```
#define endl '₩n'
#define p_q priority_queue
#define pbk push_back
#define double long double
#define rep(i, a, b) for (int i=a; i<=b; i++)
#define all(v) (v).begin(), (v).end()

using namespace std;
using ll = long long;
#define int ll

struct Matrix {
    vector<vector<int>> m;
    int l;
    Matrix(int l=10) : l(l) {
        m=vector<vector<mint>>(l+1,vector<mint>(l+1));
    }

    void setEye() {
        for(int i=1;i<=l;i++) m[i][i]=1;
    }

    Matrix operator * (const Matrix& other) const
        { Matrix ret(l);
        for(int i=1;i<=l;i++) {
            for(int j=1;j<=l;j++) {
                for(int k=1;k<=l;k++) {
                    ret.m[i][j] += m[i][k] * other.m[k][j];
                }
            }
        }
        return ret;
    }

    Matrix operator + (const Matrix& other) const
        { Matrix ret;
        for(int i=1;i<=l;i++) {
            for(int j=1;j<=l;j++) {
                ret.m[i][j] = m[i][j] + other.m[i][j];
            }
        }
        return ret;
    }
    Matrix power(ll k) { //matrix = matrix.power(k);
        Matrix ret;
        if(k==0) {
            ret.setEye();
            return ret;
        }
        if(k==1) {
            return *this;
        }
        ret = power(k/2);
        ret = ret * ret;
        if(k&1) ret = ret * (*this);
        return ret;
    }
    void rotate90CC() {
        vector<vector<mint>> after(l+1, vector<int>(l+1));
        for(int i=1;i<=l;i++) {
            for(int j=1;j<=l;j++) {
                after[i][j] = m[j][l-i+1];
            }
        }
        for(int i=1;i<=l;i++) {
            for(int j=1;j<=l;j++) {
                m[i][j] = after[i][j];
            }
        }
    }
    void transpose() {
        vector<vector<int>> after(l+1, vector<mint>(l+1));
        for(int i=1;i<=l;i++) {
            for(int j=1;j<=l;j++) {
                after[i][j] = m[j][i];
            }
        }
        for(int i=1;i<=l;i++) {
            for(int j=1;j<=l;j++) {
                m[i][j] = after[i][j];
            }
        }
    }
};

template<int D, typename T>
struct Vec : public vector<Vec<D - 1, T>> {
  static_assert(D >= 1, "Vector dimension must be greater
than zero!");
  template<typename... Args>
  Vec(int n = 0, Args... args) : vector<Vec<D - 1, T>>(n,
Vec<D - 1, T>(args...)) {}
};

template<typename T>
struct Vec<1, T> : public vector<T> {
  Vec(int n = 0, const T& val = T()) : vector<T>(n, val) {}
};
```

```
template<typename T, class BinaryOperation = plus<T>>
struct Segtree {
   vector<T> a;
   vector<T> s;
   int n;
   BinaryOperation op;
   Segtree(int n = 1) : n(n), op(BinaryOperation()) {
      a.resize(n + 1);
      s.resize(4 * n + 1);
   }
   T segment(int node, int nodeLeft, int nodeRight) {
      if (nodeLeft == nodeRight) {
         return s[node] = a[nodeLeft];
      }
      int mid = (nodeLeft + nodeRight) / 2;
      return s[node] = op(segment(node * 2, nodeLeft, mid), segment(node * 2 + 1, mid + 1, nodeRight));
   }
   void update(int node, int nodeLeft, int nodeRight, int idx, T num) {
      if (idx < nodeLeft || nodeRight < idx) return;
      if (nodeLeft == nodeRight) {
         s[node] = num;
         return;
      }
      int mid = (nodeLeft + nodeRight) / 2;
      update(node * 2, nodeLeft, mid, idx, num);
      update(node * 2 + 1, mid + 1, nodeRight, idx, num);

      s[node] = op(s[node * 2], s[node * 2 + 1]);
   }
   T query(int node, int l, int r, int nodeLeft, int nodeRight) {
      if (nodeRight < l || r < nodeLeft) return 0; //could be 0, 1e9, -1e9
      if (l <= nodeLeft && nodeRight <= r) return s[node];
      int mid = nodeLeft + nodeRight >> 1;
      return op(query(node * 2, l, r, nodeLeft, mid), query(node * 2 + 1, l, r, mid + 1, nodeRight));
   }
};

struct lazySegtree {
    vi a;
    vi s;
    vi lazy;
    int n;
    lazySegtree(int n) : n(n)
        { a.resize(n+1);
