LAPORAN PRAKTIKUM
Modul 14
GRAPH

Disusun Oleh:
Jauhar Fajar Zuhair
2311104072
S1SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

Tujuan

1. Memahami konsep dasar graph
2. Mampu mengimplementasikan graph menggunakan pointer

Pengertian Graph

Graph adalah struktur data yang terdiri dari kumpulan titik (node/vertex) yang saling terhubung melalui garis penghubung (edge). Contoh sederhananya seperti rute dari kost ke laboratorium komputer - dimana kost dan lab adalah node, sedangkan jalan penghubungnya adalah edge.

Jenis-Jenis Graph

1. Graph Berarah (Directed Graph)

- Memiliki edge dengan arah tertentu
- Hubungan antar node bersifat satu arah
- Jika node A terhubung ke B, belum tentu B terhubung ke A
- Biasa direpresentasikan menggunakan multilist karena sifatnya yang dinamis

2. Graph Tidak Berarah (Undirected Graph)

- Edge tidak memiliki arah tertentu
- Hubungan antar node bersifat dua arah
- Jika node A terhubung ke B, maka B juga terhubung ke A
- Bisa memiliki bobot/nilai pada edge (misalnya jarak atau biaya)

Metode Penelusuran Graph

1. Breadth First Search (BFS)

- Menelusuri graph level per level
- Dimulai dari root (kedalaman 0)
- Dilanjutkan ke level 1, 2, dst
- Pada tiap level, penelusuran dari kiri ke kanan

2. Depth First Search (DFS)

- Menelusuri graph dengan mengutamakan kedalaman
- Mengunjungi root terlebih dahulu
- Kemudian secara rekursif mengunjungi subtree dari node tersebut
- Baru pindah ke cabang lain setelah satu jalur selesai ditelusuri

Representasi Graph

Graph dapat direpresentasikan dalam dua cara:

1. Array 2 Dimensi (Matriks Ketetanggaan)
2. Multi Linked List - lebih fleksibel karena sifatnya yang dinamis

Kedua representasi ini memungkinkan penyimpanan informasi tentang keterhubungan antar node dalam graph.

Guided

Guided1.cpp

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

using namespace std;

struct ElmNode;

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

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;
}
```

Unguided

unGuided1.cpp

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

```
class Graph
{
private:
  int V; // jumlah vertex/kota
  vector<vector<int>> adjMatrix;
  vector<string> cityNames;

public:
  // Constructor
  Graph(int vertices)
  {
    V = vertices;
    // Inisialisasi matrix dengan ukuran V x V
    adjMatrix.resize(V, vector<int>(V, 0));
    cityNames.resize(V);
  }

  // Menambahkan nama kota
  void addCityName(int index, string name)
  {
    cityNames[index] = name;
  }

  // Menambahkan edge dengan bobot
  void addEdge(int source, int dest, int weight)
  {
    adjMatrix[source][dest] = weight;
  }

  // Mendapatkan nama kota
  string getCityName(int index)
  {
    return cityNames[index];
  }

  // Menampilkan matrix adjacency
  void printMatrix()
  {
    // Print header kota
    cout << "\n      ";
    for (int i = 0; i < V; i++)
    {
      cout << left << setw(8) << cityNames[i];
    }
    cout << "\n";
```

```
    // Print matrix dengan nama kota
    for (int i = 0; i < V; i++)
    {
      cout << left << setw(6) << cityNames[i];
      for (int j = 0; j < V; j++)
      {
        cout << left << setw(8) << adjMatrix[i][j];
      }
      cout << "\n";
    }
  }
};

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

  Graph g(V);

  // Input nama kota
  cout << "Silakan masukan nama simpul\n";
  cin.ignore(); // Clear buffer
  for (int i = 0; i < V; i++)
  {
    string cityName;
    cout << "Simpul " << i + 1 << " : ";
    getline(cin, cityName);
    g.addCityName(i, cityName);
  }

  // Input bobot antar simpul
  cout << "\nSilakan masukkan bobot antar simpul\n";
  for (int i = 0; i < V; i++)
  {
    for (int j = 0; j < V; j++)
    {
      if (i != j)
      {
        int weight;
        cout << g.getCityName(i) << "--> " << g.getCityName(j) << " = ";
        cin >> weight;
        g.addEdge(i, j, weight);
```

```
      }
    }
  }

  // Tampilkan hasil
  g.printMatrix();

  return 0;
}
```

Output

```
Silakan masukan jumlah simpul: 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU

Silakan masukkan bobot antar simpul
BALI--> PALU = 3
PALU--> BALI = 4

      BALI    PALU
BALI  0       3
PALU  4       0
```

unGuided2.cpp

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

class Graph
{
private:
  int V; // jumlah vertex/simpul
  vector<vector<int>> adjMatrix;

public:
  // Constructor
  Graph(int vertices)
  {
    V = vertices;
    // Inisialisasi matrix dengan ukuran V x V dengan nilai 0
    adjMatrix.resize(V, vector<int>(V, 0));
  }

  // Menambahkan edge untuk graf tidak berarah
  void addEdge(int v1, int v2)
```

```
  {
    // Karena graf tidak berarah, kedua arah diberi nilai 1
    adjMatrix[v1 - 1][v2 - 1] = 1;
    adjMatrix[v2 - 1][v1 - 1] = 1;
  }

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

int main()
{
  int V, E;

  cout << "Masukkan jumlah simpul: ";
  cin >> V;

  cout << "Masukkan jumlah sisi: ";
  cin >> E;

  Graph g(V);

  cout << "Masukkan pasangan simpul:" << endl;
  for (int i = 0; i < E; i++)
  {
    int v1, v2;
    cin >> v1 >> v2;
    g.addEdge(v1, v2);
  }

  g.printMatrix();

  return 0;
}
```

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
```