LAPORAN PRAKTIKUM
Modul 14
Graph

Disusun Oleh:
Berlian Seva Astryana - 2311104067
Kelas:
SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERINGFAKULTAS INFORMATIKA
TELKOM UNIVERSITYPURWOKERTO
2024

1. **Tujuan**
   1.1 Memahami konsep graph.
   1.2 Mengimplementasikan graph dengan menggunakan pointer.
2. **Landasan Teori**

Graph adalah struktur data yang terdiri dari himpunan simpul (node atau vertex) dan garis penghubung (edge). Graph digunakan untuk merepresentasikan berbagai hubungan antara objek, seperti peta jalan, jaringan komputer, atau hubungan antar entitas dalam basis data. Ada dua jenis utama graph, yaitu graph berarah (directed graph), di mana setiap edge memiliki arah tertentu, dan graph tidak berarah (undirected graph), di mana edge tidak memiliki arah. Pada graph berarah, simpul A yang terhubung ke simpul B melalui edge belum tentu berarti simpul B terhubung kembali ke simpul A. Sebaliknya, graph tidak berarah memastikan bahwa jika ada edge antara simpul A dan B, maka hubungan tersebut bersifat dua arah. Representasi graph dapat diwujudkan dalam beberapa bentuk, seperti matriks ketetanggaan (adjacency matrix) atau multi linked list, dengan kelebihan dan kekurangan masing-masing.

Dalam implementasi praktis, algoritma penelusuran seperti Breadth First Search (BFS) dan Depth First Search (DFS) sering digunakan untuk mengeksplorasi graph. BFS mengunjungi simpul berdasarkan levelnya, sedangkan DFS menjelajahi simpul secara mendalam sebelum kembali ke simpul sebelumnya. Graph juga digunakan dalam berbagai aplikasi seperti penjadwalan (topological sort), pencarian lintasan terpendek (seperti algoritma Dijkstra), dan analisis jaringan. Representasi dan pengolahan graph dalam program sering memanfaatkan struktur data dinamis seperti pointer untuk fleksibilitas dan efisiensi.

3. **Guided**

Program:

```
#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(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:

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
```

Program ini merepresentasikan graf menggunakan struktur data dinamis berbasis pointer. Graf dikelola dengan struktur utama Graph, yang berisi simpul-simpul yang direpresentasikan oleh struktur ElmNode. Setiap simpul menyimpan informasi, status kunjungan, dan daftar sisi yang terhubung. Struktur ElmEdge digunakan untuk merepresentasikan sisi, yang menyimpan pointer ke simpul tujuan. Program dapat menambahkan simpul ke graf dengan fungsi InsertNode dan menghubungkan dua simpul dengan fungsi ConnectNode. Traversal graf dapat dilakukan menggunakan Depth First Search (DFS) dan Breadth First Search (BFS). Traversal DFS dilakukan secara rekursif, dimulai dari simpul awal, mencetak simpul yang dikunjungi, dan melanjutkan ke simpul yang terhubung secara mendalam. Traversal BFS menggunakan antrian (queue) untuk mengunjungi simpul secara lebar, mulai dari simpul awal dan melanjutkan ke semua simpul yang terhubung pada tingkat berikutnya. Selain itu, fungsi ResetVisited mengatur ulang status kunjungan semua simpul, memungkinkan traversal ulang dari awal. Dalam program utama, simpul 'A' hingga 'H' ditambahkan ke graf, dan hubungan antar-simpul dibentuk, seperti 'A' yang terhubung ke 'B' dan 'C'. Setelah itu, program mencetak hasil traversal graf menggunakan DFS dan BFS, dengan output mencerminkan urutan simpul yang dikunjungi sesuai metode traversal yang digunakan. Program ini memanfaatkan struktur dinamis sehingga fleksibel untuk menambahkan simpul dan sisi tanpa batasan ukuran tetap, serta dapat digunakan untuk berbagai jenis graf kecil hingga sedang.

4. UNGUIDED

4.1 Program:

```
#include <iostream>
#include <vector>
#include <string>
#include <iomanip> // Untuk pengaturan format output

using namespace std;

int main() {
    int jumlah_simpul;

    // Meminta jumlah simpul
    cout << "Silakan masukan jumlah simpul: ";
    cin >> jumlah_simpul;

    vector<string> simpul(jumlah_simpul); // Menyimpan nama simpul
    vector<vector<int>> graph(jumlah_simpul, vector<int>(jumlah_simpul, 0)); // Matriks adjacency

    // Meminta nama simpul
    cout << "\nSilakan masukan nama simpul" << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }
```

```
    // Meminta bobot antar simpul
    cout << "\nSilakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << simpul[i] << "--> " << simpul[j] << " = ";
            cin >> graph[i][j];
        }
    }

    // Output header tabel
    cout << "\n\t";
    for (const auto& s : simpul) {
        cout << setw(10) << s; // Setiap nama simpul memiliki lebar kolom 10
    }
    cout << endl;

    // Output matriks adjacency
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << setw(10) << simpul[i]; // Nama baris (simpul)
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << setw(10) << graph[i][j]; // Bobot antar simpul
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H

Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

4.2 Program:

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

using namespace std;

int main() {
    int jumlah_simpul, jumlah_sisi;

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

    cout << "Masukkan jumlah sisi: ";
    cin >> jumlah_sisi;
```

```
    // Matriks adjacency diinisialisasi dengan 0
    vector<vector<int>>  adjacency_matrix(jumlah_simpul,  vector<int>(jumlah_simpul,
0));

    // Input pasangan simpul yang terhubung
    cout << "Masukkan pasangan simpul:" << endl;
    for (int i = 0; i < jumlah_sisi; ++i) {
        int simpul1, simpul2;
        cin >> simpul1 >> simpul2;

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

    // Output adjacency matrix
    cout << "Adjacency Matrix:" << endl;
    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << adjacency_matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H

Process returned 0 (0x0)   execution time : 0.081 s
Press any key to continue.
```

5. KESIMPULAN

Graph merupakan struktur data yang sangat berguna untuk merepresentasikan hubungan antar entitas dalam berbagai konteks, seperti jaringan, penjadwalan, dan analisis data. Dengan memahami konsep dasar graph, jenis-jenisnya (berarah dan tidak berarah), serta algoritma yang relevan seperti BFS dan DFS, kita dapat mengimplementasikan solusi yang efisien untuk berbagai permasalahan kompleks. Representasi graph menggunakan multi linked list memberikan fleksibilitas dalam penanganan data yang dinamis, sedangkan algoritma seperti topological sort membantu dalam menyusun keterurutan elemen secara logis.