 1)))
bool isPrime(int x) { return (x > 1) && ((x == 2) || ((x &
     1) && (!(check(x))))); }
void sieve(int n) {
 int i, j;
 primes.push_back(2);
 for (i = 3; i * i <= n; i += 2) {
    if (!check(i)) {
     primes.push_back(i);
      for (j = i * i; j <= n; j += i << 1) mark(j);
    }
 }
}
```

6.3 divisors

```
// Maximal number of divisors of any n-digit number:
// First 18 numbers: 4, 12, 32, 64, 128, 240, 448, 768,
     1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472,
     64512, 103680
// Number of divisors...
// Euler's totient function
// first, run a sieve for value sqrt(n);
vector<pair<int, int> > divisors;
void divs(int n) {
 int cnt, tot = 1, i;
 for (i = 0; i < (int)primes.size() && (primes[i] * primes[i
      ]) <= n; i++) {
    if (n % primes[i] == 0) {
     cnt = 1;
           while (n % primes[i] == 0) {
           n /= primes[i];
           cnt++;
    }
           divisors.push_back(make_pair(primes[i], cnt - 1));
           tot *= cnt;
    }
 }
 if (n > 1) {
         tot *= 2;
         divisors.push_back(make_pair(n, 1));
 }
 printf("Number of divisors %d\n", tot);
 for (i = 0; i < (int)divisors.size(); i++) printf("%d %d\n"
      , divisors[i].first, divisors[i].second);
}
// Number of divisors...
ll NOD(int n) {
 ll ans = 1;
 for (int K = 0; K < sz(divisors); K++) {
    ans *= (ll)(divisors[K].se + 1);
 }
 return ans;
}
// Sum of divisors...
ll SOD(int n) {
 ll ans = 1, cnt;
 for (int K = 0; K < sz(divisors); K++) {
    cnt = divisors[K].fi;
    while (divisors[K].se--) cnt *= (ll)divisors[K].fi;
    ans *= (ll)(cnt - 1) / (divisors[K].fi - 1);
 }
 return ans;
}
```

6.4 FFT

```
const int N = 3e5 + 9;

const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w * wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;
}
vector<long long> multiply(vector<int> &a, vector<int> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a);
  while((int)ret.size() > 1 && ret.back() == 0) ret.pop_back();
  return ret;
}
```

6.5 LinearSieve

```
const int MX = 10000001;
int lp[MX];
vector<int> pr;
void linear_sieve() {
  for(int i = 2; i < MX; ++i) {
    if(lp[i] == 0) {lp[i] = i; pr.PB(i);}
    for(int j = 0; j < (int)pr.size() && pr[j] <= lp[i] && (i * pr[j]) < MX; ++j) lp[i * pr[j]] = pr[j];
  }
}
```

6.6 MatrixExp

```
array<ll,4> mul(array<ll,4>x, array<ll,4>y){

    ll a=x[0],b=x[1],c=x[2],d=x[3];
    ll e=y[0],f=y[1],g=y[2],h=y[3];

    ll m=((a*e)%mod+(b*g)%mod)%mod;
    ll n=((a*f)%mod+(b*h)%mod)%mod;
    ll o=((c*e)%mod+(d*g)%mod)%mod;
    ll p=((c*f)%mod+(d*h)%mod)%mod;
    return {m,n,o,p};


}

array<ll,4> Find(ll n){

    if(n==1) return {1,1,1,0};
    array<ll,4>x=Find(n/2);

    if(n&1) return mul(mul(x,x),Find(1));
    else return mul(x,x);

}

void do_the_honour(){

    ll int n;cin >> n;

    if(n==0) cout << 0;
    else if(n==1) cout << 1;
    else if(n==2) cout << 1;
    else if(n==3) cout << 2;
    else{
        cout << Find(n-1)[0];
    }

