

LAPORAN PRAKTIKUM
Modul 14
“Graph ”



Disusun Oleh:

Ryan Gabriel Togar Simamora (2311104045)
Kelas S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

A. TUJUAN PRAKTIKUM

- Memahami konsep graph
- Mengimplementasikan graph dengan menggunakan pointer.

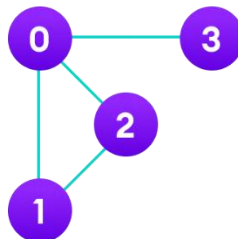
B. DASAR TEORI

Graph

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.

Sebagai contoh, terdapat graph seperti berikut :



Graph di atas terdiri atas 4 buah verteks dan 4 pasang sisi atau edge. Dengan verteks disimbolkan sebagai V, edge dilambangkan E, dan graph disimbolkan G, ilustrasi di atas dapat ditulis dalam notasi berikut :

$$V = \{0, 1, 2, 3\}$$

$$E = \{(0,1), (0,2), (0,3), (1,2)\}$$

$$G = \{V, E\}$$

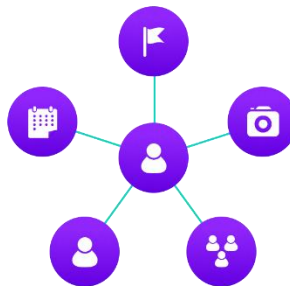
Graph banyak dimanfaatkan untuk menyelesaikan masalah dalam kehidupan nyata, dimana masalah tersebut perlu direpresentasikan atau diimajinasikan seperti sebuah jaringan. Contohnya adalah jejaring sosial (seperti Facebook, Instagram, LinkedIn, dkk)

Pengguna di Facebook dapat dimisalkan sebagai sebuah simpul atau verteks, sementara hubungan pertemanan antara pengguna tersebut dengan pengguna lain direpresentasikan sebagai edge. Tiap tiap verteks dapat berupa struktur yang mengandung informasi seperti id user, nama, gender, dll.

Tidak hanya data pengguna, data apapun yang ada di Facebook adalah sebuah simpul atau verteks. Termasuk foto, album, komentar, event, group, story, dll.

Pengguna dapat mengunggah foto. Ketika telah diunggah, foto akan menjadi bagian dari album. Foto juga dapat dikomentari oleh pengguna lain dan mereka dapat saling berbalas komentar.

Semuanya terhubung satu sama lain, baik dalam bentuk relasi one-to-many, many-to-one, atau many-to-many.

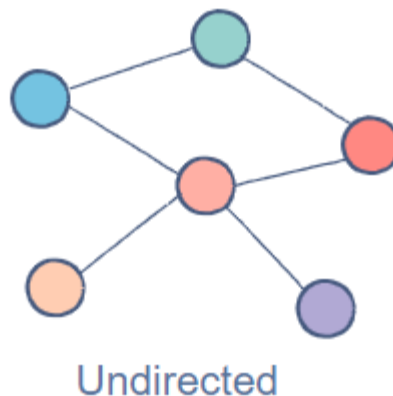


Jenis-jenis Graph

Graph dapat dibedakan berdasarkan arah jelajahnya dan ada tidaknya label bobot pada relasinya. Berdasarkan arah jelajahnya graph dibagi menjadi Undirected graph dan Directed graph.

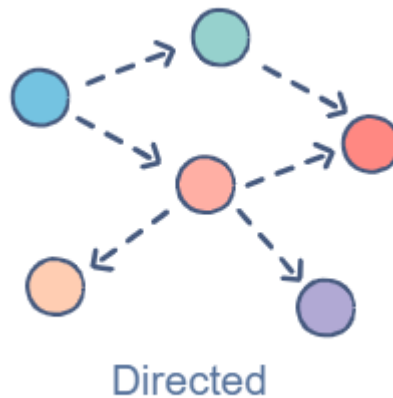
1. Undirected Graph

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.



2. 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.

