Team notebook

Aust_Simplex1ty

January 2, 2025

Contents

1 Data Structures

1.1 DSU

```
int parent[N]; //map<int,int> parent;
int siz[N];   //map<int,int> siz;
void make_set(int v) {
    parent[v]=v; siz[v]=1;
}
int find_set(int v) {
    return (v==parent[v])?v:parent[v]=find_set(parent[v]);
}
void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a == b) return;
    if(siz[a]<siz[b]) swap(a,b);
    parent[b] = a;
    siz[a]+=siz[b];
    siz[b]=0; //siz.erase(b);
}
int get_size(int v){
    return siz[find_set(v)];
}
```

1.2 LazySegTree

```
class SegmentTree
{
public:
    int n;
    vector<int>a,lazy,tree;
    SegmentTree(vector<int>arr){
        a=arr;
        n = arr.size();
        lazy.assign(4*n,0);
        tree.assign(4*n,0);
        build(1,0,n-1);
    }
    void update(int l,int r,int val){update(1,0,n-1,l,r,val);}
    int query(int l,int r){return query(1,0,n-1,l,r);}
    void print(){for(auto it : tree)cout << it << endl;}
private:
    void propagate(int node,int start,int end){
        if(start==end)tree[node] += lazy[node];
        else{
            tree[node] += (end-start+1)*lazy[node]; /*if you need sum use it
                else for other
                                                    things remove (end-start+1)
                                                        part*/
            lazy[2*node] += lazy[node];
            lazy[2*node+1] += lazy[node];
        }
        lazy[node]=0;
    }
    void build(int node,int start,int end){
        if(start==end)tree[node] = a[start];
        else {
            int mid = (start+end)/2;
            build(2*node,start,mid);
            build(2*node+1,mid+1,end);
            tree[node] = tree[2*node] + tree[2*node+1];
        }
    }
    void update(int node,int start,int end,int l,int r,int val){
        propagate(node,start,end);
        if(end<l or start>r)return;
        if(start==end)tree[node]+=val;
        else if(l<=start and end<=r){
            lazy[node] += val;
            propagate(node,start,end);
        }
        else {
            int mid = (start+end)/2;
            update(2*node,start,mid,l,r,val);
```

```
            update(2*node+1,mid+1,end,l,r,val);
            tree[node] = tree[2*node] + tree[2*node+1];
        }
    }
    int query(int node,int start,int end,int l,int r){
        if(end<l or start>r)return 0;
        propagate(node,start,end);
        if(start==end){
            return tree[node];
        }
        else if(l<=start and end<=r)return tree[node];
        else {
            int mid = (start+end)/2;
            int left = query(2*node,start,mid,l,r);
            int right = query(2*node+1,mid+1,end,l,r);
            return left+right;
        }
    }
};
```

1.3 Mo'sAlgo

```
int Mx = 100005;
struct Query
{
    int l,r,ind;
};
bool cmp(Query &a,Query &b){
    int blocks_size = sqrt(Mx);
    if(a.l/blocks_size!=b.l/blocks_size)
        return a.l/blocks_size < b.l/blocks_size;
    return a.r < b.r;
}
void add(int ind,int &d){
}
void remove(int ind,int &d){
}
vector<Query>qu(q);
for(int i=0;i<q;i++){
    int l,r; cin >> l >> r;
    qu[i] = {l-1,r-1,i};
}
sort(all(qu),cmp);
vector<int>result(q);
int curr_l = 0 , curr_r = -1 , d = 0;
for(int i=0;i<q;i++){
    int l = qu[i].l;
    int r = qu[i].r;
    while(curr_r<r)add(v[++curr_r],d);
    while(curr_r>r)remove(v[curr_r--],d);
    while(curr_l<l)remove(v[curr_l++],d);
    while(curr_l>l)add(v[--curr_l],d);
    result[qu[i].ind] = d;
}
```

1.4 PBDS

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;//sorted ascending
typedef tree<int, null_type, greater<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_rset;//sorted descending
typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_multiset;//sorted ascending
typedef tree<int, null_type, greater_equal<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_rmultiset;//sorted descending
/*
***Don't use long long as defined when using it***
1>declare ==> ordered_set A
2>insert ==> A.insert(val);
3>finding kth element (0 based - index) ==> *A.find_by_order(k);
4>finding the index in which val is located
  if not found returns its relative position(position if inserted) ==>
      A.order_of_key(val);
5>lower Bound ==> *A.lower_bound(val);
6>Upper Bound ==> *A.upper_bound(val);
7>Erase ==> A.erase(val or pointer);
  For multiset use custom delete function!
*/
void myErase(ordered_set &t, int v){
    int rank = t.order_of_key(v);
    ordered_set::iterator it = t.find_by_order(rank);
```

```
    if(*it == v)t.erase(it);
}
```

1.5 SegmentTree

```
class SegmentTree
{
public:
    int n;
    vector<int>tree,arr;
    SegmentTree(vector<int>&v){
        n = v.size();
        arr = v;
        tree.assign(4*n,0);
        build(1,0,n-1);
    }
    void update(int ind,int val){update(1,0,n-1,ind,val);}
    int query(int l,int r){return query(1,0,n-1,l,r);}
private:
    int merge(int a,int b){
        return (a+b);
    }
    void build(int node,int start,int end){
        if(start==end)tree[node] = arr[start];
        else {
            int mid = (start+end)/2;
            build(2*node,start,mid);
            build(2*node+1,mid+1,end);
            tree[node] = merge(tree[2*node],tree[2*node+1]);
        }
    }
    void update(int node,int start,int end,int ind,int val){
        if(ind<start or ind>end)return;
        if(start==end)tree[node] = val;
        else {
            int mid = (start+end)/2;
            if(start<=ind and ind<=mid)update(2*node,start,mid,ind,val);
            else update(2*node+1,mid+1,end,ind,val);
            tree[node] = merge(tree[2*node],tree[2*node+1]);
        }
    }
    int query(int node,int start,int end,int l,int r){
        if(end<l or start>r)return 0;
        if(start==end)return tree[node];
        else if(l<=start and end<=r)return tree[node];
        else{
            int mid = (start+end)/2;
            int left = query(2*node,start,mid,l,r);
            int right = query(2*node+1,mid+1,end,l,r);
            return merge(left,right);
        }
    }
};
```

2 dp

2.1 CHT

```
const int is_query = -(1LL<<62);
struct line {
    int m, b;
    mutable function<const line*()> succ;
    bool operator<(const line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const line* s = succ();
        if (!s) return 0;
        int x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct dynamic_hull : public multiset<line> { // wiint maintain upper huint
     for maximum
    const int inf = LLONG_MAX;
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
```

