LAPORAN PRAKTIKUM STRUKTUR DATA

PERTEMUAN 10

MULTI LINKED LIST

Nama :

Reyner Atira Prasetyo (2311104057)

S1SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

a. Memahami konsep penggunaan fungsi rekursif.
b. Mengimplementasikan bentuk-bentuk fungsi rekursif.
c. Mengaplikasikan struktur data tree dalam sebuah kasus pemrograman.
d. Mengimplementasikan struktur data tree, khususnya Binary Tree.
e. Mengimplementasikan struktur data tree, khususnya Binary Tree.

TOOL

1. Visual Studio Code

2. GCC

II. DASAR TEORI

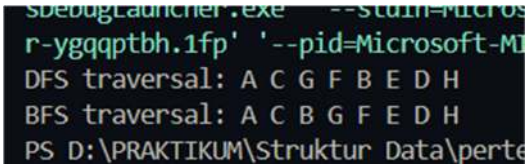
Multi List merupakan sekumpulan list yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam multi link list dapat membentuk list sendiri.Biasanya ada yang bersifat sebagai list induk dan list anak**.**

III. GUIDED

1. GuidedGraph.cpp

```cpp
#include <iostream>
#include <queue>

using namespace std;

struct ElmNode;

struct ElmEdge {
    ElmNode *Node;
    ElmEdge *Next;
};

struct ElmNode {
    char Info;
    bool visited;
    ElmEdge *FirstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *First;
};

void CreateGraph(Graph &G) {
    G.First = NULL;
}

void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode;
    newNode->Info = X;
    newNode->visited = false;
    newNode->FirstEdge = NULL;
    newNode->Next = NULL;

    if (G.First == NULL) {
        G.First = newNode;
    } else {
        ElmNode *temp = G.First;
        while (temp->Next != NULL) {
            temp = temp->Next;
        }
        temp->Next = newNode;
    }
}

void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge;
    newEdge->Node = N2;
    newEdge->Next = N1->FirstEdge;
    N1->FirstEdge = newEdge;
}

void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.First;
    while (temp != NULL) {
        cout << temp->Info << " ";
        temp = temp->Next;
    }
    cout << endl;
}

void ResetVisited(Graph &G) {
    ElmNode *temp = G.First;
    while (temp != NULL) {
        temp->visited = false;
        temp = temp->Next;
    }
}

void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return;
    }
    N->visited = true;
    cout << N->Info << " ";
    ElmEdge *edge = N->FirstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node);
        }
        edge = edge->Next;
    }
}

void PrintBFS(Graph G, ElmNode *N) {
    queue<ElmNode*> q;
    q.push(N);
    N->visited = true;

    while (!q.empty()) {
        ElmNode *current = q.front();
        q.pop();
        cout << current->Info << " ";

        ElmEdge *edge = current->FirstEdge;
        while (edge != NULL) {
            if (!edge->Node->visited) {
                edge->Node->visited = true;
                q.push(edge->Node);
            }
            edge = edge->Next;
        }
    }
}

int main() {
    Graph G;
    CreateGraph(G);

    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    ElmNode *A = G.First;
    ElmNode *B = A->Next;
    ElmNode *C = B->Next;
    ElmNode *D = C->Next;
    ElmNode *E = D->Next;
    ElmNode *F = E->Next;
    ElmNode *G1 = F->Next;
    ElmNode *H = G1->Next;

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(D, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(G1, H);

    cout << "DFS traversal: ";
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;

    cout << "BFS traversal: ";
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}
```

Hasil Run :



IV. UNGUIDED

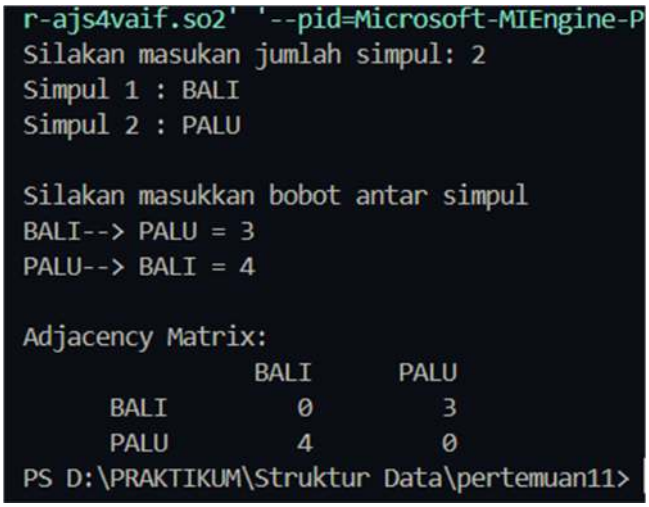
1. Unguided1.cpp

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;

int main() {
    int jumlahSimpul;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> jumlahSimpul;

    vector<string> namaSimpul(jumlahSimpul);
    for (int i = 0; i < jumlahSimpul; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> namaSimpul[i];
    }