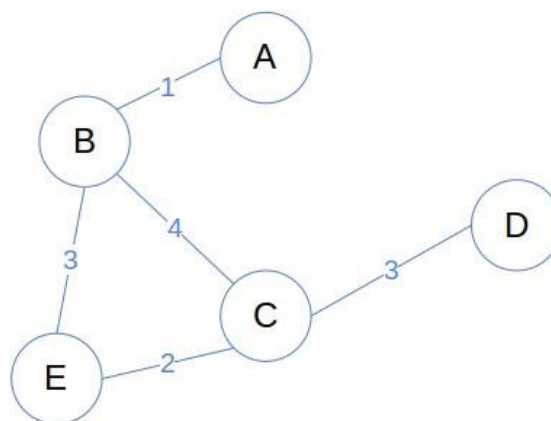


Selain arah jelajahnya, graph dapat dibagi menjadi 2 berdasarkan ada tidaknya label bobot pada koneksinya, yaitu **weighted graph** dan **unweighted graph**.

3. Weighted Graph

Weighted graph adalah jenis graph yang cabangnya diberi label bobot berupa bilangan numerik. Pemberian label bobot pada edge biasanya digunakan untuk memudahkan algoritma dalam menyelesaikan masalah.

Contoh implementasinya misalkan kita ingin menyelesaikan masalah dalam mencari rute terpendek dari lokasi A ke lokasi D, namun kita juga dituntut untuk mempertimbangkan kepadatan lalu lintas, panjang jalan dll. Untuk masalah seperti ini, kita bisa mengasosiasikan sebuah edge e dengan bobot $w(e)$ berupa bilangan ril.



Nilai bobot ini bisa apa saja yang relevan untuk masalah yang dihadapi: misalnya jarak, kepadatan, durasi, biaya, probabilitas, dan sebagainya.

4. Unweighted Graph

Berbeda dengan jenis sebelumnya, unweighted graph tidak memiliki properti bobot pada koneksinya. Graph ini hanya mempertimbangkan apakah dua node saling terhubung atau tidak.

Karakteristik Graph

Graph memiliki beberapa karakteristik sebagai berikut:

- ❖ Jarak maksimum dari sebuah simpul ke semua simpul lainnya dianggap sebagai eksentrisitas dari simpul tersebut.
- ❖ Titik yang memiliki eksentrisitas minimum dianggap sebagai titik pusat dari graph.
- ❖ Nilai eksentrisitas minimum dari semua simpul dianggap sebagai jari-jari dari graph terhubung.

Fungsi dan Kegunaan Graph

Fungsi dan kegunaan graph di antaranya:

- ❖ Graph digunakan untuk merepresentasikan aliran komputasi.
- ❖ Digunakan dalam pemodelan grafik.
- ❖ Graph dipakai pada sistem operasi untuk alokasi sumber daya.
- ❖ Google maps menggunakan graph untuk menemukan rute terpendek.
- ❖ Graph digunakan dalam sistem penerbangan untuk optimasi rute yang efektif.
- ❖ Pada state-transition diagram, graph digunakan untuk mewakili state dan transisinya.
- ❖ Di sirkuit, graph dapat digunakan untuk mewakili titik sirkuit sebagai node dan kabel sebagai edge.
- ❖ Graph digunakan dalam memecahkan teka-teki dengan hanya satu solusi, seperti labirin.
- ❖ Graph digunakan dalam jaringan komputer untuk aplikasi Peer to peer (P2P).
- ❖ Umumnya graph dalam bentuk DAG (Directed acyclic graph) digunakan sebagai alternatif blockchain untuk cryptocurrency. Misalnya crypto seperti IOTA

Kelebihan Graph

Keunggulan dari struktur data graph adalah sbb:

- ❖ Dengan menggunakan graph kita dapat dengan mudah menemukan jalur terpendek dan tetangga dari node
- ❖ Graph digunakan untuk mengimplementasikan algoritma seperti DFS dan BFS.
- ❖ Graph membantu dalam mengatur data.
- ❖ Karena strukturnya yang non-linier, membantu dalam memahami masalah yang kompleks dan visualisasinya.

Kekurangan Graph

Adapun kekurangan dari struktur data graph di antaranya

- ❖ Graph menggunakan banyak pointer yang bisa rumit untuk ditangani.
- ❖ Memiliki kompleksitas memori yang besar.
- ❖ Jika graph direpresentasikan dengan adjacency matrix maka edge tidak memungkinkan untuk sejajar dan operasi perkalian graph juga sulit dilakukan.