```
        /* compare two lines by slope, make sure denominator is not 0 */
        int v1 = (x->b - y->b);
        if (y->m == x->m) v1 = x->b > y->b ? inf : -inf;
        else v1 /= (y->m - x->m);
        int v2 = (y->b - z->b);
        if (z->m == y->m) v2 = y->b > z->b ? inf : -inf;
        else v2 /= (z->m - y->m);
        return v1 >= v2;
    }
    void insert_line(int m, int b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    int eval(int x) {
        auto l = *lower_bound((line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

2.2 DigitDp

```
pair<int,string> fun(int pos,bool upper,bool lower,bool started){
    if(pos == r.size()){
        return {1,""};
    }
    auto &ret = dp[pos][upper][lower][started];
    if(ret.ff != -1)return ret;
    for(int i=0;i<=9;i++){
        int lb = l[pos]-'0' , up = r[pos]-'0';
        if(!lower and i > up)break;
        if(!upper and i < lb)continue;
        bool is_upper = upper , is_lower = lower , is_started = started || i!=0;
        if(i > lb)is_upper = 1;
        if(i < up)is_lower = 1;
        auto it = fun(pos+1,is_upper,is_lower,is_started);
        if(is_started){
            it.ff *= i;
            it.ss = to_string(i) + it.ss;
        }
        if(it.ff > ret.ff){
            ret = it;
        }
    }
    return ret;
}
void solve(int tc){
    //make sure l.size() == r.size()
    cout << fun(0,0,0,0).ss << endl;
}
int fun(int pos,int cnt,bool under,bool started){
    if(pos == num.size()){
        if(!started)return 0;
        if(cnt < 4)return 1;
        else return 0;
    }
    int &ret = dp[pos][cnt][under][started];
    if(ret != -1)return ret;
    ret = 0;
    for(int i=0;i<=9;i++){
        int digit = num[pos] - '0';
        if(!under and i > digit)break;
        int is_under = under;
        if(i < digit)is_under = 1;
        int is_started = started || i != 0;
        ret += fun(pos + 1 , cnt + (i!=0) , is_under , is_started);
    }
    return ret;
}
void solve(int tc){
    int l,r; cin >> l >> r;
    l--;
    num = to_string(l);
    int lo = fun(0,0,0,0) + 1;
    num = to_string(r);
    int up = fun(0,0,0,0) + 1;
    cout << up-lo << endl;
}
```

2.3 DNC

```
void dnc(int k,int l,int r,int optl,int optr){
```

```
    if(l > r) return;
    int mid = (l+r) >> 1;
    int optm = optl;
    for(int i=optl;i<=min(mid,optr);i++){
        int ret = (i?dp[k-1][i-1]:0) + cost(i,mid-1,a,b,x[mid]);
        if(ret < dp[k][mid]){
            dp[k][mid] = ret;
            optm = i;
        }
    }
    dnc(k,l,mid-1,optl,optm);
    dnc(k,mid+1,r,optm,optr);
}
void solve(int tc){
    //init base case carefully
    dp[0][0] = 0;
    for(int i=1;i<=k;i++)dnc(i,1,n,1,n);
    cout << dp[k][n] << endl;
}
```

2.4 edit distance

```
string s,t; cin >> s >> t;
    int n = s.size() , m = t.size();
    vector<vector<int>>dp(n+1,vector<int>(m+1,INF));
    dp[0][0] = 0;
    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(i)dp[i][j] = min(dp[i][j],dp[i-1][j]+1);
            if(j)dp[i][j] = min(dp[i][j],dp[i][j-1]+1);
            if(i and j){
                dp[i][j] = min(dp[i][j],dp[i-1][j-1]+(s[i-1]!=t[j-1]));
            }

        }
    }
    cout << dp[n][m] << endl;
```

2.5 Tile dp

```
int dp[1001][(1<<11)];
int n,m;
void generator(int c,int nx,int i,vector<int>& masks){
    if(i==n+1){
        masks.push_back(nx);
        return;
    }
    if((c&(1<<i))!=0)generator(c,nx,i+1,masks);
    if(i!=n and (c&(1<<i))==0 and (c&(1<<(i+1)))==0)generator(c,nx,i+2,masks);
    if((c&(1<<i))==0)generator(c,nx+(1<<i),i+1,masks);
}
int fun(int i,int mask){
    if(i==m+1){
        if(mask==0)return 1;
        return 0;
    }
    if(dp[i][mask]!=-1)return dp[i][mask];
    vector<int>_allMasks;
    generator(mask,0,1,_allMasks);
    int ans = 0;
    for(auto it : _allMasks){
        ans = (ans + fun(i+1,it))%MOD;
    }
    return dp[i][mask] = ans;
}
```

3 Geometry

3.1 Formula 2d3d

```
const double PI=3.141592653589793;//acos(-1)
struct TwoDGeometry{
    static double triangleArea(double base,double height){
        return 0.5*base*height;
    }
    static double rectanglePerimeter(double length,double breadth){
        return 2*(length+breadth);
    }
    static double rectangleArea(double length,double breadth){
        return length*breadth;
```

```
    }
    static double rectangleDiagonal(double length,double breadth){
        return sqrt(length*length+breadth*breadth);
    }
    static double squarePerimeter(double side){
        return 4*side;
    }
    static double squareArea(double side){
        return side*side;
    }
    static double squareDiagonal(double side){
        return sqrt(2)*side;
    }
    static double parallelogramArea(double base,double height){
        return base*height;
    }
    static double rhombusArea(double diagonal1,double diagonal2){
        return 0.5*diagonal1*diagonal2;
    }
    static double trapeziumArea(double height,double parallelSide1,double
         parallelSide2){
        return 0.5*height*(parallelSide1+parallelSide2);
    }
    static double circlePerimeter(double radius){
        return 2*PI*radius;
    }
    static double circleArea(double radius){
        return PI*radius*radius;
    }
};

// 3D Shape Formulas
struct ThreeDGeometry{
    static double cubeSurfaceArea(double side){
        return 6*side*side;
    }
    static double cubeVolume(double side){
        return pow(side,3);
    }
    static double cuboidSurfaceArea(double length,double breadth,double
         height){
        return 2*(length*breadth+breadth*height+height*length);
    }
    static double cuboidVolume(double length,double breadth,double height){
        return length*breadth*height;
    }
    static double cylinderSurfaceArea(double radius,double height){
        return 2*PI*radius*(radius+height);
    }
    static double cylinderVolume(double radius,double height){
        return PI*radius*radius*height;
    }
    static double coneSurfaceArea(double radius,double height){
        double slantHeight=sqrt(radius*radius+height*height);
        return PI*radius*(radius+slantHeight);
    }
    static double coneVolume(double radius,double height){
        return (1.0/3)*PI*radius*radius*height;
    }
    static double sphereSurfaceArea(double radius){
        return 4*PI*radius*radius;
    }
    static double sphereVolume(double radius){
        return (4.0/3)*PI*pow(radius,3);
    }
    static double hemisphereSurfaceArea(double radius){
        return 3*PI*radius*radius;
    }
    static double hemisphereVolume(double radius){
        return (2.0/3)*PI*pow(radius,3);
    }
};
//ThreeDGeometry::sphereVolume(3)
```

