LAPORAN PRAKTIKUM

Modul 14

Graph

Disusun Oleh:

Yogi Hafidh Maulana - 2211104061

SE06-02

Asisten Praktikum :

Aldi Putra Andini

Nur Hidayah

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

A. Tujuan

- Memahami konsep dasar graf (graph) sebagai himpunan simpul (nodes) dan sisi (edges).
- Mempelajari representasi graf berarah dan tidak berarah menggunakan pointer.
- Mengimplementasikan traversal graf dengan metode BFS dan DFS.
- Membuat ADT graf untuk operasi dasar seperti membuat graf, menambah simpul, menghubungkan simpul, dan mencetak graf.
- Menerapkan graf dalam kasus praktis seperti topological sorting untuk menyusun keterurutan data.

B. Landasan Teori

Graf adalah struktur data non-linear yang terdiri dari himpunan simpul dan sisi yang menghubungkan simpul-simpul tersebut. Graf digunakan untuk merepresentasikan hubungan atau koneksi antara objek. Graf dapat dibedakan menjadi dua jenis utama, yaitu graf berarah dan graf tidak berarah. Dalam graf berarah, setiap sisi memiliki arah, sedangkan dalam graf tidak berarah, sisi hanya menunjukkan keterhubungan tanpa arah tertentu.

Graf sering digunakan dalam berbagai aplikasi, seperti jaringan komunikasi, peta jalan, analisis hubungan sosial, dan penyelesaian permasalahan seperti pencarian jalur terpendek atau pengurutan tugas berdasarkan ketergantungan. Representasi graf dapat dilakukan dengan berbagai cara, termasuk menggunakan matriks ketetanggaan, di mana hubungan antar simpul direpresentasikan dalam bentuk matriks 2 dimensi, atau daftar ketetanggaan, yang lebih efisien untuk graf dengan sedikit sisi.

Traversal graf adalah metode untuk menjelajahi simpul-simpul dalam graf. Dua algoritma traversal yang umum digunakan adalah Depth First Search (DFS) dan Breadth First Search (BFS). DFS menjelajahi graf secara mendalam melalui rekursi atau tumpukan, sementara BFS menjelajahi graf secara melebar menggunakan antrian. Selain traversal, graf juga dapat diterapkan untuk menyelesaikan masalah keterurutan seperti topological sorting, yang menentukan urutan linier dari data yang memiliki ketergantungan parsial.

Dalam implementasi praktis, graf sering direpresentasikan menggunakan pointer dinamis untuk mendukung fleksibilitas dalam manipulasi struktur data, memungkinkan efisiensi penyimpanan dan pengelolaan data yang kompleks.

C. Guided

Code:

```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;
};
```

```cpp
void CreateGraph(Graph &G)
{
    G.first = NULL;
}

Tabnine | Edit | Test | Explain | Document | Ask
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)
        {
```

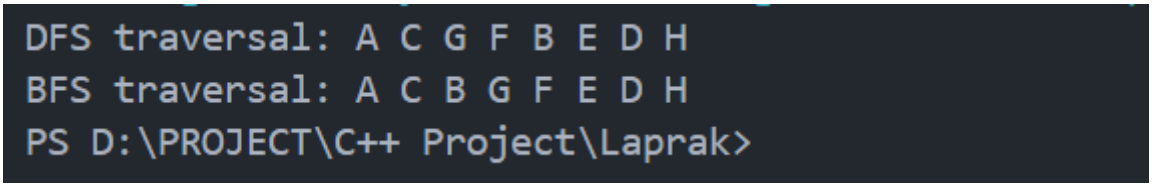
```
            temp = temp->Next;
        }
        temp->Next = newNode;
    }
}

Tabnine | Edit | Test | Explain | Document | Ask
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;
}

Tabnine | Edit | Test | Explain | Document | Ask
void ResetVisited(Graph &G)
{
    ElmNode *temp = G.first;
    while (temp != NULL)
    {
        temp->visited = false;
        temp = temp->Next;
    }
}
```

```
82  void PrintDFS(Graph G, ElmNode *N)
83  {
84      if (N == NULL)
85      {
86          return;
87      }
88      N->visited = true;
89      cout << N->info << " ";
90      ElmEdge *edge = N->firstEdge;
91      while (edge != NULL)
92      {
93          if (!edge->Node->visited)
94          {
95              PrintDFS(G, edge->Node);
96          }
97          edge = edge->Next;
98      }
99  }
100
```

```
101 void PrintBFS(Graph G, ElmNode *N)
102 {
103     queue<ElmNode *> q;
104     q.push(N);
105     N->visited = true;
106
107     while (!q.empty())
108     {
109         ElmNode *current = q.front();
110         q.pop();
111         cout << current->info << " ";
112
113         ElmEdge *edge = current->firstEdge;
114         while (edge != NULL)
115         {
116             if (!edge->Node->visited)
117             {
118                 edge->Node->visited = true;
119                 q.push(edge->Node);
120             }
121             edge = edge->Next;
122         }
123     }
124 }
```

