```
void rec(vi &v,vv &all_combs ,vi &comb,int target,int idx,int &counter){
  if (target == 0){
     vi temp;
     temp = comb;
     all_combs.push_back(temp);
     temp.clear();
     counter++;
  }

  if (target < 0){
     return;
  }

  for (int i = idx; i < (int)v.size(); i++)
  {
     comb.push_back(v[i]);
     rec(v,all_combs,comb,target-v[i],i,counter);
     comb.pop_back();

  }
  return;

}
```

COIN CHANGE

```
vector <vector<lli>>memo(1001,vector<lli>(1001,-1));
lli change(int idx,int N, vector <int> val){
  if (N == 0){
     memo[idx][N] = 1;
     return 1;
  }
  if (N < 0 || idx < 0){
     return 0;
  }
  if(memo[idx][N] > -1){
     return memo[idx][N];
  }
  if(val[idx] > N){
     memo[idx][N] = change(idx-1,N,val);
  }
  else
     memo[idx][N] = change(idx,N-val[idx],val) + change(idx-1,N,val);
  return memo[idx][N];
}
```

Knapsack

```
long long max_val(int idx, vector<int>& values, vector<int>& weights, int W, int n,
vector<vector<long long>>& memo) {
```

```
    if (idx == n || W == 0) {
        return 0;
    }
    if (memo[idx][W] != -1) {
        return memo[idx][W];
    }
    if (weights[idx] > W) {
        memo[idx][W] = max_val(idx + 1, values, weights, W, n, memo);
    }
    else {
        memo[idx][W] = max(max_val(idx + 1, values, weights, W, n, memo),
                           max_val(idx + 1, values, weights, W - weights[idx], n, memo) + values[idx]);
    }
    return memo[idx][W];
}
```

LCS

```
int dp(string s1, string s2 , int idx_1 , int idx_2){
    if (idx_1 == (int)s1.size() || idx_2 == (int)s2.size())
        return 0;
    if(memo[idx_1][idx_2] > -1)
        return memo[idx_1][idx_2];
    if(s1[idx_1] == s2[idx_2]){
            memo[idx_1][idx_2] =  1 + dp(s1,s2,idx_1 +1, idx_2 +1);
        }
    else
        memo[idx_1][idx_2] = max(dp(s1,s2,idx_1 + 1,idx_2),dp(s1,s2,idx_1,idx_2 +1));

    return memo[idx_1][idx_2];
```

GRID PATHS

```
int rec(vector<vector<char>>& grid, int n, int x, int y, vector<vector<int>>& memo) {
    if (grid[0][0] == '*' or grid[n-1][n-1] == '*')
        return 0;
    if (x >= n or y >= n)
        return 0;
    if (x == n - 1 && y == n - 1)
        return 1;
    if (grid[x][y] == '*')
        return 0;
    if (memo[x][y] != -1)
        return memo[x][y];

    memo[x][y] = (rec(grid, n, x + 1, y, memo) + rec(grid, n, x, y + 1, memo)) % 1000000007;
    return memo[x][y];
}
```

//

GRID PATHS BOTTOM UP

```
lli c(vector<vector<char>>grid, int n){
    if (grid[n-1][n-1] == '*' || grid[0][0] == '*')
        return 0;
    else{
        DP[n-1][n-1] = 1;
        for (int i = n-2 ; i >= 0; i--){
            if (grid[i][n-1] == '*')
```

```
            DP[i][n-1] = 0;
        else
            DP[i][n-1] = (DP[i+1][n-1] + DP[i][n])% 1000000007;
    }
    for(int j = n-2 ; j >= 0; j--){
        if (grid[n-1][j] == '*')
            DP[n-1][j] = 0;
        else
            DP[n-1][j] = (DP[n][j] + DP[n-1][j+1])% 1000000007;
    }
    for(int i = n-2 ; i >= 0; i--){
        for(int j = n-2 ; j >= 0; j--){
            if (grid[i][j] == '*')
                DP[i][j] = 0;
            else
                DP[i][j] = (DP[i+1][j] + DP[i][j+1])% 1000000007;
        }
    }

  }
  return DP[0][0];
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
auto ks(int n, int W , vector<lli>v , vector <lli>w){
        for(int i = 0 ; i <= n ; i++)
                DP[i][0] = 0;
        for(int j = 1; j <= W; j++)
                DP[0][j] = 0;
        for (int i = 1; i <= n ; i++){
                for(int j = 1; j <= W ; j++){
                        if (w[i-1] > j){
                                DP[i][j] = DP[i-1][j];
                        }
                        else{
                                DP[i][j] = max(DP[i-1][j],v[i-1] + DP[i-1][j-w[i-1]]);
                        }
                }
        }
        return DP[n][W];
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
auto cc(vector<int>values,int m,int N){
  for (int i = 0; i <= m; i++)
    memo[i][0] = 1;
  for (int j = 1; j <= N; j++)
    memo[0][j] = 0;

  for (int i = 1; i <= m; i++)
  {
    for (int j = 1; j <= N; j++)
    {
      if(values[i] > j)
        memo[i][j] = memo[i-1][j];
      else
        memo[i][j] = memo[i-1][j] + memo[i][j-values[i]];
    }
```

```
    }
    return memo[m][N];
}
```