4 graph

4.1 Articulation Point

```
//------------------------------------------------//
const int Mx_N = 1e4 + 10;
int Time,cnt;
vector<int>graph[Mx_N],disc(Mx_N),low(Mx_N),ap(Mx_N);
int dfsAp(int vertex,int parent){
       int children = 0;
       low[vertex] = disc[vertex] = ++Time;
```

```
	for(auto child : graph[vertex]){
		if(child == parent)continue;
		if(!disc[child]){
			children++;
			dfsAp(child,vertex);
			if(disc[vertex] <= low[child])ap[vertex]=1;
			low[vertex] = min(low[vertex],low[child]);
		}
		else low[vertex] = min(low[vertex],disc[child]);
	}
	return children;
}
void AP(int n){
	Time = 0;
	for(int i=1;i<=n;i++)
		if(!disc[i])
			ap[i]=dfsAp(i,i)>1;
	cnt=0;
	for(int i=1;i<=n;i++)if(ap[i])cnt++;
}
void clear(int n){
	for(int i=1;i<=n;i++){
		graph[i].clear();
		disc[i]=low[i]=ap[i]=0;
	}
}
//------------------------------------------------------//
```

4.2 bellman ford

```
for (int i = 0; i < n - 1; ++i)
	for (Edge e : edges)
	    if (d[e.a] < INF)
		d[e.b] = min(d[e.b], d[e.a] + e.cost);
```

4.3 Bridges In Graph

```
//------------------------------------------------//
const int Mx_N = 1e4 + 10;
int Time,cnt;
vector<int>graph[Mx_N],disc(Mx_N),low(Mx_N);
vector<pii>bridges;
void dfsBr(int vertex,int parent){
	low[vertex] = disc[vertex] = ++Time;
	for(auto child : graph[vertex]){
		if(child == parent)continue;
		if(!disc[child]){
			dfsBr(child,vertex);
			if(disc[vertex] < low[child]){
				bridges.pb({min(vertex,child),max(vertex,child)});
			}
			low[vertex] = min(low[vertex],low[child]);
		}
		else low[vertex] = min(low[vertex],disc[child]);
	}
}
void Br(int n){
	Time = 0;
	for(int i=1;i<=n;i++)
		if(!disc[i])
			dfsBr(i,i);
	cnt = bridges.size();
}
void clear(int n){
	for(int i=1;i<=n;i++){
		graph[i].clear();
		disc[i]=low[i]=0;
		bridges.clear();
	}
}
//------------------------------------------------------//
```

4.4 Find Cycle

```
bool dfs(int s,int par){
   vis[s] = 1;
   for(auto u : graph[s]){
      if(u==par)continue;
      if(vis[u]){
         st = u;
         ed = s;
         return true;
```

```
      }
      parent[u] = s;
      if(dfs(u,s))return true;
   }
   return false;
}
void solve(int tc){
   int n,e;
   cin >> n >> e;
   parent.assign(n+1,-1);
   vis.assign(n+1,false);
   st = -1;
   while(e--){
      int u,v;
      cin >> u >> v;
      graph[u].pb(v);
      graph[v].pb(u);
   }
   for(int i=1;i<=n;i++){
      if(!vis[i] && dfs(i,parent[i]))break;
   }
   if(st == -1) cout << "IMPOSSIBLE" << nline;
   else{
      vector<int>cycle;
      int v = ed
      cycle.pb(st);
      while(v!=st){
         cycle.pb(v);
         v = parent[v];
      }
      cycle.pb(st);
}
```

4.5 Max Flow

```
struct Edge{
    int index;
    int src, dest;
    ll val;
    int residualIndex;
};
struct Flow{
int n;
int src, dest;
int iteration = 0;
vector<Edge> edgesT;
vector<vector<int>> edges;
vector<int> visited;
bool solved;
ll flow;
Flow(vector<pair<int, ll>>* edges1, int n1, int s, int d){
    n = n1, src = s, dest = d;
    solved = false;
    flow = 0, iteration = 1;
    visited.resize(n);
    fill(all(visited), 0);
    edges.resize(n);
    for(int i = 0; i < n; i++){
        for(auto j : edges1[i]){
            Edge e1 = {sz(edgesT), i, j.ff, j.ss, sz(edgesT) + 1};
            Edge e2 = {sz(edgesT) + 1, j.ff, i, 0, sz(edgesT)};
            edgesT.pb(e1);
            edgesT.pb(e2);
            edges[i].pb(e1.index);
            edges[j.ff].pb(e2.index);
        }
    }
}
ll bfs(int root){
    queue<int> qu;
    qu.push(root);
    visited[root] = iteration;
    vector<int> prev(n, -1);
    while(!qu.empty()){
        int node = qu.front();
        qu.pop();
        if(node == dest)
            break;
        for(auto i : edges[node]){
            Edge e1 = edgesT[i];
            if(visited[e1.dest] != iteration && e1.val > 0){
                visited[e1.dest] = iteration;
                prev[e1.dest] = e1.index;
                qu.push(e1.dest);
            }
        }
```

```
        }
        int currNode = dest;
        if(prev[currNode] == -1)
            return 0;
        ll finalValue = INF;
        while(prev[currNode] != -1){
            Edge e1 = edgesT[prev[currNode]];
            finalValue = min(finalValue, e1.val);
            currNode = e1.src;
        }
        currNode = dest;
        while(prev[currNode] != -1){
            Edge e1 = edgesT[prev[currNode]];
            e1.val -= finalValue;
            edgesT[e1.index] = e1;
            edgesT[e1.residualIndex].val += finalValue;
            currNode = e1.src;
        }
        return finalValue;
    }
    void EdmondsKarp(){
        while(true){
            ll f = bfs(src);
            if(f == 0)
                return;
            flow += f;
            iteration++;
        }
    }
    ll maxFlow(){
        if(!solved){
            solved = true;
            EdmondsKarp();
        }
        return flow;
    }
};
```