        s.resize(4*n+1);
        lazy.resize(4*n+1);
    }

    int merge(int a, int b) {
        return a+b;
    }

    int segment(int node, int nodeLeft, int nodeRight) { // use when s, a is available and segment tree is about sum
        if (nodeLeft == nodeRight)
            { return s[node] =
            a[nodeLeft];
        }
        int mid = (nodeLeft+nodeRight)/2;
        return s[node] = merge(segment(node * 2, nodeLeft, mid), segment(node * 2 + 1, mid + 1, nodeRight));
    }
    void propagation(int node, int l, int r)
        { if (lazy[node]) {
            s[node] += (r - l + 1) * lazy[node];
            if (l != r) {
                lazy[node * 2] += lazy[node];
                lazy[node * 2 + 1] += lazy[node];
            }
            lazy[node] = 0;
        }
    }
    void update(int node, int l, int r, int nodeLeft, int nodeRight, int dif) { //This is for lazy propagation
        propagation(node, nodeLeft, nodeRight); if (nodeRight < l || r < nodeLeft) return; if (l <= nodeLeft && nodeRight <= r) {
            s[node] += (nodeRight-nodeLeft + 1) * dif;
            if (nodeLeft != nodeRight) {
                lazy[node * 2] += dif;
                lazy[node * 2 + 1] += dif;
            }
            return;
        }
        int mid = (nodeLeft+nodeRight)/2; update(node * 2, l, r, nodeLeft, mid, dif); update(node * 2 + 1, l, r, mid + 1, nodeRight, dif); s[node] = merge(s[node * 2], s[node * 2 + 1]);
```

```
    }
    ll query(int node, int l, int r, int nodeLeft, int
nodeRight) { //s should be vll
        propagation(node, nodeLeft, nodeRight);
        if (nodeRight < l || r < nodeLeft) return 0;
        if (l <= nodeLeft && nodeRight <= r) {
            return s[node];
        }
        int mid = (nodeLeft+nodeRight)/2;
        return merge(query(node * 2, l, r, nodeLeft, mid),
query(node * 2+1, l, r, mid+1, nodeRight));
    }
};

struct DSU {
    vi parent;
    vi depth; //tree depth (maximum distance from root
node)
    vi d;
    vi sz;
    stack<tiii> rb; //rollback
    DSU(int n = 1) {
        parent = vi(n + 1);
        depth = vi(n + 1, 0);
        d = vi(n + 1, 0);
        sz = vi(n + 1, 0);
        iota(parent.begin() + 1, parent.end(), 1);
        fill(sz.begin() + 1, sz.end(), 1);
    }
    int getParent(int num) {
        if (num == parent[num]) return num;
        int p = getParent(parent[num]);
        // d[num] += d[parent[num]];
        return parent[num] = p; //path compression
    }

    //modify merge to get difference between a and b
    void merge(int a, int b, ll w = 0) { //merge to b
        // int pa = getParent(a);
        // int pb = getParent(b);

        // d[pb] = d[a] + w - d[b];
        // parent[pb] = pa;

        a = getParent(a);
        b = getParent(b);

        if(d[a]<d[b]) swap(a,b);

        bool flag = 0;
        if(d[a]==d[b]) {
            d[a]++;
            flag=1;
        }
        parent[b] = a;
        rb.push({a,b,flag});
        sz[a] += sz[b];
    }
    bool isSameParent(int a, int b) {
        return getParent(a)==getParent(b);
    }
    void rollback() {
        if(rb.empty()) return;
        auto [u,v,flag] = rb.top(); //u is root node and v is
attached to u
        rb.pop();
        if(flag) d[u]--;
        parent[v] = v;
        sz[u] -= sz[v];
    }
};
```

```
vector<string> split(string input, char delimiter=' ')
    { vector<string> answer;
    stringstream ss(input);
    string temp;
    while (getline(ss, temp, delimiter))
        { answer.push_back(temp);
    }
    return answer;
}

vvi SCC;
int d[MAX];
bool finished[MAX];
vi edge[MAX];
int id, SN=0; //mark sn[i]
stack<int> s;
int sn[MAX]; //sn[i] is SCC number to which it belongs to. If sn is big, then it is at the start of DAG. If small, it is at the end of DAG. If one wants to start from the beginning of DAG, start from the largest of sn.