GRAPHS

```
auto BFS(vector <bool>check,int s, int e){
    check[s] = true;
    q.push(s);
    dist[s] = 0;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for (auto i : adj[u])
        {
            if(check[i] == false){
                check[i] = true;
                q.push(i);
                dist[i] = dist[u] + 1;
            }
        }

    }
    return dist[e];
}
```

//

Ver se grafo é bipartido

```
bool BFS(vector <int>color, int v, vector<vector<int>>adj){
    queue <int> q;
    color[v] = 1;
    q.push(v);
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for (auto i : adj[u])
        {
            if(color[i] == -1){
                color[i] = color[u]-1;
                q.push(i);
            }
            else if(color[i] == color[u])
                return false;
        }
    }
    return true;
}
```

//

```
void DFS(vector<bool>&check,vector<vector<lli>> &adj,int v){
    check[v] = true;
    for (auto i : adj[v])
    {
        if(check[i] == false)
            DFS(check,adj,i);
    }
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                                    TOPOLOGICAL ORDER-DFS
void dfs(int u) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v);
        }
    }
    stk.push(u);
}
int main() {
    int n, m;
    cin >> n >> m;
    adj.resize(n + 1);
    visited.resize(n + 1);
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }
    for (int u = 1; u <= n; u++) {
        if (!visited[u]) {
            dfs(u);
        }
    }
    cout << stk.top();
    stk.pop();
    while (!stk.empty()) {
        cout << " " << stk.top();
        stk.pop();
    }
    cout << endl;
    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                                    TOPOLOGICAL ORDER BFS
void TS(vector <int> &d,int n){
    queue<int> q2;
    queue <int> q;
    for(int i = 1; i < n ; i++){
        if(d[i] == 0){
            q.push(i);
        }
    }
    while(!q.empty()){
        int u = q.front();
        q.pop();
        q2.push(u);
        for(auto nbr : adj[u]){
            d[nbr]--;
            if(d[nbr] == 0)
```

```
            q.push(nbr);
        }
    }
    cout << q2.front();
    q2.pop();
    for(int i = 1; i < n ; i++){
        cout << " " << q2.front();
        q2.pop();
    }
    cout << endl;
}
```

//

ARTICULATION POINTS

```
vector <vector <int>>adj(5001);
vector <bool> check(1001);
vector <int> dfs(1001,0);
vector <int> low(1001,-1);
vector <int> parent(1001,-1);
int t = 1;
int c = 0;

void AP(int v){
    low[v] = dfs[v] = t++;
    for (auto nbr:adj[v]){
        if (dfs[nbr] == 0){
            parent[nbr] = v;
            AP(nbr);
            low[v] = min(low[v],low[nbr]);
            if((dfs[v] == 1) && (dfs[nbr] != 2) && (check[v] == false)){
                //cout << v << endl;
                c++;
            }
            if((dfs[v] != 1) && (low[nbr] >= dfs[v]) && (check[v] == false)){
                c++;
                //cout << v << endl;
            }
            check[v] = true;
        }
        else if (parent[v] != nbr){
            low[v] = min(low[v],dfs[nbr]);
        }
    }
}
```