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

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

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

    return 0;
}
```

Output:



Deskripsi Program:

Kode guided ini menjelaskan implementasi graf menggunakan linked list untuk merepresentasikan simpul (ElmNode) dan sisi (ElmEdge) dalam struktur graf yang dinamis. Graf ini dibangun dengan fungsi-fungsi untuk menambahkan simpul ke graf dan menghubungkan simpul-simpul tersebut melalui sisi, sehingga membentuk relasi antar-simpul. Dua algoritma traversal yang diterapkan adalah DFS (Depth-First Search) dan BFS (Breadth-First Search). DFS menggunakan pendekatan rekursif untuk menjelajahi graf secara mendalam, bergerak dari simpul ke simpul tetangganya sebelum kembali, sedangkan BFS menggunakan struktur queue untuk menjelajahi graf secara melebar, menjelajahi semua tetangga pada satu tingkat sebelum melanjutkan ke tingkat berikutnya. Simpul yang telah dikunjungi ditandai dengan atribut visited untuk menghindari traversal berulang. Dalam contoh program ini, graf dengan simpul A hingga H dibangun dan hasil traversal menunjukkan bagaimana eksplorasi berbeda dilakukan: DFS menghasilkan urutan A B D H E C F G, sedangkan BFS menghasilkan urutan A B C D E F G H. Kode ini menggambarkan dasar-dasar representasi dan penjelajahan graf dalam pemrograman menggunakan struktur data pointer.

D. Unguided

1) Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

Code:

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

Tabnine | Edit | Test | Explain | Document | Ask
void printMatrix(const vector<vector<int>> &matrix, const vector<string> &nodes)
{
    int n = nodes.size();

    cout << setw(8) << "";
    for (int i = 0; i < n; i++)
    {
        cout << setw(8) << nodes[i];
    }
    cout << endl;

    for (int i = 0; i < n; i++)
    {
        cout << setw(8) << nodes[i];
        for (int j = 0; j < n; j++)
        {
            cout << setw(8) << matrix[i][j];
        }
        cout << endl;
    }
}
int main()
{
    int numNodes;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> numNodes;

    vector<string> nodes(numNodes);
    cout << "Silakan masukan nama simpul:\n";
    for (int i = 0; i < numNodes; i++)
    {
        cout << "Simpul " << i + 1 << ": ";
        cin >> nodes[i];
    }

    vector<vector<int>> weights(numNodes, vector<int>(numNodes, 0));

    cout << "Silakan masukkan bobot antar simpul:\n";
    for (int i = 0; i < numNodes; i++)
    {
        for (int j = 0; j < numNodes; j++)
        {
            if (i != j || (i == j && weights[i][j] == 0))
            {
                cout << nodes[i] << " --> " << nodes[j] << " = ";
                cin >> weights[i][j];
            }
        }
```

```
    }

    cout << "\n";
    printMatrix(weights, nodes);

    return 0;
}
```

Output:

```
Silakan masukan jumlah simpul: 2
Silakan masukan nama simpul:
Simpul 1: SUMATRA
Simpul 2: JAWA
Silakan masukkan bobot antar simpul:
SUMATRA --> SUMATRA = 0
SUMATRA --> JAWA = 4
JAWA --> SUMATRA = 5
JAWA --> JAWA = 2

          SUMATRA    JAWA
 SUMATRA        0       4
    JAWA        5       2
```

