LAPORAN PRAKTIKUM STRUKTUR DATA
MODUL 14
GRAPH

Disusun Oleh:
Dwi Candra Pratama 2211104035 / SE07-02

Assisten Praktikum:
Aldi Putra Hamalsyah
Andini Nur Hidayah

Dosen Pengampu:
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2024

GUIDED

TUJUAN PRAKTIKUM

1. Memahami konsep graph
2. Mengimplementasikan graph dengan menggunakan pointer.

Pengertian

Graph merupakan himpunan tidak kosong dari node (vertec) dan garis penghubung (edge). Contoh sederhana tentang graph, yaitu antara Tempat Kost Anda dengan Common Lab. Tempat Kost Anda dan Common Lab merupakan node (vertec). Jalan yang menghubungkan tempat Kost dan Common Lab merupakan garis penghubung antara keduanya (edge).

Jenis-Jenis Graph

A. Graph Berarah (Directed Graph)

Graph Berarah (Directed Graph atau digraph) adalah graph di mana setiap sisi memiliki arah tertentu. Arah ini menunjukkan hubungan dari satu simpul ke simpul lainnya, sehingga hubungan bersifat asymmetrical (tidak saling timbal balik).

- Representasi Graph

  Pada dasarnya representasi dari graph berarah sama dengan graph tak-berarah. Perbedaannya apabila graph tak-berarah terdapat node A dan node B yang terhubung, secara otomatis terbentuk panah bolak balik dari A ke B dan B ke A. Pada Graph berarah node A terhubung dengan node B, belum tentu node B terhubung dengan node A.

- Topological Sort
  a. Pengertian

     Diberikan urutan partial dari elemen suatu himpunan, dikehendaki agar elemen yang terurut parsial tersebut mempunyai keterurutan linier. Contoh dari keterurutan parsial banyak dijumpai dalam kehidupan sehari-hari, misalnya:

     1. Dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisite mata pelajaran lain. Urutan linier adalah urutan untuk seluruh mata pelajaran dalamkurikulum
     2. Dalam suatu proyek, suatu pekerjaan harus dikerjakan lebih dulu dari pekerjaan lain (misalnya membuat fondasi harus sebelum dinding, membuat dinding harus sebelum pintu. Namun pintu dapat dikerjakan bersamaan dengan jendela, dsb
     3. Dalam sebuah program Pascal, pemanggilan prosedur harus sedemikian rupa, sehingga peletakan prosedur pada teks program harus seuai dengan urutan (partial) pemanggilan.

B. Graph Tidak Berarah (Undirected Graph)

Jenis graf di mana setiap sisi (edge) tidak memiliki arah. Artinya, hubungan antara dua simpul (node) bersifat dua arah atau timbal balik. Jika ada sisi antara simpul A dan B, maka kita dapat bergerak dari A ke B dan juga dari B ke A.

Misalnya:

Dari gambar contoh graph pada halaman sebelumnya dapat disimpulkan bahwa: A bertetangga dengan B, B bertetangga dengan C, A tidak bertetangga dengan C, B tidak bertetangga dengan D.

- Representasi Graph

  Dari definisi graph dapat kita simpulkan bahwa graph dapat direpresentasikan dengan Matrik Ketetanggaan (Adjacency Matrices), yaitu matrik yang menyatakan keterhubungan antar node dalam graph. Implementasi matrik ketetanggaan dalam bahasa pemrograman dapat berupa : Array 2 Dimensi dan Multi Linked List. Graph dapat direpresentasikan dengan matrik n x n, dimana n merupakan jumlah node dalam graph tersebut.

C. Metode-Metode Penulusuran Graph

1. Breadth First Search (BFS)
   Cara kerja algoritma ini adalah dengan mengunjungi root (depth 0) kemudian ke depth 1, 2, dan seterusnya. Kunjungan pada masing-masing level dimulai dari kiri ke kanan. Contohnya seperti ini
2. Depth First Search (DFS)
   Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian rekursif ke subtree node tersebut.

