LAPORAN PRAKTIKUM
MODUL 14
“GRAPH”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

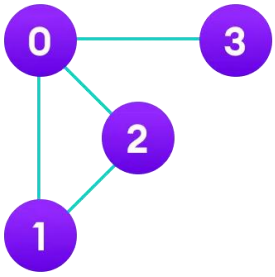
2024

1. Tujuan

- Memahami konsep graph.
- Mengimplementasikan graph dengan menggunakan pointer.

2. Landasan Teori

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan. Simpul pada graph disebut dengan verteks (V), sedangkan sisi yang menghubungkan antar verteks disebut edge (E). Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y.
Graph dapat dibedakan berdasarkan arah jelajahnya dan ada tidaknya label bobot pada relasinya.



Berdasarkan arah jelajahnya graph dibagi menjadi dua yaitu,

- Undirected Graph
  Pada undirected graph, simpul-simpulnya terhubung dengan edge yang sifatnya dua arah. Misalnya kita punya simpul 1 dan 2 yang saling terhubung, kita bisa menjelajah dari simpul 1 ke simpul 2, begitu juga sebaliknya.
- Directed Graph
  Kebalikan dari undirected graph, pada graph jenis ini simpul-simpulnya terhubung oleh edge yang hanya bisa melakukan jelajah satu arah pada simpul yang ditunjuk. Sebagai contoh jika ada simpul A yang terhubung ke simpul B, namun arah panahnya menuju simpul B, maka kita hanya bisa melakukan jelajah (traversing) dari simpul A ke simpul B, dan tidak berlaku sebaliknya.

Metode penelusuran pada graf adalah teknik untuk mencari simpul dan sisi dalam graf tanpa membuat loop. Beberapa metode penelusuran pada graf, di antaranya:

- BFS adalah singkatan dari Breadth-first search. BFS adalah metode untuk membuat grafik data, pohon pencarian, dan melintasi struktur. Metode ini mengunjungi & menandai semua simpul penting dalam jaringan dengan cara yang tepat. Algoritme ini memilih satu simpul (titik awal atau asal) dalam jaringan dan mengunjungi semua simpul di dekat simpul yang dipilih. Setelah mengunjungi dan menandai simpul awal, algoritme melanjutkan ke simpul tak berpenghuni berikutnya dan menganalisisnya. Semua simpul ditunjukkan setelah dikunjungi. Siklus ini berlanjut hingga semua simpul dalam grafik telah dikunjungi dan ditandai dengan benar.

- DFS (Depth initial search) merupakan strategi pencarian mendalam yang dapat digunakan untuk menemukan atau menjelajahi grafik atau pohon. Sebelum melakukan backtracking, algoritma dimulai dari akar pohon dan menjelajahi setiap jalur. Ketika iterasi mencapai jalan buntu, struktur informasi tumpukan digunakan untuk mengingat, mengidentifikasi titik puncak berikutnya, dan memulai pencarian. Bentuk lengkap DFS adalah pencarian mendalam.

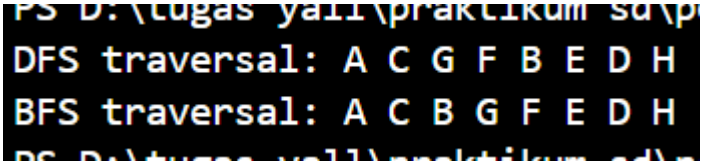
3. Guided

Program ini adalah implementasi struktur data graf menggunakan linked list dengan dua algoritma traversal yaitu DFS (Depth First Search) dan BFS (Breadth First Search). Dimana program utamanya akan membuat graph dengan 8 node (A-H), membuat koneksi antar node membentuk struktur pohon, melakukan traversal dengan DFS dan BFS. Output program akan menampilkan urutan node yang dikunjungi menggunakan kedua metode traversal tersebut, menunjukkan perbedaan cara kerja DFS dan BFS dalam menelusuri graf.

Code program:

```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;
};

// Fungsi untuk membuat graf baru
void CreateGraph(Graph &G) {
    G.first = NULL; // Inisialisasi graf kosong
}

// Fungsi untuk menambahkan node baru ke graf
void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode; // Alokasi memori untuk node baru
    newNode->info = X;              // Menyimpan informasi
    newNode->visited = false;       // Status awal belum dikunjungi
    newNode->firstEdge = NULL;      // Belum ada tepi
    newNode->Next = NULL;           // Node berikutnya adalah NULL

    if (G.first == NULL) {
        G.first = newNode; // Jika graf kosong, tambahkan sebagai simpul pertama
    } else {
        ElmNode *temp = G.first;
        while (temp->Next != NULL) {
            temp = temp->Next; // Cari node terakhir dalam daftar
        }
        temp->Next = newNode; // Tambahkan node baru di akhir
    }
}

// Fungsi untuk membuat hubungan antar node (menambahkan tepi)
void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge; // Alokasi memori untuk tepi baru
    newEdge->Node = N2;             // Tepi menghubungkan ke simpul N2
    newEdge->Next = N1->firstEdge; // Sisipkan di awal daftar tepi
    N1->firstEdge = newEdge;        // Perbarui daftar tepi N1
}

// Fungsi untuk mencetak informasi node dalam graf
void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " "; // Cetak informasi setiap simpul
        temp = temp->Next;         // Lanjut ke simpul berikutnya
    }
    cout << endl;
}

// Fungsi untuk mengatur ulang status visited semua simpul
void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false; // Reset visited ke false
        temp = temp->Next;
    }
}

// Fungsi untuk traversal graf menggunakan DFS
void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return; // Basis rekursi, jika simpul NULL, selesai
    }
    N->visited = true; // Tandai simpul telah dikunjungi
    cout << N->info << " "; // Cetak informasi simpul

    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node); // Rekursi ke simpul yang terhubung
        }
        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 dari program:



4. Unguided

1. Ini merupakan program untuk menghitung jarak antar kota menggunakan representasi graph dalam bentuk matriks adjacency. Digunakan Array string namaSimpul[MAX] untuk menyimpan nama-nama kota, Array 2D bobot[MAX][MAX] untuk menyimpan jarak antar kota dalam bentuk matriks adjacency dan Variabel jumlahSimpul untuk menyimpan jumlah kota. Program menampilkan hasil dalam bentuk matriks dengan format yang rapi menggunakan setw().
   Code program:

```
#include <iostream>
#include <string>
#include <iomanip>
#define MAX 100
using namespace std;

int main() {
   int jumlahSimpul;
   string namaSimpul[MAX];
   int bobot[MAX][MAX];

   // Input jumlah simpul
   cout << "Silakan masukan jumlah simpul : ";
   cin >> jumlahSimpul;

   // Input nama simpul
   for(int i = 0; i < jumlahSimpul; i++) {
      cout << "Simpul " << i+1 << " : ";
      cin >> namaSimpul[i];
   }

   // Inisialisasi matriks bobot dengan 0
   for(int i = 0; i < jumlahSimpul; i++) {
      for(int j = 0; j < jumlahSimpul; j++) {
         bobot[i][j] = 0;
      }
   }
```

```
    // Input bobot antar simpul
    cout << "Silakan masukkan bobot antar simpul\n";

    // Input untuk BALI ke BALI
    cout << namaSimpul[0] << "--> " << namaSimpul[0] << " = ";
    cin >> bobot[0][0];

    // Input untuk BALI ke PALU
    cout << namaSimpul[0] << "--> " << namaSimpul[1] << " = ";
    cin >> bobot[0][1];

    // Input untuk PALU ke BALI
    cout << namaSimpul[1] << "--> " << namaSimpul[0] << " = ";
    cin >> bobot[1][0];

    // Input untuk PALU ke PALU
    cout << namaSimpul[1] << "--> " << namaSimpul[1] << " = ";
    cin >> bobot[1][1];

    // Menampilkan matriks bobot
    cout << "\n";
    cout << setw(10) << " ";
    for(int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << namaSimpul[i];
    }
    cout << "\n";

    for(int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << namaSimpul[i];
        for(int j = 0; j < jumlahSimpul; j++) {
            cout << setw(10) << bobot[i][j];
        }
        cout << "\n";
    }

    return 0;
}
```

Output program:

```
Silakan masukan jumlah simpul : 2
Simpul 1 : Bali
Simpul 2 : Palu
Silakan masukkan bobot antar simpul
Bali--> Bali = 0
Bali--> Palu = 3
Palu--> Bali = 4
Palu--> Palu = 0

               Bali      Palu
      Bali        0         3
      Palu        4         0
```

2. Sama seperti program pertama, program kedua ini juga merepresentasikan graf tidak berarah menggunakan adjacency matrix. Program ini memiliki beberapa komponen penting seperti, Array 2D adjacencyMatrix untuk menyimpan representasi graf, input untuk jumlah simpul dan jumlah sisi, loop untuk menerima pasangan simpul yang terhubung, pengisian matrix di kedua arah karena graf tidak berarah, lalu tampilan matrix hasil. Cara kerja program dengan memasukkan jumlah simpul, masukkan jumlah sisi, masukkan pasangan simpul yang terhubung dan kemudian program akan menampilkan adjacency matrixnya.
   Code program:

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

int main() {
   int jumlahSimpul, jumlahSisi;
   int adjacencyMatrix[100][100] = {0}; // Inisialisasi matrix dengan 0

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

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

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

      // Karena graf tidak berarah, kita perlu mengisi kedua arah
      // Kurangi 1 dari input karena array dimulai dari 0
      adjacencyMatrix[simpul1-1][simpul2-1] = 1;
```

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

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

    return 0;
}
```

Output dari program:

```
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:\tugas yall\praktikum
```

5. Kesimpulan

Laporan ini membahas konsep graph sebagai struktur data non-linear yang terdiri dari simpul (vertex) dan sisi (edge), serta implementasinya menggunakan dua metode traversal, yaitu DFS (Depth First Search) dan BFS (Breadth First Search). Melalui program guided, graph diimplementasikan dengan linked list untuk menampilkan urutan kunjungan simpul menggunakan kedua metode traversal tersebut. Sedangkan pada program unguided, graph direpresentasikan menggunakan adjacency matrix untuk menghitung jarak antar simpul (misalnya kota) dan menampilkan matriks hasil. Laporan ini menunjukkan perbedaan cara kerja DFS dan BFS serta representasi graf tidak berarah menggunakan matriks.