//

SCC

```
vector<vector<int>> adj(1001);
vector<bool> on_stack(1001);
vector<int> dfs(1001, 0);
vector<int> low(1001, -1);
int t = 1;
int c = 0;
stack<int> S;
```

```
void Tarjan(int v){
    low[v] = dfs[v] = t++;
    S.push(v);
    on_stack[v] = true;
    for(auto nbr:adj[v]){
        if(dfs[nbr] == 0){
            Tarjan(nbr);
            low[v] = min(low[v],low[nbr]);
        }
        else if(on_stack[nbr] == true){
            low[v] = min(low[v],dfs[nbr]);
        }
    }
    if (low[v] == dfs[v]){
        int nbr;
        do {
            nbr = S.top();
            S.pop();
            on_stack[nbr] = false;
        } while (nbr != v);
        c++;
    }
}
```

//

DIJKSTRA

```
#define INF INT_MAX

typedef pair<int, int> pii;

vector<int> dijkstra(vector<vector<pii>>& graph, int source, int target) {
    int n = graph.size();
    vector<int> dist(n, INF);
    vector<bool> visited(n, false);

    dist[source] = 0;

    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push(make_pair(0, source));

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        if (visited[u]) {
            continue;
        }
        visited[u] = true;
        if (u == target) {
            break;
        }
        for (auto& neighbor : graph[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;
```

```
            if (dist[v] > dist[u] + weight) {
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v));
            }
        }
    }

    return dist;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                    Bellman Ford
vector<int> BF(vector<vector<pii>>& graph, int source) {
    int n = (int)graph.size();
    vector<int> dist(n+1, INF);
    dist[source] = 0;

    for (int i = 1; i <= n-1; i++) {
        for (int u = 1; u <= n-1; u++) {
            for (auto nbr : graph[u]) {
                int v = nbr.first;
                int weight = nbr.second;
                if (dist[v] > dist[u] + weight) {
                    dist[v] = dist[u] + weight;
                }
            }
        }
    }

    for (int u = 1; u <= n-1; u++) {
        for (auto nbr : graph[u]) {
            int v = nbr.first;
            int weight = nbr.second;
            if (dist[v] > dist[u] + weight) {
                flag = true;
            }
        }
    }

    return dist;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                            FLOYD WARSHALL
void Floyd_Warshall(vector<vector<int>>& matrix) {
    int numVertices = (int)matrix.size();
    for (int k = 0; k < numVertices; k++) {
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                if (matrix[i][k] != INT_MAX && matrix[k][j] != INT_MAX) {
                    matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j]);
                }
            }
        }
    }
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                        UNION FIND
void makeSet(int v) {
    parent[v] = v;
}

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

void unionSets(int u, int v) {
    int root1 = findSet(u);
    int root2 = findSet(v);
    parent[root2] = root1;
}

bool check(int u, int v) {
    return findSet(u) == findSet(v);
}

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        makeSet(i);
    }
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        unionSets(u, v);
    }
    int q;
    cin >> q;
    for (int i = 0; i < q; i++) {
        int u, v;
        cin >> u >> v;
        if (check(u, v)) {
            cout << "YES" << endl;
        } else {
            cout << "NO" << endl;
        }
    }
}
(KRUSKAL) ->> Minimum Spanning tree
void Kruskal(int NumVertices) {
    for (int i = 1; i <= NumVertices; i++) {
        makeSet(i);
    }

    sort(graph.begin(), graph.end());
```

```
    for (auto& weightedEdge : graph) {
        int u = weightedEdge.second.first;
        int v = weightedEdge.second.second;

        if (findSet(u) != findSet(v)) {
            ANS += weightedEdge.first;
            unionSets(u, v);
        }
    }
}
```

