Demo Praktikum 4 Struktur Data

Muhammad Alfan Mahdi – 5025221275

1. **AHOCK**

   Input:

   6
   1 2
   1 5
   1 6
   2 5
   3 4
   3 5
   4 5
   Yee!

   Output:

   6 Targetnya
   5 Paling Bahaya

   Penjelasan:

   Output merupakan node target serang dengan neighbor paling sedikit, dan node paling berbahaya dengan neighbor paling banyak.

   Cara Kerja Code:

   - Fungsi findLeastNeighbors:
     - Fungsi ini mencari vertex dengan jumlah tetangga paling sedikit dalam graf.
     - Fungsi ini menerima parameter adj yang merupakan adjacency list graf dan totalVertices yang merupakan total jumlah vertex dalam graf.
     - Fungsi ini melakukan iterasi untuk setiap vertex dalam graf dan membandingkan jumlah tetangganya dengan variabel leastNeighbors.
     - Jika jumlah tetangga kurang dari atau sama dengan leastNeighbors, maka leastNeighbors diperbarui dan vertexWithLeastNeighbors diatur menjadi vertex saat ini.
     - Pada akhirnya, fungsi ini mencetak vertex dengan jumlah tetangga paling sedikit beserta pesan "Targetnya".
   - Fungsi findMostNeighbors:
     - Fungsi ini mencari vertex dengan jumlah tetangga paling banyak dalam graf.
     - Fungsi ini menerima parameter adj yang merupakan adjacency list graf dan totalVertices yang merupakan total jumlah vertex dalam graf.
     - Fungsi ini melakukan iterasi untuk setiap vertex dalam graf dan membandingkan jumlah tetangganya dengan variabel mostNeighbors.
     - Jika jumlah tetangga lebih besar dari mostNeighbors, maka mostNeighbors diperbarui dan vertexWithMostNeighbors diatur menjadi vertex saat ini.
     - Jika jumlah tetangga sama dengan mostNeighbors, maka iterasi dilanjutkan tanpa melakukan perubahan.

    - Pada akhirnya, fungsi ini mencetak vertex dengan jumlah tetangga paling banyak beserta pesan "Paling Bahaya".
- Fungsi main:
    - Fungsi ini merupakan fungsi utama dari program.
    - Di dalam fungsi main, terdapat deklarasi variabel n, x, y, dan input.
    - Program membaca nilai n dari input yang merupakan jumlah vertex dalam graf.
    - Selanjutnya, program mendeklarasikan array vektor adj dengan ukuran n untuk merepresentasikan adjacency list graf.
    - Program membaca input dengan menggunakan loop do-while hingga input adalah "Yee!".
    - Pada setiap iterasi, program membaca input dan jika input adalah "Yee!", loop akan berhenti. Jika tidak, program akan mengambil nilai x dan y dari input untuk menambahkan edge antara vertex x-1 dan y-1 (indeks dimulai dari 0) menggunakan fungsi addEdge.
    - Setelah itu, program memanggil fungsi findLeastNeighbors untuk mencari vertex dengan jumlah tetangga paling sedikit dan findMostNeighbors untuk mencari vertex dengan jumlah tetangga paling banyak.
    - Akhirnya, program mengembalikan nilai 0 dan berakhir.

Source Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <string>

using namespace std;

void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

void dfs(vector<long>& result, long start, vector<int>* adj, int
totalVertices)
{
    vector<bool> visited(totalVertices, false);
    stack<long> st;

    st.push(start);
    visited[start] = true;

    while (!st.empty()) {
        long temp = st.top();
        st.pop();
```

```
        result.push_back(temp);

        for (auto it : adj[temp]) {
            if (!visited[it]) {
                st.push(it);
                visited[it] = true;
            }
        }
    }
}

void findLeastNeighbors(vector<int>* adj, int totalVertices)
{
    int leastNeighbors = 1000000000;
    int vertexWithLeastNeighbors = 0;

    for (int i = 0; i < totalVertices; i++) {
        if (adj[i].size() <= leastNeighbors) {
            leastNeighbors = adj[i].size();
            vertexWithLeastNeighbors = i;
        }
    }
    cout << vertexWithLeastNeighbors + 1 << " Targetnya" << endl;
}

void findMostNeighbors(vector<int>* adj, int totalVertices)
{
    int mostNeighbors = 0;
    int vertexWithMostNeighbors = 0;