4.6 Maximum Bipartite Algorithm(HopkroftkarpAlgorithm)

```
struct HopcroftKarp { //O(Sqrt(V) * E)
  static const int inf = 1e9;
  int n;
  vector<int> l, r, d;
  vector<vector<int>> g;
  HopcroftKarp(int _n, int _m) {
    n = _n;
    int p = _n + _m + 1;
    g.resize(p);
    l.resize(p, 0);
    r.resize(p, 0);
    d.resize(p, 0);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v + n); //right id is increased by n, so is l[u]
  }
  bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
      if (!l[u]) d[u] = 0, q.push(u);
      else d[u] = inf;
    }
    d[0] = inf;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto v : g[u]) {
        if (d[r[v]] == inf) {
          d[r[v]] = d[u] + 1;
          q.push(r[v]);
        }
      }
    }
    return d[0] != inf;
  }
  bool dfs(int u) {
    if (!u) return true;
    for (auto v : g[u]) {
      if(d[r[v]] == d[u] + 1 && dfs(r[v])) {
        l[u] = v;
        r[v] = u;
        return true;
      }
    }
    d[u] = inf;
    return false;
```

```
  }
  int maximum_matching() {
    int ans = 0;
    while (bfs()) {
      for(int u = 1; u <= n; u++) if (!l[u] && dfs(u)) ans++;
    }
    return ans;
  }
};//1 Based kintu
```

4.7 SCC

```
void dfs(int v,vector<int>&vis,vector<int>adj[],stack<int>&st){
    vis[v]=1;
    for(auto it : adj[v]){
        if(!vis[it]){
            dfs(it,vis,adj,st);
        }
    }
    st.push(v);
}
vector<vector<int>>scc_list;
vector<int>all_scc;
void dfs(int v,vector<int>&vis,vector<int>rev_adj[]){
    vis[v]=1;
    all_scc.pb(v);
    for(auto it : rev_adj[v]){
        if(!vis[it])
            dfs(it,vis,rev_adj);
    }
}
int Kosaraju(int V,vector<int>adj[]){ // 1 based
   vector<int>vis(V+1,0);
   stack<int>st;
   for(int i=1;i<=V;i++){
        if(!vis[i])dfs(i,vis,adj,st);
   }
   vector<int>rev_adj[V+1];
   for(int i=1;i<=V;i++){
        vis[i]=0;
        for(auto it : adj[i]){
            rev_adj[it].pb(i);
        }
   }
   int scc=0;
   while(!st.empty()){
        int v = st.top();
        st.pop();
        if(!vis[v]){
            scc++;
            dfs(v,vis,rev_adj);
            scc_list.pb(all_scc);
            all_scc.clear();
        }
   }
   return scc;
}
```

5 Misc

5.1 2DPrefixSum

```
void build(){
       for(int i=1;i<=n;i++)
       for(int j=1;j<=m;j++)
              pref[i][j] = pref[i-1][j] + pref[i][j-1]
                                - pref[i-1][j-1] + a[i][j];
}
int query(int x1,y1,x2,y2){
       return pref[x2][y2]-pref[x1-1][y2]
                -pref[x2][y1-1]+pref[x1-1][y1-1];
}
void update(int a1,b1,a2,b2){
       pref[a1][b1]++;pref[a2+1][b2+1]++;
       pref[a1][b2+1]--;pref[a2][b1+1]--;
}
```

5.2 co-ordinate compression

```
vector<int>indices;
int getCompressedIndex(int a) {
```

```
      return lower_bound(indices.begin(), indices.end(), a) - indices.begin();
}
//========== COORDINATE COMPRESSION =======
sort(indices.begin(), indices.end());
indices.erase(unique(indices.begin(), indices.end()), indices.end());
```

5.3 Date

```
ll date_to_int(ll d, ll m, ll y) {
   ll u = 1461 * (y + 4800 + (m- 14) / 12) / 4;
   ll v = 367 * (m- 2- (m- 14) / 12 * 12) / 12;
   ll w = 3 * ((y + 4900 + (m- 14) / 12) / 100) /4;
   return u + v- w + d- 32075;
}
tuple<ll, ll, ll> int_to_date(ll u) {
   ll x, n, i, j, d, m, y;
   x = u + 68569;
   n = 4 * x / 146097;
   x-= (146097 * n + 3) / 4;
   i = (4000 * (x + 1)) / 1461001;
   x-= 1461 * i / 4- 31;
   j = 80 * x / 2447;
   d = x- 2447 * j / 80;
   x = j / 11;
   m = j + 2- 12 * x;
   y = 100 * (n- 49) + i + x;
   return make_tuple(d, m, y);
}
```

5.4 fop

```
ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
freopen("input.txt","r",stdin);
freopen("output.txt","w",stdout);
```

5.5 kadane

```
int kadane(int st,int ed,vector<int>&v){
   int max_so_far = -1e9,max_yet=0;
   for(int i=st;i<=ed;i++){
      max_yet += v[i];
      if(max_yet > max_so_far){max_so_far = max_yet;
      if(max_yet < 0) max_yet = 0;
   }
   return max_so_far;
}
```

5.6 Matrix Exponentiation

```
const int nmax = 4;
long long int mod = 1e9+7;
struct Matrix{
    /// after constructing val contains garbage
    long long int val[nmax][nmax];
    int row, col;

    Matrix(int _r, int _c){
        row = _r;
        col = _c;

        /* memset */
        for(int i = 0; i<row; i++)
            for(int j = 0; j<col; j++)
                val[i][j] = 0;
    }

    Matrix operator*(Matrix other){
        Matrix result(row, other.col); /// O(nmax*nmax)

        for(int i = 0; i<row; i++){
            for(int j = 0; j<other.col; j++){
                for(int k = 0; k<col; k++){
                    result.val[i][j] += val[i][k] * other.val[k][j];
                    result.val[i][j] %= mod;
                }
            }
        }
        /// O(row * col * other.col)
```

```
        return result; /// O(nmax*nmax)
    }

    void print(){
        for(int i = 0; i<row; i++){
            for(int j = 0; j<col; j++){
                cout<<val[i][j]<<" ";
            }
            cout<<endl;
        }
    }
};

/// X^n
Matrix Matexpo(Matrix X, long long int n){
    Matrix Y(X.row, X.col);

    //cout<<n<<endl;

    if(n == 0){
        for(int i = 0; i<X.row; i++)
            Y.val[i][i] = 1;

        return Y;
    }
    Y = Matexpo(X, n/2);
    Y = Y * Y;

    if(n % 2 == 1){
        Y = Y * X;
    }

    return Y;
}
/*
    ->adjust the matrix size as per requirement as it can cause time limit
           X2 is enough for 1e9
    ->
    Matrix F(2, 1);

    F.val[0][0] = a; // change the initial value
    F.val[1][0] = b;

    Matrix M(2, 2);
    M.val[0][0] = 0, M.val[0][1] = 1;//co-efficients
    M.val[1][0] = 1, M.val[1][1] = 1;

    M = Matexpo(M,n); //it will be n-j if base case starts from jth term

    F = M * F;

    cout<< F.val[0][0]<<endl;

    O(log n)
*/
```