Berikut adalah IMPLEMENTASI SAAT PRAKTIKUM

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

using namespace std;

// Struktur untuk merepresentasikan simpul pada graf
struct ElmNode;

// Struktur untuk merepresentasikan sisi pada graf
struct ElmEdge {
   ElmNode *Node;  // Simpul tujuan yang terhubung oleh sisi ini
   ElmEdge *Next;  // Pointer ke sisi berikutnya dari simpul yang sama
};

// Struktur untuk simpul pada graf
struct ElmNode {
   char info;          // Informasi pada simpul (contoh: 'A', 'B', dll.)
   bool visited;       // Status apakah simpul telah dikunjungi (untuk DFS/BFS)
   ElmEdge *firstEdge; // Pointer ke sisi pertama dari simpul ini
   ElmNode *Next;      // Pointer ke simpul berikutnya dalam daftar
};
```

```
// Struktur untuk graf
struct Graph {
   ElmNode *first; // Pointer ke simpul pertama dalam graf
};

// Membuat graf kosong
void CreateGraph(Graph &G) {
   G.first = NULL; // Awalnya, graf tidak memiliki simpul
}

// Menambahkan simpul baru ke graf
void InsertNode(Graph &G, char X) {
   ElmNode *newNode = new ElmNode; // Buat simpul baru
   newNode->info = X;              // Berikan informasi pada simpul
   newNode->visited = false;       // Tandai simpul sebagai belum dikunjungi
   newNode->firstEdge = NULL;      // Tidak memiliki sisi pada awalnya
   newNode->Next = NULL;

   if (G.first == NULL) {          // Jika graf kosong
      G.first = newNode;           // Jadikan simpul baru sebagai simpul pertama
   } else {                        // Jika sudah ada simpul
      ElmNode *temp = G.first;     // Temukan simpul terakhir
      while (temp->Next != NULL) {
         temp = temp->Next;
      }
      temp->Next = newNode;        // Tambahkan simpul baru di akhir daftar
   }
}

// Menghubungkan dua simpul dengan sisi
void ConnectNode(ElmNode *N1, ElmNode *N2) {
   ElmEdge *newEdge = new ElmEdge; // Buat sisi baru
   newEdge->Node = N2;             // Hubungkan ke simpul tujuan
   newEdge->Next = N1->firstEdge;  // Hubungkan ke sisi pertama simpul asal
   N1->firstEdge = newEdge;        // Jadikan sisi ini sebagai sisi pertama
}

// Mencetak semua simpul pada graf
void PrintInfoGraph(Graph G) {
   ElmNode *temp = G.first;
   while (temp != NULL) {
      cout << temp->info << " "; // Cetak informasi pada simpul
      temp = temp->Next;
   }
   cout << endl;
}
```

```
// Mengatur ulang status "visited" semua simpul
void ResetVisited(Graph &G) {
   ElmNode *temp = G.first;
   while (temp != NULL) {
      temp->visited = false; // Tandai semua simpul sebagai belum dikunjungi
      temp = temp->Next;
   }
}

// Traversal Depth-First Search (DFS)
void PrintDFS(Graph G, ElmNode *N) {
   if (N == NULL) {
      return; // Jika simpul kosong, hentikan
   }
   N->visited = true;           // Tandai simpul sebagai dikunjungi
   cout << N->info << " ";      // Cetak simpul
   ElmEdge *edge = N->firstEdge;
   while (edge != NULL) {       // Kunjungi semua sisi dari simpul
      if (!edge->Node->visited) {
         PrintDFS(G, edge->Node); // Rekursif DFS ke simpul tujuan
      }
      edge = edge->Next;        // Pindah ke sisi berikutnya
   }
}