int SCCnode[MAX] {}; //if SCCbfs is needed...
int nodeValue[MAX] {}; //if SCCbfs is needed...
int dfs(int x) {
    d[x] = ++id; //노드마다 고유한 아이디 부여
    s.push(x); //스택에 자기 자신을 삽입 int
    parent = d[x];
    for (auto i : edge[x]) {
        if (d[i] == 0) { //방문 안 한 이웃
            parent = min(parent, dfs(i));
        }
        else if (finished[i] == 0) { //처리 중인 이웃
            parent = min(parent, d[i]);
        }
    }
    if (parent == d[x]) {
        vector<int> scc;
        while (true) {
            int t = s.top();
            s.pop();
            scc.push_back(t);
            finished[t] = 1;
            sn[t] = SN;
            //d[t] = x; //to make scc recognizable with d
            //SCCnode[SN] += nodeValue[t]; if (t == x)
            break;
        }
        SN++;
        SCC.push_back(scc); //SCC 의 SN 번째 그래프랑 대응된다
    }
    return parent;
}

vi SCCedge[MAX]; //index refers to SN. Could be replaced with set if you don't want to overlap
int inDegree[MAX]; //index refers to
SN void SCCtopology_sort() {
    for(int i=1;i<=N;i++) { //id starts with 1
        for(auto next : edge[i]) {
            if(sn[next]!=sn[i])
                { SCCedge[sn[i]].pbk(sn[next]); //SN could be overlapped. Could be solved with set but it is often not needed
                inDegree[sn[next]]+=1; //If inDegree is 0, then it is the start of the SCC graph. There could be many
            }
        }
    }
}

//If SCC sum is needed
int SCCdp[MAX] {};
void SCCbfs(int x) { //x is sn. bfs graph is not vertex graph but scc graph. scc graph is DAG so visited array is not needed
    //bfs starts with x. Function flows through SCC graph (topologically)
    qi q;
    q.push(x);
    SCCdp[x] = SCCnode[x];

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

        for(auto next : SCCedge[cur]) {
            if(SCCdp[next] < SCCdp[cur] + nodeValue[next]) {
                SCCdp[next] = SCCdp[cur] + nodeValue[next];
                q.push(next);
            }
        }
    }
}
```

```
bool inQ[MAX] {};
vpii edge[MAX];
int d[MAX];
int cycle[MAX] {};
void SPFA(int start) {
    fill(d+1, d+1+N, INF);
    qi q;
    d[start] =0;
    q.push(start);
    inQ[start] = 1;
    cycle[start] += 1;
    while(!q.empty()) {
        int cur = q.front();
        q.pop();
        inQ[cur] = 0;
        for(auto next : edge[cur]) {
            if(d[next.first] > d[cur] + next.second) {
                d[next.first] = d[cur] + next.second;
                if(!inQ[next.first]) {
                    cycle[next.first] += 1;
                    if(cycle[next.first]>=N) {
                        cout << "CYCLE!!!!" << endl;
