LAPORAN PRAKTIKUM

Modul 14

“Graph”

Disusun Oleh:

Marvel Sanjaya Setiawan (2311104053)

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. **Tujuan**
    - Konsep dasar graf.
    - Implementasi graf menggunakan pointer.
2. **Landasan Teori**

    Graph

    Graph adalah himpunan node (vertec) dan edge yang menghubungkannya.

    Jenis Graph:

    - Graph Berarah: Edge memiliki arah tertentu.
    - Graph Tidak Berarah: Edge tanpa arah.

    Representasi Graph:

    - Multilist: Menggunakan pointer untuk hubungan dinamis.
    - Matriks Ketetanggaan: Matriks 2D untuk koneksi antar node.

    Topological Sort

    - Urutkan elemen graph berarah secara linier berdasarkan prioritas.
    - Proses: Pilih node tanpa predecessor, masukkan ke list, hapus node, ulangi.

    Metode Penelusuran

    - BFS: Penelusuran per level menggunakan queue.
    - DFS: Penelusuran rekursif menggunakan stack.

    Operasi Dasar Graph

    Tambah/hapus node dan edge, hubungkan node, dan tampilkan informasi graph.

Telkom University Purwokerto

3. Guided

```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;
};

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, E);
    ConnectNode(B, D);
    ConnectNode(B, F);
    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;
}
```

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

Cara Kerja:

1. ElmNode: Menyimpan data, pointer ke node berikutnya (Next), dan pointer ke sisi pertama (firstEdge).
2. ElmEdge: Menyimpan pointer ke simpul yang terhubung (Node) dan pointer ke sisi berikutnya (Next).
3. Graph: Mengelola graf dengan pointer ke simpul pertama (first).
4. CreateGraph: Inisialisasi graf dengan mengatur pointer first ke NULL.
5. InsertNode: Menambahkan simpul baru di akhir graf.
6. ConnectNode: Menambahkan sisi baru antara dua simpul.
7. PrintInfoGraph: Menampilkan informasi semua simpul.
8. ResetVisited: Mengatur ulang status kunjungan semua simpul ke false.
9. PrintDFS: Traversal graf dengan algoritma Depth-First Search (DFS).
10. PrintBFS: Traversal graf dengan algoritma Breadth-First Search (BFS).
11. Main:
    - Membuat graf dan menambahkan simpul 'A' hingga 'H'.
    - Menghubungkan simpul dengan sisi.
    - Melakukan traversal DFS dan BFS mulai dari simpul 'A' dan menampilkan hasilnya.

4. Unguided

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>

using namespace std;

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

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

    vector<vector<int>> graph(V, vector<int>(V, 0));
    cout << "Silakan masukkan bobot antar simpul \n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cout << cities[i] << "--> " << cities[j] << " = ";
            cin >> graph[i][j];
        }
    }

    // Print header for the matrix
    cout << "\n" << setw(8) << " ";
    for (int i = 0; i < V; i++) {
        cout << setw(8) << cities[i];
    }
    cout << endl;

    // Print matrix
    for (int i = 0; i < V; i++) {
        cout << setw(8) << cities[i];
        for (int j = 0; j < V; j++) {
            cout << setw(8) << graph[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

```
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
```

Cara Kerja:

1. Variabel Utama: V untuk jumlah simpul.
2. Input Pengguna: Memasukkan jumlah simpul, nama simpul, dan bobot antar simpul..
3. Struktur Data: vector<string> cities untuk menyimpan nama simpul, vector<vector<int>> graph untuk adjacency matrix.
4. Logika Program: Inisialisasi matriks ketetanggaan dengan 0, Input bobot antar simpul.
5. Output: Menampilkan adjacency matrix..
6. main: Menginput data dari pengguna, menampilkan adjacency matrix dalam bentuk tabel.

Telkom University Purwokerto

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

using namespace std;

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

    // Membuat adjacency matrix dengan ukuran VxV dan menginisialisasi dengan 0
    vector<vector<int>> adjacencyMatrix(V, vector<int>(V, 0));

    cout << "Masukkan pasangan simpul:" << endl;
    for (int i = 0; i < E; i++) {
        int u, v;
        cin >> u >> v;
        // Karena simpul mulai dari 1, bukan 0, kurangi 1 untuk indeks matriks
        u--;
        v--;
        adjacencyMatrix[u][v] = 1;
        adjacencyMatrix[v][u] = 1; // Graf tidak berarah
    }

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

    return 0;
}

```

```
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
```

Cara Kerja:

1. Variabel Utama: V (jumlah simpul) dan E (jumlah sisi).
2. Input Pengguna: Memasukkan jumlah simpul (V), jumlah sisi (E), dan pasangan simpul (u, v).
3. Struktur Data: adjacencyMatrix (matriks ketetanggaan V x V diinisialisasi 0).
4. Logika Program: Inisialisasi matriks ketetanggaan, input pasangan simpul, perbarui matriks dengan 1.
5. Output: Menampilkan adjacency matrix.
6. Main: Meminta input pengguna, mengisi adjacency matrix, dan menampilkan adjacency matrix.

5. Kesimpulan

Graf merepresentasikan hubungan menggunakan simpul dan sisi. Implementasi bisa menggunakan pointer atau matriks ketetanggaan, dengan metode penelusuran BFS dan DFS untuk berbagai aplikasi. Operasi dasar termasuk menambah, menghapus, dan menampilkan simpul serta sisi.