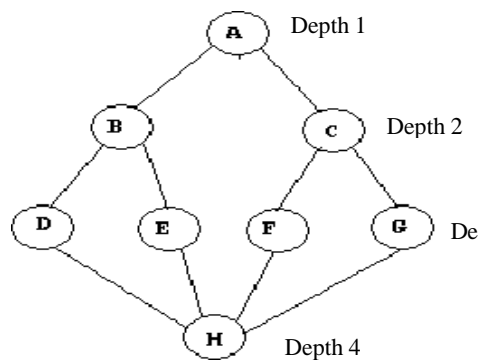
Metode-Metode Penelusuran Graph

A. 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.

Contoh :

Perhatikan *graph* berikut :

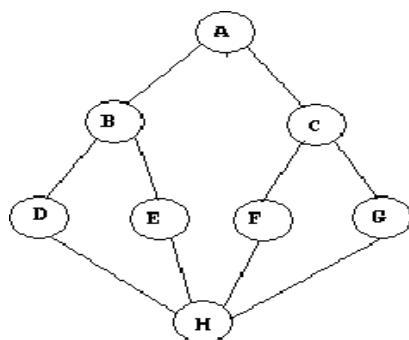


Urutannya hasil penelusuran BFS : A B C D E F G H

B. Depth First Search (DFS)

Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian rekursif ke subtree node tersebut.

Contoh :



Urutan hasil penelusuran DFS : A B D H E F C G

C. GUIDED 1

Sourcecode :

File GuidedGraph.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, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(D, H);
}

```

```

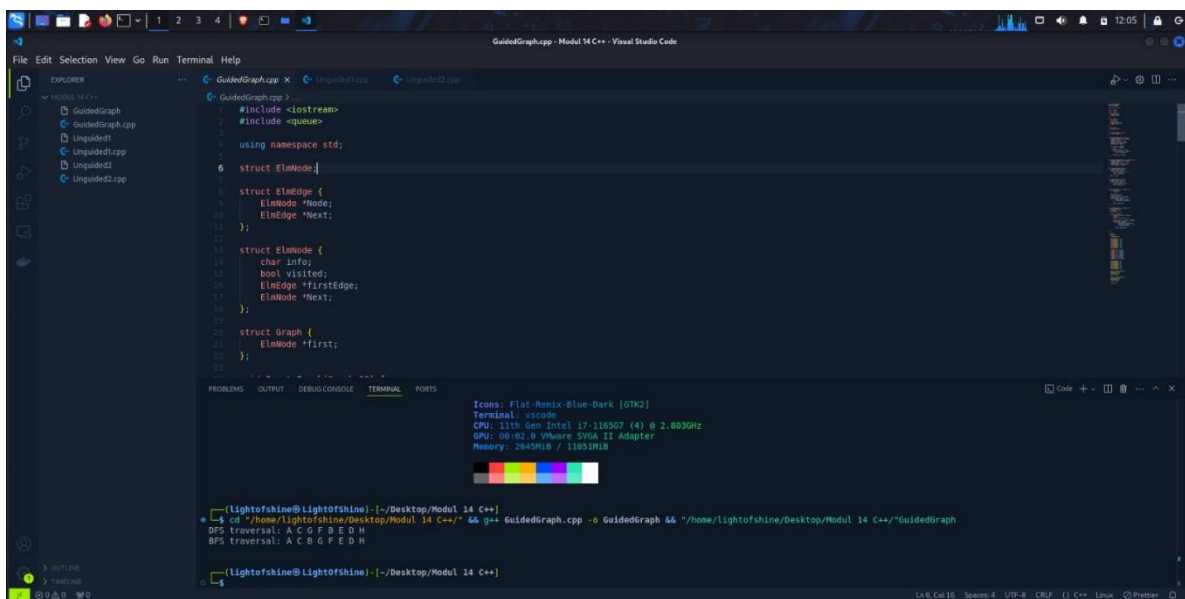
cout << "DFS traversal: ";
ResetVisited(G);
PrintDFS(G, A);
cout << endl;

cout << "BFS traversal: ";
ResetVisited(G);
PrintBFS(G, A);
cout << endl;

return 0;
}

```

Output :