    for (int i = 0; i < totalVertices; i++) {
        if (adj[i].size() > mostNeighbors) {
            mostNeighbors = adj[i].size();
            vertexWithMostNeighbors = i;
        } else if (adj[i].size() == mostNeighbors) {
            continue;
        }
    }
    cout << vertexWithMostNeighbors + 1 << " Paling Bahaya" << endl;
}

int main() {
    int n, x, y;
    string input;

    cin >> n;
```

```
    vector<int> adj[n];

    do {
        cin >> input;
        if(input == "Yee!") {
            break;
        }
        else {
            x = stoi(input.substr(0, input.find(' ')));
            y = stoi(input.substr(input.find(' ') + 1));
            addEdge(adj, x - 1, y - 1); // Adjust the indices to start from 0
        }
    } while(input != "Yee!");

    findLeastNeighbors(adj, n);
    findMostNeighbors(adj, n);

    return 0;
}
```

2. CN

Input:

8

12 3 5 11 15 4 8 6

7

Output:

3

Penjelasan:

Output merupakan jumlah komponen terhubung dalam graf.

Cara Kerja Kode:

- Fungsi addEdge:
    - Fungsi ini digunakan untuk menambahkan edge antara dua vertex dalam graf.
    - Fungsi ini menerima adjacency list adj[], dan vertex u dan v yang akan dihubungkan.
    - Edge ditambahkan dengan memasukkan vertex v ke dalam list adjacency vertex u, dan sebaliknya.
- Fungsi dfs:
    - Fungsi ini merupakan implementasi Depth-First Search (DFS) dalam graf.
    - Fungsi ini menerima vektor result yang digunakan untuk menyimpan hasil DFS, vertex start sebagai titik awal DFS, adjacency list adj yang merepresentasikan graf, dan totalVertices yang merupakan jumlah total vertex dalam graf.
    - Fungsi ini menggunakan stack st dan vektor visited untuk melacak vertex yang telah dikunjungi.

- Pada awalnya, vertex start dimasukkan ke dalam stack dan ditandai sebagai telah dikunjungi.
- Selama stack tidak kosong, vertex teratas diambil dari stack, ditambahkan ke result, dan dilakukan iterasi pada semua tetangga vertex tersebut.
- Jika tetangga belum dikunjungi, tetangga tersebut dimasukkan ke dalam stack dan ditandai sebagai telah dikunjungi.
- Proses ini berlanjut hingga semua vertex terhubung telah dikunjungi.

- Fungsi main:
  - Fungsi utama yang menjalankan algoritma untuk menghitung jumlah komponen terhubung dalam graf.
  - Membaca input n yang merupakan jumlah vertex dalam graf.
  - Membuat vektor list untuk menyimpan nilai-nilai vertex.
  - Membaca input x sebanyak n kali dan menyimpannya ke dalam vektor list.
  - Membaca input c yang digunakan dalam perhitungan pengecekan edge.
  - Membuat array adj sebagai adjacency list dengan ukuran n.
  - Melakukan nested loop untuk memeriksa pasangan vertex dalam list (indeks i dan j).
  - Jika (list[i] xor list[j]) % c == 0, maka addEdge(adj, i, j) dipanggil untuk menambahkan edge antara vertex i dan j.
  - Membuat vektor result untuk menyimpan hasil DFS dan variabel components untuk menghitung jumlah komponen terhubung.
  - Melakukan loop untuk setiap vertex dalam graf.
  - Jika result masih kosong, maka menjalankan DFS dari vertex tersebut dan menambahkan 1 ke components.
  - Jika result tidak kosong dan vertex tidak ada dalam result, maka menjalankan DFS dari vertex tersebut dan menambahkan 1 ke components.
  - Cetak nilai components, yang merupakan jumlah komponen terhubung dalam graf.

Source Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <algorithm>

using namespace std;

void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

```
void dfs(vector<long>& result, long start, vector<int>* adj, int
totalVertices)
{
    vector<bool> visited(totalVertices, false);
    stack<long> st;

    st.push(start);
    visited[start] = true;

    while (!st.empty()) {
        long temp = st.top();
        st.pop();

        result.push_back(temp);

        for (auto it : adj[temp]) {
            if (!visited[it]) {
                st.push(it);
                visited[it] = true;
            }
        }
    }
}

int main()
{
    int n, x, c;

    cin >> n;

    vector<int> list;
    vector<int> adj[n];

    for (int i = 0; i < n; i++) {
        cin >> x;
        list.push_back(x);
    }

    cin >> c;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if ((list[i] xor list[j]) % c == 0) {
                addEdge(adj, i, j);
            }
        }
    }
```