5.7 ternary search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;//set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);//evaluates the function at m1
        double f2 = f(m2);//evaluates the function at m2
        if (f1 < f2)l = m1;
        else r = m2;
    }
    return f(l);//return the maximum of f(x) in [l, r]
}
```

6 Number Theory

6.1 all gcd

```
pair<int,int>extended_gcd(int a,int b){
    if(b==0) return {1,1};
    auto [x1,y1] = extended_gcd(b,a%b);
    int x = y1 , y=x1-(a/b)*y1;
    return {x,y};
}
int gcd(int a,int b){
    if(b==0)return a;
```

```
    return gcd(b,a%b);
}
```

6.2 binpow

```
int binpow(int a,int b){
    a%=MOD;
    int res=1;
    while(b>0){
        if(b&1)res = res*a%MOD;
        a=a*a%MOD;
        b>>=1;
    }
    return res;
}
```

6.3 Countofdivisors

```
int count_of_divisors(int n){
        int ans = 1;
        for(int i=2;i*i<=n;i++){
                if(n%i==0){
                        int e =0;
                        while(n%i==0){
                                e++;
                                n/=i;
                        }
                        ans *= (e+1);
                }
        }
        if(n > 1)ans *= 2;
        return ans;
}
```

6.4 CSOD

```
int CSOD(int n){
        int i=1;
        int ans = 0;
        while(i<=n){
                int q = n/i;
                int j = (n/q) + 1;
                int s = sum(i,j-1);//i + (i+1) + ...(j-1)
                ans += s*q;
        }
        return ans;
}
```

6.5 Mobius

```
mobb[1]=1;
for(int i=1;i<m;i++){
    for(int j=i;j<m;j+=i){
        if(j!=i)mobb[j] -= mobb[i];
        if(i>1)divs[j].pb(i);
    }
}
```

6.6 ncr

```
const int N = 1e6 + 9, mod = 1e9 + 7;

int f[N], inv[N], finv[N];
void prec() {
  f[0] = 1;
  for (int i = 1; i < N; i++) f[i] = 1LL * i * f[i - 1] % mod;
  inv[1] = 1;
  for (int i = 2; i < N; i++ ) {
    inv[i] = (-(1LL * mod / i) * inv[mod % i] ) % mod;
    inv[i] = (inv[i] + mod) % mod;
  }
  finv[0] = 1;
  for (int i = 1; i < N; i++) finv[i] = 1LL * inv[i] * finv[i - 1] % mod;
}
int ncr(int n, int r) {
  if (n < r || n < 0 || r < 0) return 0;
  return 1LL * f[n] * finv[n - r] % mod * finv[r] % mod;
```

```
}

void brute() {
 for (int i = 0; i < N; i++) {
   C[i][0] = 1;
 }
 for (int i = 1; i < N; i++) {
   for (int j = 1; j <= i; j++) {
     C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
   }
 }
}
```

6.7 PrimeFactofN!

```
void Nfact_PF(int n){
    //Sieve till n and generate all primes <= n
    for(auto it : primes){
        int c = it , cnt = 0;
        if(c > n)break;
        while(c<=n){
            cnt += n / c;
            c *= it;
        }
        cout << it << " " << cnt << endl;
    }
}
```

6.8 PrimeFactorization(logn)

```
vector<int>spf(N+1,1);
void sieve(){
    spf[0] = 0;
    for(int i=2;i<=N;i++){
        if(spf[i]==1){
            for(int j=i;j<=N;j+=i){
                if(spf[j]==1)
                    spf[j]=i;
            }
        }
    }
}
void getFactorization(int x,vector<int>&gg){
    while(x!=1){
        gg.pb(spf[x]);
        x = x/spf[x];
    }
}
```

6.9 Segmented Sieve

```
vector<bool>chk(N+1,1);
vector<int>primes;
void sieve(){
    for(int i=3;i<=N;i+=2){
        if(chk[i]){
            for(int j=i*i;j<=N;j+=i+i)
                chk[j]=0;
        }
    }
    primes.pb(2);
    for(int i=3;i<=N;i+=2)if(chk[i])primes.pb(i);
}
int SegmentedSieve(int l,int r){
    if(l<3)l=2;
    vector<bool>is_prime(r-l+1,1);
    for(auto p : primes){
        int spm = (l/p) * p;
        if(spm < l)spm += p;
        if(spm==p)spm += p;
        for(int i=spm;i<=r;i+=p){
            is_prime[i-l] = 0;
        }
    }
    int cnt = 0;
    for(int i=l;i<=r;i++){
        if(is_prime[i-l]==1){
            cnt++;
        }
    }
    return cnt;
}
```

6.10 sieve

```
const int Max_N = 1e6 + 10;
vector<int>primes , chk(Max_N,1);
void sieve(){
    for(int i=3;i<Max_N;i+=2){
        if(chk[i]){
                for(int j=i*i;j<Max_N;j+=i+i){
                chk[j] = 0;
            }
        }
    }
    chk[2]=1;
    primes.pb(2);
    for(int i=3;i<Max_N;i+=2){
        if(chk[i])primes.pb(i);
    }
}
/*
        to check prime
                if(n%2==0)return n==2;
                else if(n&1 and chk[i])return 1;
                else return 0;
*/
```

6.11 SumOfDivisors

```
int sum_of_divisor(int n){
    int ans = 1;
    for(int i=2;i*i<=n;i++){
        if(n%i==0){
            int e = 0 , gg=1,sum = 1;
            while(n%i==0){
                n/=i,e++;
                gg *= i;
                sum += gg;
            }
            ans *= sum;
        }
    }
    if(n > 1)
        ans *= (1+n);
    return ans;
}
```

7 StressTesting

7.1 gen

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

inline ll gen_random(ll l, ll r) {
    return uniform_int_distribution<ll>(l, r)(rng);
}


// // Random Real Number Generator:

// mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

// inline double gen_random(double l, double r) {
//     return uniform_real_distribution<double>(l, r)(rng);
// }
int main(){
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        int n = gen_random(1,100);
        cout << n << endl;
}
```