// Traversal Breadth-First Search (BFS)
void PrintBFS(Graph G, ElmNode *N) {
   queue<ElmNode*> q;          // Buat antrian untuk BFS
   q.push(N);                  // Masukkan simpul awal ke antrian
   N->visited = true;          // Tandai simpul sebagai dikunjungi

   while (!q.empty()) {        // Selama antrian tidak kosong
      ElmNode *current = q.front();
      q.pop();
      cout << current->info << " "; // Cetak simpul saat ini

      ElmEdge *edge = current->firstEdge;
      while (edge != NULL) { // Kunjungi semua sisi dari simpul
         if (!edge->Node->visited) {
            edge->Node->visited = true; // Tandai simpul tujuan sebagai dikunjungi
            q.push(edge->Node);         // Masukkan ke antrian
         }
         edge = edge->Next;             // Pindah ke sisi berikutnya
      }
   }
```

```
}

int main() {
    Graph G;
    CreateGraph(G); // Buat graf kosong

    // Tambahkan simpul ke graf
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    // Ambil pointer ke simpul-simpul
    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;

    // Hubungkan simpul-simpul dengan sisi
    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(D, H);

    // Traversal DFS
    cout << "DFS traversal: ";
    ResetVisited(G); // Reset status visited sebelum traversal
    PrintDFS(G, A);
    cout << endl;

    // Traversal BFS
    cout << "BFS traversal: ";
    ResetVisited(G); // Reset status visited sebelum traversal
    PrintBFS(G, A);
    cout << endl;
```

```
    return 0;
}
```

OutPutnya:

```
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13> cd 'd:\Semester5
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13\Guided\output> &
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13\Guided\output>
```

UnGuided

Latihan soal

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

   Output Program

```
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
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

Process returned 0 (0x0)   execution time : 11.763 s
Press any key to continue.
```

   Source Codenya:

```
#include <iostream>
#include <vector>
#include <limits.h>
using namespace std;

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

    vector<string> kota(n);
    cout << "Silakan masukan nama simpul:\n";
    for (int i = 0; i < n; i++) {
```

```
        cout << "Simpul " << i + 1 << ": ";
        cin >> kota[i];
    }

    vector<vector<int>> jarak(n, vector<int>(n, INT_MAX));
    cout << "Silakan masukan bobot antar simpul:\n";
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            cout << kota[i] << " --> " << kota[j] << ": ";
            int bobot;
            cin >> bobot;
            jarak[i][j] = bobot;
            jarak[j][i] = bobot;
        }
    }

    cout << "\nAdjacency Matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (jarak[i][j] == INT_MAX)
                cout << "0 ";
            else
                cout << jarak[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

OutPutnya:

```
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13\UnGuided\output> & .\'SoalNo1.exe'
Silakan masukan jumlah simpul: 2
Silakan masukan nama simpul:
Simpul 1: Solo
Simpul 2: Purwokerto
Silakan masukan bobot antar simpul:
Solo --> Purwokerto: 5

Adjacency Matrix:
0 5
5 0
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13\UnGuided\output>
```

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.

Input Contoh:
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul:
1 2
1 3
2 4
3 4

Output Contoh:
Adjacency Matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0

Source Codenya:

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

int main() {
    int n, m;
    cout << "Masukkan jumlah simpul: ";
    cin >> n;
    cout << "Masukkan jumlah sisi: ";
    cin >> m;

    vector<vector<int>> adjMatrix(n, vector<int>(n, 0));

    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adjMatrix[u - 1][v - 1] = 1;
        adjMatrix[v - 1][u - 1] = 1; // Graf tak berarah
    }

    cout << "Adjacency Matrix:\n";
    for (const auto &row : adjMatrix) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
```

```
        }

        return 0;
}
```

OutPutnya:

```
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13\UnGuided\output> & .\'SoalNo2.exe'
Masukkan jumlah simpul: 9
Masukkan jumlah sisi: 3
Masukkan pasangan simpul:
1 3
5 6
6 7
Adjacency Matrix:
0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
PS D:\Semester5\StrukturData\PraktikumStrukturData\Pertemuan 13\UnGuided\output>
```