}
```

6.7 nCrWithArray

```
// Complexity O(n)
const int MX = 1e6;
ll fact[MX + 5], inv[MX + 5];
void factorial() {
 fact[0] = fact[1] = 1;
 for (ll K = 1; K <= MX; K++) fact[K] = fact[K - 1] * K % mod;
 inv[MX] = inverse(fact[MX], mod);
 for (ll K = MX - 1; K >= 1; K--) inv[K] = inv[K + 1] * (K + 1) % mod;
 inv[0] = 1;
}
// function call ...
factorial();
cout << (fact[18] * inv[5] % mod) * inv[13] % mod << "\n";
// Combinatorics problems can be done using stars and bars. The number of ways to put n identical objects into k labeled boxes is (n+k-1)C(n)
// while counting nCr where n,r size is big to get rid of overflow we can do this :
// Code:
minncr(k,n-k);
ll ans=1;
for(ll i=1;i<=k;i++){
   ans = (ans*(n-i+1) )/ i ;
}
```

6.8 SegmentedSieve

```
// Block/Segmented Sieve:
// Output: prime list in range {L, R}; (R-L+1) <= 1e7 && R <= 1e12;
```

```
vector<char> segmentedSieve(long long L, long long R) { //
     generate all primes up to sqrt(R)
 long long lim = sqrt(R);
 vector<char> mark(lim + 1, false);
 vector<long long> primes;
 for (long long i = 2; i <= lim; ++i) {
    if (!mark[i]) {
    primes.emplace_back(i);
    for (long long j = i * i; j <= lim; j += i) mark[j] =
         true;
    }
 }
 vector<char> isPrime(R - L + 1, true);
 for (long long i : primes)
  for (long long j = max(i * i, (L + i - 1) / i * i); j <= R
       ; j += i) isPrime[j - L] = false;
 if (L == 1) isPrime[0] = false;
 return isPrime;
}
```

6.9 Sieve

```
// sieve :- prime number generator;
// 0 for prime, 1 for not prime;
vector<int> primes;
vector<bool> mark(1000002);
void sieve(int n) {
 int i, j, limit = sqrt(n*1.) + 2;
 mark[1] = 1;
 for(i = 4; i<=n; i+=2) mark[i] = 1;
 primes.push_back(2);
 for(i = 3; i <= n; i += 2){
    if(!mark[i]){
    primes.push_back(i);
     if(i<=limit){
       for(j = i*i; j <= n; j += i*2){
         mark[j] = 1;
       }
      }
    }
 }
}
```

6.10 TotientFunction

```
#define MX 1e5+6;
// phi pre-calculation function . . .
ll phi[MX];
void phi_function()
{
    memset(phi, 0, sizeof phi);
    phi[1] = 1;
    for(int i=2; i<=MAX; ++i){
     if(phi[i]==0){
             phi[i]=i-1;
        for(int j=i+i; j<=MAX; j+=i){
            if(phi[j]==0) phi[j] = j;
              phi[j]= phi[j]/i * (i-1);
             }
     }
    }
}
// gcd sum calculation function . . .
ll table[MAX];
void div()
{
 memset(table, 0, sizeof table);
 for(int i=1;i<MAX;i++){
             for(int j=i+i;j<MAX;j=j+i){
           table[j]+=(i*phi[j/i]);
         }
 }
 for(int i=2;i<MAX;i++)
         table[i]+=table[i-1];
}
```

7 String

7.1 Manacher

```
vector<int> manacher_odd(string s) {
  // size of paliendriom at center s[i] = 2 * p[i] + 1
  int n = s.size();
  s = "$" + s + "^";
  vector<int> p(n + 2);
  int l = 1, r = 1;
  for(int i = 1; i <= n; i++) {
    p[i] = max(0, min(r - i, p[l + (r - i)]));

    while(s[i - p[i]] == s[i + p[i]]) {
      p[i]++;
    }
    if(i + p[i] > r) {
      l = i - p[i], r = i + p[i];
    }
  }
  return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
  // size of odd paliendriom at center s[i] = p[2*i]
  // size of even paliendriom at center s[i]s[i+1] = p[2*i
       +1]
  string t;
  for(auto c: s) t += string("#") + c;
  auto res = manacher_odd(t + "#");
  return vector<int>(begin(res) + 1, end(res) - 1);
}
```

7.2 prefixFunction

```
vector<int> prefix_function(string s) {
    int n = s.length();
    vector<int> pi(n);
    pi[0] = 0;
    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;
}
```

7.3 StringHashing