Deskripsi Code:
Kode soal unguided nomer satu ini adalah implementasi program untuk membangun dan menampilkan graf berbobot menggunakan matriks ketetanggaan (adjacency matrix). Graf direpresentasikan sebagai kumpulan simpul (nodes) yang saling terhubung oleh bobot (weights), di mana hubungan antar-simpul direkam dalam bentuk matriks dua dimensi. Pengguna diminta memasukkan jumlah simpul, nama setiap simpul, dan bobot antar simpul, termasuk hubungan dari simpul ke dirinya sendiri (biasanya 0). Matriks bobot diinisialisasi dengan nilai 0 dan diperbarui berdasarkan input pengguna. Fungsi printMatrix digunakan untuk mencetak matriks bobot dengan format yang rapi, menampilkan hubungan antar simpul dengan nama simpul pada baris dan kolom. Program ini menggambarkan bagaimana graf berbobot dapat direpresentasikan secara efisien menggunakan matriks, yang berguna untuk menyimpan informasi lengkap tentang hubungan antar simpul, cocok untuk analisis seperti pencarian jalur terpendek atau deteksi koneksi antar simpul.

2) Buatlah sebuah program C++ untuk merepresentasikan graf tidak berarah menggunakan adjacency matrix. Program harus dapat:
   - Menerima input jumlah simpul dan jumlah sisi.
   - Menerima input pasangan simpul yang terhubung oleh sisi.
   - Menampilkan adjacency matrix dari graf tersebut.

Code:

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

Tabnine | Edit | Test | Explain | Document | Ask
void printAdjacencyMatrix(const vector<vector<int>> &matrix)
{
    int n = matrix.size();
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

int main()
{
    int numNodes, numEdges;

    cout << "Masukkan jumlah simpul: ";
    cin >> numNodes;
    cout << "Masukkan jumlah sisi: ";
    cin >> numEdges;

    vector<vector<int>> adjacencyMatrix(numNodes, vector<int>(numNodes, 0));

    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < numEdges; i++)
    {
        int node1, node2;
        cin >> node1 >> node2;

        adjacencyMatrix[node1 - 1][node2 - 1] = 1;
        adjacencyMatrix[node2 - 1][node1 - 1] = 1;
    }
```

```
38
39      cout << "\nAdjacency Matrix:\n";
40      printAdjacencyMatrix(adjacencyMatrix);
41
42      return 0;
43  }
```

Output:

```
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul:
1
2
1
3
2
4
3
4

Adjacency Matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
PS D:\PROJECT\C++ Project\Laprak>
```

Deskripsi Code:

Soal kode unguided nomer dua adalah implementasi representasi graf **tidak berarah** menggunakan adjacency matrix. Graf direpresentasikan sebagai matriks dua dimensi berukuran n × n (dengan n adalah jumlah simpul) di mana elemen matrix[i][j]=1matrix[i][j] = 1 menunjukkan adanya sisi antara simpul i+1 dan simpul j+1, sedangkan matrix[i][j]=0matrix[i][j] = 0 menunjukkan tidak adanya hubungan. Program ini dimulai dengan menerima jumlah simpul dan jumlah sisi dari pengguna, kemudian meminta pasangan simpul yang terhubung. Setiap pasangan yang dimasukkan akan memperbarui matriks adjacency dengan menambahkan 1 pada posisi yang sesuai, baik untuk arah node1 ke node2 maupun node2 ke node1, karena graf tidak berarah. Fungsi printAdjacencyMatrix digunakan untuk mencetak matriks adjacency yang merepresentasikan hubungan antar simpul dalam graf. Kode ini mencerminkan bagaimana adjacency matrix digunakan sebagai representasi graf yang efisien untuk menyimpan dan memvisualisasikan hubungan antar simpul, cocok untuk aplikasi yang membutuhkan akses cepat terhadap koneksi antar simpul.

E. Kesimpulan

Graf merupakan salah satu struktur data yang sangat penting dalam ilmu komputer, digunakan untuk merepresentasikan hubungan atau koneksi antar objek. Dengan memahami konsep graf, jenis-jenisnya (berarah dan tidak berarah), serta metode representasi seperti matriks ketetanggaan dan daftar ketetanggaan, kita dapat mengimplementasikan berbagai algoritma traversal seperti DFS dan BFS untuk menjelajahi graf secara efisien. Selain itu, graf juga memiliki banyak aplikasi praktis, termasuk pencarian jalur terpendek, analisis hubungan, dan pengurutan berdasarkan ketergantungan (topological sorting). Melalui implementasi dinamis menggunakan pointer, graf dapat dikelola dengan lebih fleksibel dan efisien, menjadikannya alat yang sangat berguna dalam menyelesaikan berbagai permasalahan kompleks di dunia nyata.

Yogi Hafidh Maulana – 2211104061©