7.2 stress

```
@echo off
```

```
if [%1]==[] (set /A numLoop = 100) else (set /A numLoop = %1)
if [%2]==[] (set /A doComp = 1) else (set /A doComp = %2)

if %doComp% equ 1 (
    echo Compiling solution, gen, brute...

    g++ -std=c++17 gen.cpp -o gen
    g++ -std=c++17 solution.cpp -o solution
    g++ -std=c++17 brute.cpp -o brute

    echo Done compiling.
)

set "diff_found="

for /l %%x in (1, 1, %numLoop%) do (
    echo %%x
    gen > input.in
    solution < input.in > output.out
    brute < input.in > output2.out

    rem add \f after "fc" to ignore trailing whitespaces and to convert
    rem multiple whitespaces into one space
    fc output.out output2.out > diagnostics
    if errorlevel 1 (
        set "diff_found=y"
        goto :break
    )
)

:break

if defined diff_found (
    echo A difference has been found.
    echo Input:
    type input.in
    echo.
    echo.

    echo Output:
    type output.out
    echo.

    echo Expected:
    type output2.out
    echo.
) else (
    echo All tests passed :D
)

del input.in
del output.out
del output2.out
```

8 String

8.1 Hashing

```
const int N = 1e6 + 9;
//Dont forget to call prec() function
int power(long long n, long long k, const int mod) {
  int ans = 1 % mod;
  n %= mod;
  if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}

const int MOD1 = 1e9+7, MOD2 = 1e9+9;
const int p1 = 31, p2 = 97;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] = {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
```

```
  ipw[0] = {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
  }

}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 - indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL * pw[i].first * s[i] % MOD1) % MOD1;
      p.second = (hs[i].second + 1LL * pw[i].second * s[i] % MOD2) % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r) { // 1 - indexed
    assert(1 <= l && l <= r && r <= n);
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l - 1].first + MOD1) * 1LL * ipw[l -
        1].first % MOD1;
    ans.second = (hs[r].second - hs[l - 1].second + MOD2) * 1LL * ipw[l -
        1].second % MOD2;
    return ans;
  }
  pair<int,int>merge_hash(int l1,int r1,int l2,int r2){
        auto p1 = get_hash(l1,r1);
        auto p2 = get_hash(l2,r2);
        int len = r1-l1+1;
        pair<int,int>ans;
        ans.first = {(p1.first + (p2.first*pw[len].first) % MOD1) % MOD1};
        ans.second = {(p1.second + (p2.second*pw[len].second) % MOD2) % MOD2};
        return ans;
  }
  pair<int, int> get_hash() {
    return get_hash(1, n);
  }
};
```

8.2 Multiset Hashing

```
const int MOD[2] = {998244353, 1000000007};
int BASE[2] = {0, 0};
int POW[2][MX];
int IPOW[2][MX];
int power(int a, int p, int m) {
  int ans = 1;
  a %= m;
  while (p) {
    if (p & 1) ans = (ans*a) % m;
    p >>= 1;
    a = (a*a) % m;
  }
  return ans;
}
void hash_pre() {//Call this idiot
  int b1, b2, i, j, inv;
  mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
  b1 = (500 + (rnd() % (MOD[0]-500*2+1)));
  b2 = 0;
  do {
    b2 = (500 + (rnd() % (MOD[1]-500*2+1)));
  } while (b1 == b2);
  BASE[0] = b1;
  BASE[1] = b2;
  for (i = 0; i < 2; ++i) {
    int *pw = POW[i], *ipw = IPOW[i], x = BASE[i], m = MOD[i];
    pw[0] = 1;
    ipw[0] = 1;
    inv = power(x, m-2, m);
    for (j = 1; j < MX; ++j) {
      pw[j] = (pw[j-1] * x) % m;
      ipw[j] = (ipw[j-1] * inv) % m;
    }
  }
}
template<typename T>
struct DoubleHash {
  int n;
```

```
  T s;
  vector<pair<int,int>> h;
  DoubleHash() {}
  DoubleHash(T s) : s(s) {//0 based
    n = s.size();
    h.resize(n+1);
        int *pw0 = POW[0], m0 = MOD[0];
    int *pw1 = POW[1], m1 = MOD[1];
    int i = 0;
    h[i] = {0, 0};
    for (i = 1; i <= n; ++i) {
      h[i] = {
        (h[i-1].first + pw0[s[i-1]]) % m0,
        (h[i-1].second + pw1[s[i-1]]) % m1,
      };
    }
  }
  pair<int,int> get_hash(int l, int r) {//0 based
    assert((0 <= l) && (l <= r) && (r < n));
    ++l; ++r;
    int *ipw0 = IPOW[0], m0 = MOD[0];
    int *ipw1 = IPOW[1], m1 = MOD[1];
    return {
      (h[r].first - h[l-1].first + m0) % m0,
      (h[r].second - h[l-1].second + m1) % m1,
    };
  }
  pair<int,int> get_hash() {
    return get_hash(0, n-1);
  }
};
```

8.3 Trie

```
struct TrieNode
{
    TrieNode* childNode[26];
    bool wordEnd;
    TrieNode(){
        wordEnd = 0;
        for(int i=0;i<26;i++){
            childNode[i] = NULL;
        }
    }
};
void insert_key(TrieNode* root , string &key){
    TrieNode *currNode = root;
    for(auto c : key){
        if(currNode->childNode[c-'a']==NULL){
           TrieNode* newNode = new TrieNode();
           currNode->childNode[c-'a']= newNode;
           currNode = currNode->childNode[c-'a'];
        }
        currNode = currNode -> childNode[c-'a'];
    }
    currNode->wordEnd = 1;
}
bool search_key(TrieNode *root , string &key){
    TrieNode *currNode = root;
    for(auto c : key){
        if(currNode->childNode[c-'a']==NULL) return 0;
        currNode = currNode->childNode[c-'a'];
    }
    return (currNode->wordEnd);
}
```

8.4 Z-algo

```
void zAlgo(){
       int L = 0, R = 0;
       for (int i = 1; i < n; i++) {
              if (i > R) {
              L = R = i;
                     while (R < n && s[R-L] == s[R]) R++;
                     z[i] = R-L; R--;
              } else {
              int k = i-L;
              if (z[k] < R-i+1) z[i] = z[k];
              else {
                     L = i;
                     while (R < n && s[R-L] == s[R]) R++;
                     z[i] = R-L; R--;
              }
              }
```

```
    }
}
```