```
const int N = 2e5 + 9;
const int mod1 = 1e9+21, mod2 = 1e9+9;
const int p1 = 37, p2 = 31;

pair<int,int> pw[N], invpw[N];
int bigMod(int a,int b,int mod) {
    int ans = 1;
    while(b) {
        if(b & 1) {
            ans = 1LL * ans * a % mod;
        }
        a = 1LL * a * a % mod;
        b >>= 1;
    }
    return (ans % mod);
}
void preCalc() {
    pw[0] = {1, 1};
    for(int i = 1; i < N; i++) {
        pw[i].first = 1LL * pw[i-1].first * p1 % mod1;
    }
    for(int i = 1; i < N; i++) {
        pw[i].second = 1LL * pw[i-1].second * p2 % mod2;
    }
    int invpw1 = bigMod(p1, mod1 - 2, mod1);
    int invpw2 = bigMod(p2, mod2 - 2, mod2);
    invpw[0] = {1, 1};
    for(int i = 1; i < N; i++) {
        invpw[i].first = 1LL * invpw[i-1].first * invpw1 %
            mod1;
        invpw[i].second = 1LL * invpw[i-1].second * invpw2 %
            mod2;
    }
}

class Hash_node {
public:
    int n;
    string s;
    vector<pair<int,int>> hs, prehs;
    Hash_node(){}
    Hash_node(string _s) {
        s = _s;
        n = _s.size();
        hs.resize(n);
        for(int i = 0; i < n; i++) {
            hs[i].first += 1LL * s[i] * pw[i].first % mod1;
            hs[i].first %= mod1;
        }
        for(int i = 0; i < n; i++) {
            hs[i].second += 1LL * s[i] * pw[i].second % mod2;
            hs[i].second %= mod2;
        }
        prehs.resize(n);
        prehs[0].first = hs[0].first;
        for(int i = 1; i < n; i++) {
            prehs[i].first = prehs[i-1].first + hs[i].first;
            prehs[i].first %= mod1;
        }
        prehs[0].second = hs[0].second;
        for(int i = 1; i < n; i++) {
            prehs[i].second = prehs[i-1].second + hs[i].
                second;
            prehs[i].second %= mod2;
        }
    }
    pair<int,int> getHash(int i,int j) {
        assert(i <= j);
        pair<int,int> ans({0, 0});
        if(i == 0) {
            ans.first = prehs[j].first;
            ans.second = prehs[j].second;
        }
        else {
            ans.first = (prehs[j].first - prehs[i-1].first +
                mod1) % mod1;
            ans.first = 1LL * ans.first * invpw[i].first %
                mod1;
            ans.second = (prehs[j].second - prehs[i-1].second
                 + mod2) % mod2;
            ans.second = 1LL * ans.second * invpw[i].second %
                 mod2;
        }
        return ans;
    }
};
‘‘‘
```

7.4 SuffixArray

```
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);

    for (int i = 0; i < n; i++)
        cnt[s[i]]++;

    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i - 1];

    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;

    c[p[0]] = 0;
    int classes = 1;

    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]])
            classes++;

        c[p[i]] = classes - 1;
    }

    vector<int> pn(n), cn(n);

    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }

        fill(cnt.begin(), cnt.begin() + classes, 0);

        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;

        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i - 1];

        for (int i = n - 1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];

        cn[p[0]] = 0;
        classes = 1;

        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)
                ) % n]};
            pair<int, int> prev = {c[p[i - 1]], c[(p[i - 1] +
                 (1 << h)) % n]};
            if (cur != prev)
                ++classes;

            cn[p[i]] = classes - 1;
        }

        c.swap(cn);
        if (classes == n)
            break;
    }

    return p;
}

vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

vector<int> lcp_construction(string const& s, vector<int>
     const& p) {
    int n = s.size();
    vector<int> rank(n, 0);

    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n - 1, 0);

    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }

        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i + k] == s[j + k
             ])
            k++;

        lcp[rank[i]] = k;

        if (k)
            k--;
    }

    return lcp;
}

//int main() {
//    string s;
//    cin >> s;
//    vector<int> v = suffix_array_construction(s);
//
//    for (int i = 0; i < v.size(); i++)
//        cout << v[i] << " ";
//
//    cout << endl;
//
//    vector<int> lcp = lcp_construction(s, v);
//
//    for (int i = 0; i < lcp.size(); i++)
//        cout << lcp[i] << " ";
//
```