    // Membuat adjacency matrix
    vector<vector<int>> graph(jumlahSimpul, vector<int>(jumlahSimpul, 0));
    cout << "\nSilakan masukkan bobot antar simpul\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            if (i == j) {
                graph[i][j] = 0; // Bobot dari simpul ke dirinya sendiri 0
            } else {
                cout << namaSimpul[i] << "--> " << namaSimpul[j] << " = ";
                cin >> graph[i][j];
            }
        }
    }

    // Menampilkan adjacency matrix
    cout << "\nAdjacency Matrix:\n";
    cout << setw(10) << " ";
    for (const string& nama : namaSimpul) {
        cout << setw(10) << nama;
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << namaSimpul[i];
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << setw(10) << graph[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Penjelasan :

- Variabel jumlahSimpul menerima input jumlah simpul dari pengguna.
- vector<string>: Digunakan untuk menyimpan nama dari setiap simpul (misalnya: nama kota).

- Loop akan meminta pengguna untuk memasukkan nama setiap simpul sebanyak jumlahSimpul kali.
- vector<vector<int>>: Matriks dua dimensi untuk menyimpan bobot (jarak) antar simpul.
- Inisialisasi matriks dengan ukuran jumlahSimpul x jumlahSimpul dan diisi dengan 0.
- graph[i][j]: Menyimpan bobot (jarak) dari simpul i ke simpul j.
- Jika i == j (sama), maka bobot otomatis 0 (tidak ada jarak ke diri sendiri).
- Jika tidak, pengguna diminta memasukkan bobot untuk koneksi antar simpul.



2. Unguided2.cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int jumlahSimpul, jumlahSisi;

    // Input jumlah simpul dan sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan jumlah sisi: ";
    cin >> jumlahSisi;

    // Inisialisasi adjacency matrix dengan nol
    vector<vector<int>> adjacencyMatrix(jumlahSimpul, vector<int>(jumlahSimpul, 0));

    // Input pasangan simpul yang terhubung oleh sisi
    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < jumlahSisi; i++) {
        int simpul1, simpul2;
        cin >> simpul1 >> simpul2;

        // Karena graf tidak berarah, update kedua arah
        adjacencyMatrix[simpul1 - 1][simpul2 - 1] = 1;
        adjacencyMatrix[simpul2 - 1][simpul1 - 1] = 1;
    }

    // Menampilkan adjacency matrix
    cout << "\nAdjacency Matrix:\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << adjacencyMatrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

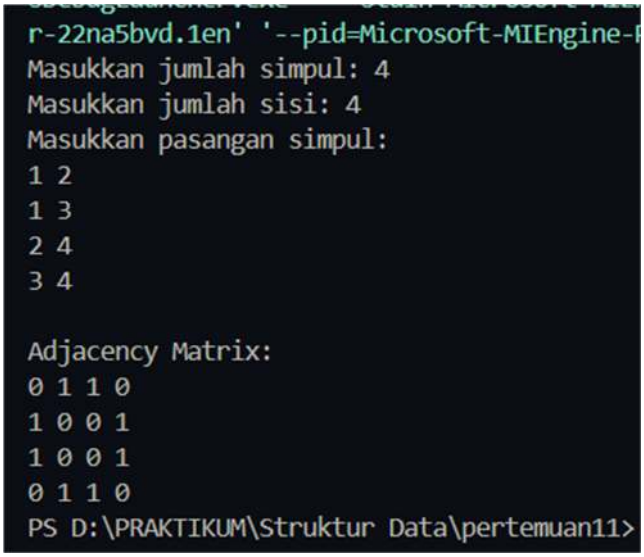
Penjelasan :

- Input jumlah simpul dan sisi menggunakan cin : Meminta jumlah simpul (jumlahSimpul) dan sisi (jumlahSisi) dari pengguna.
- Matriks 2D (adjacencyMatrix) berukuran jumlahSimpul x jumlahSimpul diisi dengan nol untuk merepresentasikan tidak adanya koneksi awal antar simpul.
- Untuk setiap sisi, pengguna memasukkan pasangan simpul yang terhubung. Matriks diperbarui di kedua arah karena graf tidak berarah:

  adjacencyMatrix[simpul1 - 1][simpul2 - 1] = 1;

adjacencyMatrix[simpul2 - 1][simpul1 - 1] = 1;

- Matriks ditampilkan, menunjukkan koneksi antar simpul. Nilai 1 menandakan adanya koneksi, sedangkan 0 menandakan tidak ada koneksi.



V. KESIMPULAN

Praktikum struktur data graph memperkenalkan konsep dan implementasi graph sebagai struktur data yang merepresentasikan hubungan antar objek melalui simpul (nodes) dan sisi (edges). Graph dapat direpresentasikan dengan matriks adjacency atau list adjacency, dengan kelebihan masing-masing dalam efisiensi memori dan waktu.

Algoritma traversal seperti DFS dan BFS terbukti efektif dalam menyelesaikan berbagai masalah, termasuk pencarian jalur dan deteksi siklus. Selain itu, graph memiliki banyak aplikasi praktis, seperti pencarian rute, analisis jaringan, dan penjadwalan. Praktikum ini membantu memahami pentingnya struktur data graph dan algoritmanya dalam menyelesaikan masalah yang kompleks.