                        return;
                    }
                    q.push(next.first);
                    inQ[next.first] = 1;
                }
            }
        }
    }
}

//network flow with dinic
vector<int> edge[MAX];
int f[MAX][MAX], c[MAX][MAX];
int level[MAX], work[MAX];
int src, sink;
int bias;

void addEdge(int s, int e, int value=1) { //Decide whether graph is directed graph or undirected graph
    edge[s].pbk(e);
    edge[e].pbk(s);
    c[s][e] = value;
}
bool bfs() { // to create level graph and decide if flow is no longer needed
    queue<int> q;
    q.push(src); //
    memset(level, -1, sizeof(level));
    level[src] = 0;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        for (auto next : edge[cur]) {
            if (level[next] == -1 && c[cur][next] - f[cur][next] > 0) {
                q.push(next);
                level[next] = level[cur] + 1;
            }
        }
    }
    if (level[sink] == -1) return false;
    else return true;
}
int dfs(int cur, int flow) { //cur node has flow to offer to the next level
    if (cur == sink) return flow;

    for (int& i = work[cur]; i < edge[cur].size(); i++)
        { int next = edge[cur][i];
        if (level[next] == level[cur] + 1 && c[cur][next] - f[cur][next] > 0) {
            int ret = dfs(next, min(flow, c[cur][next] - f[cur][next]));
            if (ret > 0) {
                f[cur][next] += ret;
                f[next][cur] -= ret; //always remember to create reverse flow
                return ret;
            }
        }
    }
    return 0;
}
int Network_Flow() {
    int totalFlow = 0;
    while (bfs()) {
        memset(work, 0, sizeof(work));
        while (true) {
            int flow = dfs(src, INF); //INF varies from range to range
            if (flow == 0) break;
            ● totalFlow += flow;
        }
    }
    return totalFlow;
}
```

```
// This is CCW

//depending on input, the value ccw could be beyond INTEGER. Even long long could be dangerous. Watch carefully on input range
/*
struct Point {
	ll x, y;
	Point(ll x, ll y) : x(x), y(y) {}
};
struct Line {
	Point p1, p2;
};
ll CCW(Point A, Point B, Point C) { //A, B, C is in order
	ll ccw = (B.x - A.x) * (C.y - A.y) - (C.x - A.x) * (B.y - A.y); //Cross product
	if(ccw>0) return 1;
	else if(ccw<0) return -1;
	else return 0;
}
//
int LineInterSection(Line l1, Line l2) {
		ll l1_l2 = CCW(l1.p1, l1.p2, l2.p1) * CCW(l1.p1, l1.p2, l2.p2);
	ll l2_l1 = CCW(l2.p1, l2.p2, l1.p1) * CCW(l2.p1, l2.p2, l1.p2);


		if(l1_l2==0 && l2_l1==0) { //l1 and l2 is on the same line. If p1 <= p4 && p3 <= p2, the line meets.