```
//    return 0;
//}



int check(int n,int mid, string& pattern,vector<int>&v,
     vector<int>&lcp,string &s)
{
    int flag = -1, patternSize = pattern.size(),
        suffixStart = v[mid];

    // Check if the suffix can contain the entire pattern
    if (n - suffixStart >= patternSize)
        flag = 0;

    // Compare characters of the pattern and suffix
    for (int i = 0; i < min(n - suffixStart, patternSize);
         i++) {
        if (s[suffixStart + i] < pattern[i])
            return -1;
        if (s[suffixStart + i] > pattern[i])
            return 1;
    }
    return flag;
}



int countocc(string &pat,int n,vector<int>&v,vector<int>&lcp
     ,string &s){


    int left=0,right=n-1;
    int answer=-1;
    int l=left,r=right;

    while(l<=r){

        int mid=(l+r)/2;
        int num=check(n,mid,pat,v,lcp,s);

        if(num==0){
            answer=mid;
            r=mid-1;
        }
        else if(num==1){
            r=mid-1;
        }
```

```
        else{
            l=mid+1;
        }

    }

    if(answer==-1){
        return 0;
    }
    left=answer;
    l=left,r=right;
     answer=left;


        while(l<=r){

        int mid=(l+r)/2;
        int num=check(n,mid,pat,v,lcp,s);

        if(num==0){
            answer=mid;
            l=mid+1;
        }
        else if(num==-1){
            l=mid+1;
        }
        else{
            r=mid-1;
        }

    }

    right=answer;
    return right-left+1;


}


void do_the_honour(){

    string s,t;cin >> s ;
    string demo=s;

    vector<int> v = suffix_array_construction(demo);
    vector<int> lcp = lcp_construction(demo,v);

    //for(auto u:v) cout << u << " ";
    //cout << endl;
    int k;cin >> k;
    int n=s.size();

    for(int i=0;i<v.size();i++){
        if(i==0){
            int sz=n-v[i];
            if(sz>=k){
                for(int i=v[i];i<=(v[i]+k-1);i++) cout << s[i];
                cout << endl;return;
            }
            else k-=sz;
        }
        else{

            int sz=n-v[i]-lcp[i-1];
            if(sz>=k){
                cout << s.substr(v[i],lcp[i-1]+k) << endl;
                return;
            }
            else k-=sz;

        }
    }

}
```

7.5 Z-algo

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

8 Trees

8.1 Centroid

```
int sz;
void dfs(int node, int par){

    subtree[node]=1;
    sz++;
    for(auto u:adj[node]){
        if(u==par or vis[u]) continue;
        dfs(u,node);
        subtree[node]+=subtree[u];
    }

}

int centroid(int node,int par,int n){

    for(auto u:adj[node]){
        if(u==par) continue;
        if(!vis[u] and (subtree[u]*2)>n) return centroid(u,node,n);
    }
    return node;

}

void create_centroid_graph(int node,int par){

     sz=0;
    dfs(node,par);
    int c=centroid(node,0,sz);
    parent[c]=par;
    vis[c]=1;
    for(auto u:adj[c]){
        if(vis[u]) continue;

        create_centroid_graph(u,c);
    }

}
 create_centroid_graph(1,0);
```

8.2 dynamicMST

```
#include "bits/stdc++.h"
using namespace std;
#define all(x) begin(x),end(x)
typedef long long ll;
typedef vector<int> vi;

const int oo = 1e9;

struct DSU{
    vector<int> sz,parent;
    int components;
    void reset(int n) {
        fill(sz.begin(),sz.begin()+n,1);
        iota(parent.begin(),parent.begin()+n,0);
        components=n;
    }
    DSU(int n) : sz(n),parent(n) {
        reset(n);
    }
    void link(int a, int b) {
        components--;
        if(sz[a]<sz[b]) swap(a,b);
        sz[a]+=sz[b];
        parent[b] = a;
    }
    bool unite(int a, int b) {
        int pa = find(a), pb = find(b);
        if(pa!=pb) {
            link(pa,pb);
            return true;
        }
        return false;
    }
    int find(int a) {
        if(a==parent[a]) return a;
        return parent[a] = find(parent[a]);
    }
};