//

Finding the largest group of people

```
vector<int> parent(5001);
vector<int> setSize(5001);
map<string,int> my_map;

void makeSet(int v) {
    parent[v] = v;
    setSize[v] = 1;
}

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

void unionSets(int u, int v) {
    int root1 = findSet(u);
    int root2 = findSet(v);
    if (root1 != root2) {
        if (setSize[root1] < setSize[root2]) {
            parent[root1] = root2;
            setSize[root2] += setSize[root1];
        } else {
            parent[root2] = root1;
            setSize[root1] += setSize[root2];
        }
    }
}

bool check(int u, int v) {
    return findSet(u) == findSet(v);
}

int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 0; i < n; i++)
    {
        string name;
```

```
        cin >> name;
        my_map[name] = i+1;
    }

    for (int i = 1; i <= n; i++) {
        makeSet(i);
    }
    for (int i = 0; i < m; i++) {
        string u, v;
        cin >> u >> v;
        unionSets(my_map[u], my_map[v]);
    }
    int largestSetSize = 0;
    for (int i = 1; i <= n; i++) {
        int root = findSet(i);
        largestSetSize = max(largestSetSize, setSize[root]);
    }
    cout << largestSetSize << endl;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//DETEÇÃO DE CICLOS NUM GRAFO (ROUND TRIP)
vi temp;
bool flag = false;
stack<int> s;
int sv,ev;

bool DFS(vector<bool>&check,vector<vector<int>> &adj,vi &parent,int v,int p){
  check[v] = true;
  parent[v] = p;
  for (auto i : adj[v])
  {
      if (i == p)continue;
      if(check[i]){
          sv = i;
          ev = v;
          return true;
      }
      if (!check[i])
          if (DFS(check,adj,parent,i,v))
              return true;
  }
  return false;
}

bool visit_all(int n,vector<bool>&check,vector<vector<int>> &adj,vi &parent,int p){
  for(int i = 1; i <= n ; i++){
      if (!check[i]){
          if (DFS(check,adj,parent,i,-1)) return true;
      }
```

```
    }
    return false;
}

void solve() {
    int n, m;
    cin >> n >> m;
    vvi adj(n + 1);
    vector<bool> check(n + 1, false);
    vi parent(n+1);
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    parent[1] = -1;
    if(visit_all(n,check, adj, parent,-1)){
        int a = ev;
        while (a != sv){
            s.push(a);
            a = parent[a];
        }
        cout << s.size() + 2 << endl;
        cout << sv;
        while(!s.empty()){
            cout << " " << s.top();
            s.pop();
        }
        cout << " " << sv << endl;
    }
    else
        cout << "IMPOSSIBLE" << "\n";
}
```

//////////////PROBLEMA DOS DADOS//////////////////////

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

const int MOD = 1000000007;
const int MAX_N = 31;
const int MAX_K = 601;

int memo[MAX_N][MAX_K];

int rec(int N, int M, int K, int sum) {
    if (N == 0) {
```

```
        return sum >= K;
    }

    if (memo[N][sum] != -1) {
        return memo[N][sum];
    }

    int result = 0;
    for (int i = 1; i <= M; i++) {
        result += rec(N - 1, M, K, sum + i);
        result %= MOD;
    }

    memo[N][sum] = result;
    return result;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t;
    cin >> t;

    while (t--) {
        int N, M, K;
        cin >> N >> M >> K;

        memset(memo, -1, sizeof(memo));

        int result = rec(N, M, K, 0);
        cout << result << endl;
    }

    return 0;
}
```

FLOOD FILL

```
int n, m, visited[MAX_N][MAX_N];
char grid[MAX_N][MAX_N];
```

```
const int di[] = {1, 0, -1, 0};
const int dj[] = {0, -1, 0, 1};

bool valid(int i, int j) {
	return i >= 0 && j >= 0 && i < n && j < m && grid[i][j] == '.';
}

void dfs(int i, int j) {
	visited[i][j] = 1;
	for (int k = 0; k < 4; k++) {
		int ni = i + di[k], nj = j + dj[k];
		if (valid(ni, nj) && !visited[ni][nj]) {
			dfs(ni, nj);
		}
	}
}

void solve() {
	cin >> n >> m;
	for (int i = 0; i < n; i++)
		for (int j = 0; j < m; j++)
			cin >> grid[i][j];

	int ans = 0;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < m; j++) {
			if (valid(i, j) && !visited[i][j]) {
				dfs(i, j);
				ans++;
			}
		}
	}
	cout << ans << "\n";
}
```

6.3.2 Longest Palindrome