			if(l1.p1.x > l1.p2.x) swap(l1.p1, l1.p2);
			if(l2.p1.x > l2.p2.x) swap(l2.p1, l2.p2);

			return l1.p1.x <= l2.p2.x && l2.p1.x <= l1.p2.x;
		}
	return (l1_l2 <= 0) && (l2_l1 <= 0);
}
vector<Point> v;
bool cmp(const Point& a, const Point& b) {
	ll ccw = CCW(v[0], a, b);
	if(ccw) return ccw>0;
	if(a.y==b.y) return a.x < b.x;
	return a.y < b.y;
}
stack<Point> s;
//to find if X point exists within the convex polygon, do CCW(i,i+1,X point) and see if CCW value is the same for all void Convex_Hull() {
	sort(all(v), [](Point a, Point b) {
		if(a.y==b.y) return a.x<b.x;
		return a.y<b.y;
	});

	sort(v.begin()+1, v.end(), cmp);

	s.push(v[0]);
	s.push(v[1]);

	rep(i,2,N-1) {
		while(s.size()>=2) {
			auto t2 = s.top();
			s.pop();
			auto t1 = s.top();

			if(CCW(t1, t2, v[i])>0) {
				s.push(t2);
				break;
			}
		}
		s.push(v[i]);
	}
	cout << s.size();
}

void LIS(vi& v) { //vector v's size is N
	vi lis;
	vi dp(N);

	fill(all(dp), 1); //set all dp element to at least 1
	rep(i,0,N-1) { // N
		int cur = v[i];

		 auto iter = lower_bound(all(lis), cur);
		//if found, replace the value with cur. if not, cur is the highest value of lis
		if(iter!=lis.end()) {
			*iter = cur;
			dp[i] = iter - lis.begin()+1;
		}
		else {
			lis.pbk(cur);
			dp[i] = lis.size();
		}
	}
	stack<int> s;

	int sz = *max_element(all(dp)); //LIS size
	cout << sz << endl;
```

```
    for(int i=N-1;i>=0;i--) {
        if(sz==dp[i]) {
            s.push(v[i]);
            sz--;
        }
    }

    while(!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
}

// This is LCA with binary algorithm (O(logN))
/*
#define MAX 100001
vi edge[MAX]; //vpii
edge[MAX] {}; //int
d[MAX][18];
int parent[MAX][18]; //18 is log2(MAX)
int level[MAX];
int maxLevel;

//init -> set_tree -> LCA(a,b)
void init() {
    cin >> N; //Has N node and N-1 edges
    for(int i=0;i<N-1;i++) {
        int a, b;
        cin >> a >> b;
        edge[a].pbk(b);
        edge[b].pbk(a);
    }
    maxLevel = (int)floor(log2(MAX));
}
//maps node and depth and set 2^i
parent //use before LCA function
//set_tree(root, 0) should do it
//if node 0 exists, then this function needs to be
altered //get root node by inDegree array.
void set_tree(int node, int pnode)
    { level[node] = level[pnode]+1;
    parent[node][0] = pnode;

    for(int i=1;i<=maxLevel;i++) { int prev =
        parent[node][i-1]; parent[node][i] =
        parent[prev][i-1]; //d[node][i] =
        d[prev][i-1] + d[node][i-1];
    }

    for(auto child : edge[node])
        { if(child==pnode) continue;
        //d[child.first][0] = child.second;
        set_tree(child, node);
    }
}

int LCA(int a, int b) {
    if(a==1 || b==1) return 1;

    int target = a, compare = b;
    if(level[a] < level[b]) swap(target, compare); //target is
deeper

    // int answer = 0; //for length
    //set level[] equal
    if(level[target]!=level[compare]) {
        for(int i=maxLevel;i>=0;i--) { if(level[parent[target][i]]
            >= level[compare]) {
                //answer += d[target][i];
                target = parent[target][i];
            }
        }
    }

    int ret = target;

    //set target==compare
    if(target!=compare) {
        for(int i=maxLevel;i>=0;i--)
            { if(parent[target][i]!=parent[compare][i]) {
                // answer += d[target][i];
                // answer += d[compare][i];

                target = parent[target][i];
                compare = parent[compare][i];
            }
            ret = parent[target][i];
        }
    }
    //answer += d[target][0] + d[compare][0];
    return ret;
}
```

```
#define TRIENODE 26
struct Trie {
    Trie *next[TRIENODE]; // 다음 노드를 가리키는 포인터 배열