struct dynamicMST {
    struct edge {
        int l,r;
        int u,v,w;
        bool operator<(const edge& o) {
            return w<o.w;
        }
    };
    vector<edge> ives; // edges + time interval that they are active.
    vector<array<int,3>> startes;
    vi touch; // last time this edge was touched
    int totaln;
DSU dsu,dsu2;
vi id;
dynamicMST(vector<array<int,3>> ES, int n) : startes(ES), touch(ES.size()), totaln(n), dsu(n),dsu2(n), id(n) {
    // give all edges upfront.
}
int q=0;
void update(int i, int x) {
    // update edge weight of edge i to x
    // if you want to delete the edge, just set it to infinity
    q++;
    auto& [u,v,w] = startes[i];
    ives.push_back({touch[i],q,u,v,w});
    touch[i]=q;
    w = x;
}
vector<ll> ans;
void solve(int l, int r, vector<edge> es, int n, ll cost=0) {
    // remove edges that don't belong to this interval
    es.erase(stable_partition(all(es),[&](const edge& e) {return !(e.r<=l or r<=e.l);}),es.end());
    dsu.reset(n),dsu2.reset(n);

    // compressing connected components
    for(auto& e : es) if(l<e.l or e.r<r) { // active edges
        dsu.unite(e.u,e.v);
    }

    for(auto& e : es) if(e.l<=l and r<=e.r) { // fully overlapping edges
        if(dsu.unite(e.u,e.v)) {
            cost+=e.w;
            dsu2.unite(e.u,e.v);
        }
    }

    if(l+1==r) { // base case, we found the MST.
        ans[l]=cost;
        return;
    }

    int cnt=0; // relabel all connected components to 0...cnt-1
    for(int i=0;i<n;++i) if(dsu2.find(i)==i) id[i]=cnt++;
    dsu.reset(cnt);
        for(auto& e : es) { // relabeling and marking useless edges
            e.u = id[dsu2.find(e.u)], e.v = id[dsu2.find(e.v)];
            if(e.l<=l and r<=e.r) {
                if(!dsu.unite(e.u,e.v)) e.l=oo,e.r=-oo; // mark useless edge, will get deleted in next step
            }
        }
        int m = (l+r)/2;
        solve(l,m,es,cnt,cost);
        solve(m,r,es,cnt,cost);
    }
    vector<ll> run() {
        int m = startes.size();
        q++;
        for(int i=0;i<m;++i) {
            auto& [u,v,w] = startes[i];
            ives.push_back({touch[i],q,u,v,w});
        }

        sort(all(ives)); // (q+m) log(q+m) time
        ans.resize(q);
        solve(0,q,ives,totaln); // (q+m) log(q) alpha(n) time
        return ans;
    }
};

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n,m,q; cin >> n >> m >> q;
    vector<array<int,3>> es(m);

    for(auto& [u,v,w] : es) {
        cin >> u >> v >> w;
        --u,--v;
    }

    dynamicMST mst(es,n);

    for(int i=0;i<q;++i) {
        int k,d;
        cin >> k >> d, --k;
        mst.update(k,d);
    }
    auto ans = mst.run();
    for(int i=1;i<=q;++i) { // ans[0] gives the MST cost of the initial MST.
        cout << ans[i] << '\n';
    }

}
```

8.3 HLD

```
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 initofHLD(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);
}
vector<int> tree(400001), arr(100001);
int arrtotree[300001];
```

```
void init(int n)
{
    for (int i = 0; i < n; i++)
        tree[n + i] = arr[i];

    for (int i = n - 1; i >= 1; i--)
        tree[i] = max(tree[2 * i], tree[2 * i + 1]);
}

void update(int pos, int value, int n)
{
    pos += n;
    tree[pos] = value;

    while (pos > 1) {
        pos >>= 1;
        tree[pos] = max(tree[2 * pos], tree[2 * pos + 1]);
    }
}

int segquery(int left, int right, int n)
{
    left += n;
    right += n + 1; // inclusive range
    int mi = (int)-1;

    while (left < right) {
        if (left & 1) {
            mi = max(mi, tree[left]);
            left++;
        }
        if (right & 1) {
            right--;
            mi = max(mi, tree[right]);
        }
        left >>= 1;
        right >>= 1;
    }
    return mi;
}