9 Tree

9.1 Centroid Tree

```
vector<pair<int, int>> edgeList;
vector<bool> deleted;
vector<int> subtree;
inline int getD(int index, int s){
    return edgeList[index].ff ^ edgeList[index].ss ^ s;
}
void computeSubtrees(int root, vector<int>* edges, int parent){
    subtree[root] = 1;
    for(auto i : edges[root]){
        int dest = getD(i, root);
        if(!deleted[i] && dest != parent){
            computeSubtrees(dest, edges, root);
            subtree[root] += subtree[dest];
        }
    }
}
int findCentroid(int root, vector<int>* edges, int n, int parent){
    for(auto i : edges[root]){
        int dest = getD(i, root);
        if(!deleted[i] && dest != parent && subtree[dest] > n / 2){
            return findCentroid(dest, edges, n, root);
        }
    }
    return root;
}
int decompose(int root, vector<int>* edges, vector<int>* edgesN, int parent){
    computeSubtrees(root, edges, -1);
    int n = subtree[root];
    root = findCentroid(root, edges, n, -1);
    if(parent != -1){
        edgesN[root].pb(parent);
        edgesN[parent].pb(root);
    }
    for(auto i : edges[root]){
        int dest = getD(i, root);
        if(deleted[i])
            continue;
        deleted[i] = true;
        decompose(dest, edges, edgesN, root);
    }
    return root;
}
void solve() {
    int n;
    cin >> n;
    edgeList.clear();
    deleted.clear();
    vector<int>* edges= new vector<int>[n];
    for(int i = 0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
        a--, b--;
        edges[a].pb(sz(edgeList));
        edges[b].pb(sz(edgeList));
        edgeList.pb({a, b});
        deleted.pb(0);
    }
    vector<int>* edgesN = new vector<int>[n];
    subtree.resize(n);
    fill(all(subtree), 0);
    int root = 0;
    root = decompose(root, edges, edgesN, -1);
}
```

9.2 HLD

```
// No need to change anything here
struct BinaryLifting {
        int n;
        int maxLog;
        ll maxRequirement;
        vector<vector<int>> parent;
        vector<int> *edges;
        vector<int> logValues;
        bool precomputedLogs = false;
```

```
BinaryLifting(int n1, vector<int> *edges1, ll requirement, int root) {
	n = n1;
	edges = edges1;
	parent.resize(n);
	maxLog = log2(requirement + 1);
	maxRequirement = requirement;
	for (int i = 0; i < n; i++) {
		parent[i].resize(maxLog + 1);
		for (int j = 0; j <= maxLog; j++) {
			parent[i][j] = -1;
		}
	}
	fillParentTable(root);
	if (maxRequirement <= 1000000LL)
		precomputeLogs();
}
BinaryLifting() {}
void fillParentTable(int root) {
	vector<bool> visited(n);
	dfsBinaryLifting(root, visited);
	int intermediate = -1;
	for (int i = 1; i <= maxLog; i++) {
		for (int j = 0; j < n; j++) {
			intermediate = parent[j][i - 1];
			if (intermediate != -1) {
				parent[j][i] = parent[intermediate][i - 1];
			}
		}
	}
}
void dfsBinaryLifting(int root, vector<bool> &visited) {
	visited[root] = true;
	for (auto i : edges[root]) {
		if (!visited[i]) {
			parent[i][0] = root;
			dfsBinaryLifting(i, visited);
		}
	}
}
void precomputeLogs() {
	precomputedLogs = true;
	logValues.resize(maxRequirement + 1);
	logValues[1] = 0;
	for (int i = 2; i <= maxRequirement; i++) {
		logValues[i] = logValues[i / 2] + 1;
	}
}
int kthParent(int start, int k) {
	int a = start;
	while (k > 0) {
		int x = getLog(k);
		a = parent[a][x];
		if (a == -1)
			return a;
		k -= (1 << x);
	}
	return a;
}
int getLog(ll x) {
	return precomputedLogs ? logValues[x] : log2(x);
}
};

// No need to change anything here
struct LCA {
	int n;
	BinaryLifting *bl_object;
	vector<int> level;
	vector<int> *edges;
	LCA(int n1, vector<int> *edges1, int root, BinaryLifting *bl) {
		n = n1;
		bl_object = bl;
		edges = edges1;
		level.resize(n);
		dfsLCA(root, -1);
	}
	LCA() {}
	void dfsLCA(int root, int parent) {
		for (auto i : edges[root]) {
			if (i != parent) {
				level[i] = level[root] + 1;
				dfsLCA(i, root);
			}
		}
	}
	int getLCA(int a, int b) {
		if (level[a] > level[b]) {
			swap(a, b);
```

```
            }
            b = bl_object->kthParent(b, level[b] - level[a]);
            if (a == b)
                  return a;
            for (int i = bl_object->maxLog; i >= 0; i--) {
                  int parent1 = bl_object->parent[a][i];
                  int parent2 = bl_object->parent[b][i];
                  if (parent2 != parent1 && parent1 != -1 && parent2 !=
                      -1) {
                     a = parent1;
                     b = parent2;
                  }
            }
            return bl_object->parent[a][0];
      }
};


template<typename Node, typename Update>
struct SegTree {
      vector<Node> tree;
      vector<ll> arr; // type may change
      int n;
      SegTree(int a_len, vector<ll> &a) { // change if type updated
            arr = a;
            n = a_len;
            tree.resize(4 * n); fill(all(tree), Node());
            build(0, n - 1, 1);
      }
      SegTree() {}
      void build(int start, int end, int index) // Never change this
      {
            if (start == end)     {
                  tree[index] = Node(arr[start]);
                  return;
            }
            int mid = (start + end) / 2;
            build(start, mid, 2 * index);
            build(mid + 1, end, 2 * index + 1);
            tree[index].merge(tree[2 * index], tree[2 * index + 1]);
      }
      void update(int start, int end, int index, int query_index, Update &u)
          // Never Change this
      {
            if (start == end) {
                  u.apply(tree[index]);
                  return;
            }
            int mid = (start + end) / 2;
            if (mid >= query_index)
                  update(start, mid, 2 * index, query_index, u);
            else
                  update(mid + 1, end, 2 * index + 1, query_index, u);
            tree[index].merge(tree[2 * index], tree[2 * index + 1]);
      }
      Node query(int start, int end, int index, int left, int right) { //
          Never change this
            if (start > right || end < left)
                  return Node();
            if (start >= left && end <= right)
                  return tree[index];
            int mid = (start + end) / 2;
            Node l, r, ans;
            l = query(start, mid, 2 * index, left, right);
            r = query(mid + 1, end, 2 * index + 1, left, right);
            ans.merge(l, r);
            return ans;
      }
      void make_update(int index, ll val) { // pass in as many parameters as
          required
            Update new_update = Update(val); // may change
            update(0, n - 1, 1, index, new_update);
      }
      Node make_query(int left, int right) {
            return query(0, n - 1, 1, left, right);
      }
};

struct Node1 {
      ll val; // may change
      Node1() { // Identity element
            val = -INF;    // may change
      }
      Node1(ll p1) { // Actual Node
            val = p1; // may change
      }
      void merge(Node1 &l, Node1 &r) { // Merge two child nodes
            val = max(l.val, r.val); // may change
```

```
	}
};

struct Update1 {
	ll val; // may change
	Update1(ll p1) { // Actual Update
		val = p1; // may change
	}
	void apply(Node1 &a) { // apply update to given node
		a.val = val; // may change
	}
};
template<typename Node, typename Update>
struct HLD {
	int n;
	int rootHere;
	vector<int> *edges;
	vector<int> big_child;
	vector<int> subtree_sum;
	vector<int> chain;
	vector<int> label;
	vector<ll> values;
	SegTree<Node, Update> s1;
	LCA *lca_object;
	BinaryLifting *bl_object;
	HLD(int n1, vector<int> *edges1, int root1, vector<ll> &values1, LCA
	    *lca) {
		n = n1;
		lca_object = lca;
		bl_object = lca->bl_object;
		edges = edges1;
		rootHere = root1;
		big_child.resize(n);
		subtree_sum.resize(n);
		label.resize(n);
		chain.resize(n);
		values = values1;
		dfsPrecompute(rootHere, -1);
		int label_time = 0;
		dfsLabels(rootHere, -1, label_time);
		for (int i = 0; i < n; i++)
			chain[i] = i;
		dfsChains(rootHere, -1);
		s1 = SegTree<Node, Update>(n, values);
		for (int i = 0; i < n; i++) {
			s1.make_update(label[i], values[i]);
		}
		// debugHLD();
	}
	void dfsPrecompute(int root, int parent) {
		subtree_sum[root] = 1;
		big_child[root] = -1;
		int biggest = -1;
		for (auto i : edges[root]) {
			if (i != parent) {
				dfsPrecompute(i, root);
				subtree_sum[root] += subtree_sum[i];
				if (subtree_sum[i] > biggest) {
					big_child[root] = i;
					biggest = subtree_sum[i];
				}
			}
		}
	}
	void dfsLabels(int root, int parent, int &label_time) {
		label[root] = label_time++;
		if (big_child[root] != -1)
			dfsLabels(big_child[root], root, label_time);
		for (auto i : edges[root])
			if (i != parent && i != big_child[root])
				dfsLabels(i, root, label_time);
	}
	void dfsChains(int root, int parent) {
		if (big_child[root] != -1)
			chain[big_child[root]] = chain[root];
		for (auto i : edges[root])
			if (i != parent)
				dfsChains(i, root);
	}
	void debugHLD() {
		debug(big_child);
		debug(subtree_sum);
		debug(chain);
		debug(label);
		debug(values);
	}
	Node queryChain(int here, int toReach) {
		Node val = Node(0);
```

```
            int top;
            while (lca_object->level[here] > lca_object->level[toReach]) {
                top = chain[here];
                if (lca_object->level[top] <= lca_object->level[toReach])
                    top = bl_object->kthParent(here,
                        lca_object->level[here] -
                        lca_object->level[toReach] - 1);
                Node a1 = val;
                Node a2 = s1.make_query(label[top], label[here]);
                val.merge(a1, a2);
                here = bl_object->parent[top][0];
            }
            return val;
        }
        ll findAnswer(int u, int v) {
            int lca = lca_object->getLCA(u, v);
            Node n1 = queryChain(u, lca);
            Node n2 = queryChain(v, lca);
            Node merged;
            merged.merge(n1, n2);
            Node n3 = Node(s1.make_query(label[lca], label[lca]));
            Node ans;
            ans.merge(merged, n3);
            return ans.val;
        }
        void makeUpdateatIndex(int u, ll val) {
            s1.make_update(label[u], val);
        }
};
// Change accordingly for edge weights instead of node values
void solve() {
        int n, q;
        cin >> n >> q;
        vector<ll> values(n);
        for (int i = 0; i < n; i++)
            cin >> values[i];
        vector<int> *edges = new vector<int>[n];
        for (int i = 0; i < n - 1; i++) {
            int a, b;
            cin >> a >> b;
            edges[a - 1].pb(b - 1);
            edges[b - 1].pb(a - 1);
        }
        BinaryLifting bl_object = BinaryLifting(n, edges, n, 0);
        LCA lca_object = LCA(n, edges, 0, &bl_object);
        HLD<Node1, Update1> hld = HLD<Node1, Update1>(n, edges, 0, values,
            &lca_object);
        while (q--) {
            int a, b, c;
            cin >> a >> b >> c;
            if (a == 1) {
                hld.makeUpdateatIndex(b - 1, c);
            } else {
                cout << hld.findAnswer(b - 1, c - 1) << " ";
            }
        }
}
```