    //map<string, Trie*> next;
    bool finish;
    Trie() {
        fill(next, next + TRIENODE, nullptr);
        finish = false;
    }

    ~Trie() {
        for (int i = 0; i < TRIENODE; i++)
            { if (next[i]) delete next[i];
        }
    }

    int getIdx(char c) {
        return c-'A';
    }
    void insert(const string& str, int k=0)
        { if (k==str.size()) {
            finish = true;
            return;
        }

        int curKey = getIdx(str[k]); //or 'a' or '0'

        if (next[curKey]==nullptr) next[curKey] = new Trie();

        next[curKey]->insert(str, k+1); // 다음 문자 삽입
    }

    bool find(const string& str, int k=0) {
        if (k==str.size()) return true; // 문자열이 끝나는 위치를 반환
        int curKey = getIdx(str[k]); //or 'a' or '0'
        if (next[curKey] == nullptr) return false; // 찾는 값이 존재하지 않음
         return next[curKey]->find(str, k+1); // 다음 문자를 탐색
    }
};
```

```
vi getPi(const string& pattern) { int
    patternSize = pattern.size();

    vi pi(patternSize);

    int j = 0;
    for(int i=1;i<patternSize;i++) {
        while(j>0 && pattern[i] != pattern[j])
            { j = pi[j-1];
        }

        if(pattern[i]==pattern[j]) {
            j+=1;
            pi[i] = j;
        }
    }
    return pi;
}

void KMP(const string& parent, const string& pattern)
    { vi pi = getPi(pattern);

    int parentSize = parent.size();
    int patternSize = pattern.size();

    int j =0;

    for(int i=0;i<parentSize;i++) {
        while(j>0 && parent[i] != pattern[j])
            { j = pi[j-1];
        }
        if(parent[i] == pattern[j]) {
            if(j == patternSize-1) {
                cout << i-patternSize + 2 << endl;
//index starts from 1
                j = pi[j];
            }
            else {
                j++;
            }
        }
    }
}
```

```
int A[MAX] {}; //saves the sz of palindrome each side including i itself
string preprocess(const string& str) { //to find even palindrome as well
    string temp;
    for(auto c : str) {
        temp.pbk('#');
        temp.pbk(c);
    }
    temp.pbk('#');
    return temp;
}
void manacher(string str) {
    int r = 0, p = 0; //p is the value that maximize j+A[j]
    str = preprocess(str);
    int sz = str.size();

    for (int i = 0; i < sz; i++) {
        if (i <= r) {
            A[i] = min(A[2 * p - i], r - i);
        }
        while (i-A[i]-1>=0 && i+A[i]+1<sz && str[i-A[i]-1]==str[i+A[i]+1]) {
            A[i]++;
        }
        if (r < i + A[i]) {
            r = i + A[i];
            p = i;
        }
    }
}
```

```
#define MAX 1001
vi edge[MAX];
bool done[MAX] {};
int b[MAX] {};
//for 1~N memset(done, done+MAX, 0), if(dfs(i)) cnt++;
bool dfs(int cur) {
    for(auto next : edge[cur])
        { if(done[next])
        continue; done[next] =
        1;

        if(b[next]==0 || dfs(b[next])) {
            b[next] = cur;
            return true;
        }
    }
    return false;
}
```

```
//big integer addition
string add(string a, string b) {
  // Initialize variables
  string result = "";
  int carry = 0;
  int a_len = a.length();
  int b_len = b.length();

  // Loop through both strings, starting at the last character
  for (int i = a_len - 1, j = b_len - 1; i >= 0 || j >= 0 || carry; i--, j--)
  {
    // Get the current digit of each string, or 0 if we have reached the beginning
    int x = (i >= 0) ? a[i] - '0' : 0;
    int y = (j >= 0) ? b[j] - '0' : 0;

    // Add the digits and carry, and store the result
    result = char((x + y + carry) % 10 + '0') + result;

    // Calculate the new carry
    carry = (x + y + carry) / 10;
    cout << "CARRY : " << carry << endl;
  }

  return result;
}
```