```
    vector<long> result;
    int components = 0;

    for (int i = 0; i < n; i++) {
        if (result.empty()) {
            dfs(result, i, adj, n);
            components++;
        } else if (find(result.begin(), result.end(), i) == result.end()) {
            dfs(result, i, adj, n);
            components++;
        }
    }

    cout << components << endl;

    return 0;
}
```

3. **PBJ**

Input:

5 4
1 2
2 3
3 1
4 5

Output:

1

Penjelasan:

Output merupakan banyak edge yang dibutuhkan agar semua vertex dapat terhubung menjadi satu connected graf.

Cara Kerja Code:

- Fungsi addEdge:
    - Fungsi ini sama dengan yang telah dijelaskan sebelumnya, digunakan untuk menambahkan edge antara dua vertex dalam graf.
    - Edge ditambahkan dengan memasukkan vertex v ke dalam list adjacency vertex u, dan sebaliknya.
- Fungsi dfs:
    - Fungsi ini juga sama dengan yang telah dijelaskan sebelumnya, merupakan implementasi Depth-First Search (DFS) dalam graf.
    - Perbedaannya, fungsi ini menerima argumen tambahan visited yang merupakan vektor boolean untuk melacak vertex yang telah dikunjungi.
- Fungsi countEdgesToAddForConnectedLine:
    - Fungsi ini menghitung jumlah edge yang perlu ditambahkan agar semua vertex dalam graf terhubung dalam satu garis.
    - Fungsi ini menerima adjacency list adj, dan jumlah total vertex totalVertices.

- Fungsi ini menggunakan DFS untuk menghitung jumlah komponen terhubung dalam graf.
- Pertama, inisialisasi vektor visited dengan nilai false untuk semua vertex.
- Kemudian, lakukan DFS untuk setiap vertex yang belum dikunjungi. Setiap kali DFS dipanggil, tambahkan 1 ke components dan catat hasil DFS dalam vektor result.
- Setelah itu, cek nilai components. Jika components adalah 1, artinya semua vertex sudah terhubung dan hanya dibutuhkan totalVertices - 1 edge untuk membentuk garis. Jika components lebih dari 1, artinya terdapat lebih dari satu komponen terhubung, maka jumlah edge yang dibutuhkan adalah components - 1.
- Kembalikan nilai edgesNeeded yang merupakan jumlah edge yang perlu ditambahkan.

- Fungsi main:
  - Fungsi utama yang menjalankan algoritma untuk menghitung jumlah edge yang perlu ditambahkan.
  - Membaca input v dan n yang merupakan jumlah vertex dan jumlah edge dalam graf.
  - Membuat array adj sebagai adjacency list dengan ukuran v + 1.
  - Melakukan loop sebanyak n kali untuk membaca pasangan vertex a dan b yang membentuk edge, dan memanggil fungsi addEdge untuk menambahkan edge tersebut ke dalam adjacency list adj.
  - Cetak hasil dari fungsi countEdgesToAddForConnectedLine yang menghitung jumlah edge yang perlu ditambahkan untuk menghubungkan semua vertex dalam satu garis.

Source Code:

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

void dfs(vector<long>& result, long start, vector<int>* adj, vector<bool>&
visited)
{
    stack<long> st;
```

```
    st.push(start);
    visited[start] = true;

    while (!st.empty()) {
        long temp = st.top();
        st.pop();

        result.push_back(temp);

        for (auto it : adj[temp]) {
            if (!visited[it]) {
                st.push(it);
                visited[it] = true;
            }
        }
    }
}

int countEdgesToAddForConnectedLine(vector<int>* adj, int totalVertices)
{
    vector<bool> visited(totalVertices + 1, false);
    vector<long> result;

    int components = 0;
    int edgesNeeded = 0;

    for (int i = 1; i <= totalVertices; i++) {
        if (!visited[i]) {
            components++;
            dfs(result, i, adj, visited);
        }
    }

    if (components == 1)
        edgesNeeded = totalVertices - 1;
    else
        edgesNeeded = components - 1;

    return edgesNeeded;
}

int main()
{
    int v, n, a, b;

    cin >> v >> n;

    vector<int> adj[v + 1];
```

```
    for (int i = 0; i < n; i++) {
        cin >> a >> b;
        addEdge(adj, a, b);
    }

    cout << countEdgesToAddForConnectedLine(adj, v);

    return 0;
}
```