int query(int a, int b, int n) {
    int res = 0;
    while (head[a] != head[b]) {
        if (depth[head[a]] > depth[head[b]]) swap(a, b);
        res = max(res, segquery(pos[head[b]], pos[b], n));
        b = parent[head[b]];
    }
    if (depth[a] > depth[b]) swap(a, b);
    res = max(res, segquery(pos[a], pos[b], n));
    return res;
}

void do_the_honour() {
    int n, q;
    cin >> n >> q;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) cin >> a[i];

    vector<vector<int>> adj(n);
    for (int i = 1; i < n; ++i) {
        int x, y;
        cin >> x >> y;
        --x; --y;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    initofHLD(adj);

    arr.resize(n);
    tree.resize(4 * n);
    for (int i = 0; i < n; ++i) arr[pos[i]] = a[i];
    init(n);

    while (q--) {
        int type;
        cin >> type;
        if (type == 2) {
            int l, r;
            cin >> l >> r;
            cout << query(l - 1, r - 1, n) << ' ';
        } else {
            int s, x;
            cin >> s >> x;
            s--;
            update(pos[s], x, n);
        }
    }
    cout << '\n';
}
```

8.4 NumOfPathsPassingThroughNodei

```
int dp[32][200001];
int level[200001];
vector<int>adj[200001];

void dfs(int node,int par){
    for(auto u:adj[node]){
        if(u==par) continue;
        level[u]=level[node]+1;
        dp[0][u]=node;
        dfs(u,node);
    }


}

int c[200001];


void dfs22(int node,int par){

    for(auto u:adj[node]){
        if(u==par) continue;
        dfs22(u,node);
        c[node]+=c[u];
    }

}

void do_the_honour(){

    int n,m;cin >> n >> m ;

    for(int i=2;i<=n;i++){
        int x,y;cin >> x >> y;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    level[1]=0;
 dfs(1,0);
    dp[0][1]=0;


    for(int i=1;i<=24;i++){
        for(int j=1;j<=n;j++) dp[i][j]=dp[i-1][dp[i-1][j]];
    }

    for(int i=0;i<m;i++){
        int a,b;cin >> a >> b;
        int x=a,y=b;

        if(level[a]<level[b]) swap(a,b),swap(x,y);
        int diff=level[a]-level[b];

        for(int i=0;i<=24;i++){
            if(diff&(1<<i)){
                a=dp[i][a];
            }
        }
        int node=-1;
        if(a==b){
            node=a;
        }else{

            for(int ii=24;ii>=0;ii--){
                if(dp[ii][a]!=dp[ii][b]){
                    a=dp[ii][a];
                    b=dp[ii][b];
                }
            }
            node=dp[0][a];

        }


        c[x]++,c[y]++,c[node]--;
        if(dp[0][node]>0) c[dp[0][node]]--;


    }

    dfs22(1,0);
    for(int i=1;i<=n;i++) cout << c[i] << " ";
}
```

8.5 smallTolarge

```
//finding sum of most frequent values in the subtree of each node;small to large marging
vector<int>adj[200001];
map<int,int>s[200001];
int ans[200001];
int par[200001];
int max_freq_cnt[200001];
int max_freq[200001];
int merge(int a,int b){
    if(s[a].size()<s[b].size()) swap(a,b);
    for(auto u:s[b]) {
        int color=u.first;
        int c=u.second;
        int new_cnt=c+s[a][color];
        s[a][color]+=c;

        if(new_cnt>max_freq[a]){
            max_freq[a]=new_cnt;
            max_freq_cnt[a]=color;
        }
        else if(new_cnt==max_freq[a]) max_freq_cnt[a]+=color;
    }

    return a;

}

int dfs(int node,int p){

    int i=par[node];

    for(auto u:adj[node]){
        if(p==u) continue;
        i=merge(i,dfs(u,node));
    }



    ans[node]=max_freq_cnt[i];
    return i;
}


void do_the_honour(){

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

    for(int i=1;i<=n;i++){
        s[i][color[i]]++;
        par[i]=i;
        max_freq[i]=1;
        max_freq_cnt[i]+=color[i];
    }

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

    dfs(1,0);
    for(int i=1;i<=n;i++) cout << ans[i] << " ";


}
```

8.6 Tree Flattening

```
void dfs(int node,int par){

    timer++;
    fa[timer]=node;
    for(auto u:adj[node]){
        if(u==par) continue;
        dfs(u,node);

    }
    timer++;
    fa[timer]=node;

}
```