10 Z-sheet

10.1 combinatorics

1. Stars and bars: The number of ways to put n identical objects into k labeled boxes is
$$\binom{n+k-1}{k} = \binom{n+k-1}{k-1}$$

2. $x_1 + x_2 + \cdots + x_r = n$ where $x_i \geq 0$
$$\binom{n+r-1}{r-1}$$

3. $x_1 + x_2 + \cdots + x_r = n$ where $x_i \geq b$
$$\binom{(n-rb)+r-1}{r-1}$$

4. The Catalan number C_n represents the number of ways to correctly match n pairs of parentheses.
$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} \quad \text{for } n \geq 0$$

5. number of ways to partition n labeled objects in k (possibly empty) labeled sets where each set can have more than one object $= k^n$

6. The Stirling numbers of the first kind, denoted as $c(n,k)$ or $[n\ k]$, count the number of permutations of n elements with exactly k disjoint cycles.

$$c(n,k) = \begin{cases} 0 & \text{if } k > n \text{ or } k = 0 \text{ and } n > 0, \\ 1 & \text{if } n = k = 0, \text{ or } = k \\ (n-1)\,c(n-1,k) + c(n-1,k-1) & \end{cases}$$

7. The Stirling numbers of the second kind, denoted by $S(n,k)$, represent the number of ways to partition a set of n elements into k non-empty, disjoint subsets.

$$S(n,k) = \begin{cases} 0 & \text{if } k > n \text{ or } k = 0 \text{ and } n > 0, \\ 1 & \text{if } n = k \text{ or } n = 0, \\ k \cdot S(n-1,k) + S(n-1,k-1) & \text{otherwise.} \end{cases}$$

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n.$$

8. all formula that has sets as un-labeled just multiply with $k!$ if set were labeled.

9. The formula for the inclusion-exclusion principle is:

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = \sum_{i=1}^{n}|A_i| - \sum_{1\le i<j\le n}|A_i \cap A_j| + \sum_{1\le i<j<k\le n}|A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1}|A_1 \cap A_2 \cap \cdots \cap A_n|.$$

10. De-arrangement(1 to n all are there without matching their index):

$$!n = n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$