```
memset(DP, 0, sizeof(DP));
int len = strlen(str), i, j;
for(i = 0; i < len; i++) {
    for(j = 0; j + i < len; j++) {
        if(str[j] == str[i+j]) {
            DP[j][j+i] = DP[j+1][j+i-1] + (i == 0 ? 1 : 2);
        } else {
            DP[j][j+i] = max(DP[j+1][j+i], DP[j][j+i-1]);
        }
    }
}
printf("%d\n", DP[0][len-1]);
//the reconstruction is equal to the LCS
```

6.3.1 Longest Common Subsequence

```
void lcs( string X, string Y, int m, int n ){
   int L[m+1][n+1];
   for (int i=0; i<=m; i++){
     for (int j=0; j<=n; j++){
       if (i == 0 || j == 0)
         L[i][j] = 0;
       else if (X[i-1] == Y[j-1])
         L[i][j] = L[i-1][j-1] + 1;
       else
         L[i][j] = max(L[i-1][j], L[i][j-1]);
     }
   }

   int size_common = L[m][n];
   char lcs[size_common+1];
   lcs[size_common] = ''; // Set the terminating character

   int i = m, j = n;
   while (i > 0 && j > 0){
      if (X[i-1] == Y[j-1]){
          lcs[size_common-1] = X[i-1]; // Put current character in result
          i--; j--; size_common--;   // reduce values of i, j and index
      }
      else if (L[i-1][j] > L[i][j-1])
         i--;
      else
         j--;
   }
   cout << "LCS␣of␣" << X << "␣and␣" << Y << "␣is␣" << lcs;
}
```

```
LIS[1] = 1
for i = 2 to n do
 LIS[i] = 0
 for j = 1 to i-1 do
  if sj < si and LIS[j] > LIS[i] then
   LIS[i] = LIS[j]
 LIS[i] = LIS[i] + 1
return max(LIS [1],..., LIS[n])
```

2.10 Monotomic Paths
-> Number of paths from (0,0) to (n, m)
Function count(n,m)
for i = 1 to n do {1st base case}
T[i ,1] = 1
for j = 1 to m do {2nd base case}
T[1, j] = 1
for i = 2 to n do
for j = 2 to m do
T[i , j] = T[i−1,j] + T[i , j−1])
return T[n,m]

```
// Given a list of jobs with start time, end time, and profit, find the maximum profit

void solve() {
	int n; cin >> n;
	ar<ll,3> arr[n];
	for (int i = 0; i < n; i++) cin >> arr[i][1] >> arr[i][0] >> arr[i][2];
	sort(arr, arr + n); // sort based on end time
	ll dp[n];
	dp[0] = arr[0][2];
	for (int i = 1; i < n; i++) {
	int k = lower_bound(arr, arr + n, ar<ll,3>{arr[i][1], 0, 0}) - arr - 1;
	if (k >= 0)
	dp[i] = max(dp[i - 1], dp[k] + arr[i][2]);
	else
	dp[i] = max(dp[i - 1], arr[i][2]);

	}
	cout << dp[n - 1] << "\n";
}
// Find the sum of the digits of the numbers between a and b (0 <= a <= b <= 1e9)
vector<int> num;
ll dp[10][9 * 10][2];

// dp[pos][sum][flag]
// pos = current position, starting from the left (0-index)
// sum = sum of all digits till the given position
// flag = the number we are building has already become smaller than b? [0 = no, 1 = yes]

ll memo(int pos, int sum, int flag) {
	if (pos == num.size()) return sum;
	if (dp[pos][sum][flag] != -1) return dp[pos][sum][flag];

	ll res = 0;
	int lmt = (flag) ? 9 : num[pos];
	for (int i = 0; i <= lmt; i++) {
	int next_flag = (i < lmt) ? 1 : flag;
	res += memo(pos + 1, sum + i, next_flag);
	}
	return dp[pos][sum][flag] = res;
}

ll calc(int n) {
	num.clear();
	while (n) {
	num.push_back(n % 10);
	n /= 10;
	}
	reverse(num.begin(), num.end());
	memset(dp, -1, sizeof dp);
	return memo(0, 0, 0);
}
// Given a length x and n cutting points, find the minimum cost perform all n cuts
```

```
// Cost of a cut is equal to the length of the current stick
int opt[MAX_N][MAX_N];

void solve() {
	while (true) {
	int x; cin >> x;
	if (!x) return;

	int n; cin >> n;
	int arr[n + 2];
	// adding the beginning point and the ending point
	arr[0] = 0; arr[n + 1] = x;
	for (int i = 1; i <= n; i++) cin >> arr[i];
	vector<vector<int>> dp(n + 2, vector<int>(n + 2, INF));
	for (int i = 0; i < n + 1; i++) {
	dp[i][i + 1] = 0;
	opt[i][i + 1] = i;
	}
	// range dp
	for (int i = n + 1; i >= 0; i--) {
	for (int j = i; j <= n + 1; j++) {
	for (int k = i + 1; k < j; k++) {
		if (dp[i][j] > dp[i][k] + dp[k][j] + arr[j] - arr[i]) {
		dp[i][j] = dp[i][k] + dp[k][j] + arr[j] - arr[i];
		}
	}
	}
	}
	cout << "The minimum cutting is " << dp[0][n + 1] << ".\n";
	}
}


//////////////////////////////////////////BRIDGES AND APS////////////////////////////////////////////////////////////////
// Given an undirected graph, find all bridges and articulation points
// Perform a DFS to form a DFS spanning tree
// u-v is a bridge <=> there is a back edge connecting a descendant of v and an ancestor of u
// Time complexity: O(n + m)

#include <bits/stdc++.h>

using namespace std;

#define ar array
#define ll long long

const int MAX_N = 1e5 + 1;
const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LINF = 1e18;

// dfs_num[u] stores the iteration counter when DFS visits node u for the 1st time
```

```
// dfs_low[u] stores the lowest dfs_num reachable from the current DFS spanning subtree of node u
// dfs_low[u] can only be made smaller if there is a back edge to form a cycle and v is currently visited

int n, m, dfsCounter;
int dfs_num[MAX_N], dfs_low[MAX_N], visited[MAX_N];
vector<int> adj[MAX_N];

void dfs(int u, int p = -1) {
	dfs_num[u] = dfs_low[u] = dfsCounter++;
	visited[u] = 1;
	int num_child = 0;
	for (int v : adj[u]) {
		if (v == p) continue;
		// back edge
		if (visited[v]) dfs_low[u] = min(dfs_low[u], dfs_num[v]);
		// tree edge
		else {
			dfs(v, u);
			dfs_low[u] = min(dfs_low[u], dfs_low[v]);
			num_child++;
			if (dfs_low[v] > dfs_num[u])
				cout << "Edge " << u << "-" << v << " is a bridge\n";
			if (dfs_low[v] >= dfs_num[u] && p != -1)
				cout << "Node " << u << " is an articulation point\n";
		}
	}
	// special case: the root node is an articulation point if it has more than 1 child
	if (p == -1 && num_child > 1)
		cout << "Node " << u << " is an articulation point\n";
}

void solve() {
	cin >> n >> m;
	for (int i = 0; i < m; i++) {
		int u, v; cin >> u >> v;
		adj[u].push_back(v);
		adj[v].push_back(u);
	}
	memset(dfs_low, -1, sizeof dfs_low);
	memset(dfs_num, -1, sizeof dfs_num);
	for (int i = 1; i <= n; i++)
		if (!visited[i])
			dfs(i);

	/*
	Example input:
	12 16
	1 3
	3 5
	5 7
	3 7
	3 8
```

```
        1 5
        1 6
        3 6
        6 2
        2 8
        2 4
        4 10
        1 9
        9 11
        11 12
        9 12

        Expected output:
        Edge 4-10 is a bridge
        Node 4 is an articulation point
        Edge 2-4 is a bridge
        Node 2 is an articulation point
        Node 9 is an articulation point
        Edge 1-9 is a bridge
        Node 1 is an articulation point
        */
}

int main() {
        ios_base::sync_with_stdio(0);
        cin.tie(0); cout.tie(0);
        // freopen("input.txt", "r", stdin);
        // freopen("output.txt", "w", stdout);

        int tc; tc = 1;
        for (int t = 1; t <= tc; t++) {
        // cout << "Case #" << t  << ": ";
        solve();
        }
}
```