```

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

struct ElNode {
    struct ElEdge {
        ElNode *Node;
        ElEdge *Next;
    };
    char info;
    bool visited;
    ElEdge *firstEdge;
    ElNode *Next;
};

struct Graph {
    ElNode *first;
};

(Lightofshime@Lightofshime)~/Desktop/Modul 14 C++
cd ~/home/lightofshime/desktop/Modul 14 C++/ - && g++ GuidedGraph.cpp -o GuidedGraph && ./GuidedGraph
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H

```

D. 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 masukan 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.
```

Jawab :

Source code :

File Unguided1.cpp

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

using namespace std;

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

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

```

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

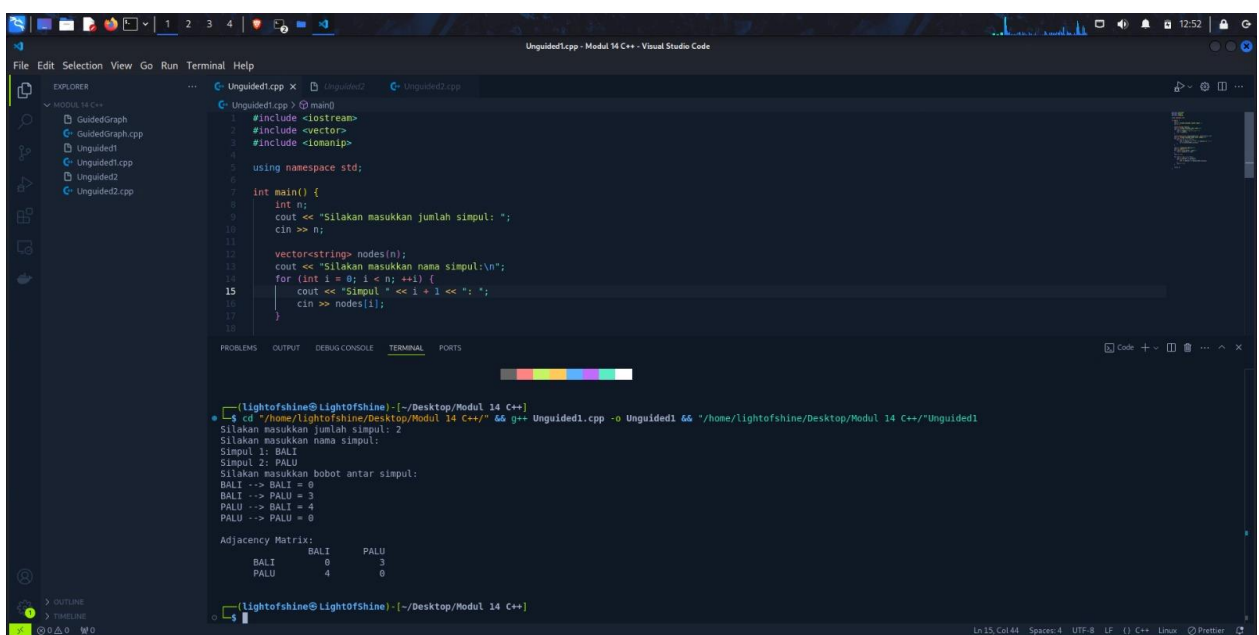
cout << "\nAdjacency Matrix:\n";
cout << setw(10) << "";
for (const string &node : nodes) {
    cout << setw(10) << node;
}
cout << "\n";

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

return 0;
}

```

Output :



```

$ cd /home/lightofshine/Desktop/Modul 14 C++/ && g++ Unguided1.cpp -o Unguided1 && ./Unguided1
Silakan masukkan jumlah simpul: 2
Silakan masukkan 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

Adjacency Matrix:
BALI      BALI      PALU
BALI      0         3
PALU      4         0

```

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

Jawab :

Source code :

File Unguided2.cpp

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

using namespace std;

// Fungsi untuk mencetak adjacency matrix
void printMatrix(const vector<vector<int>>& matrix) {
    cout << "Adjacency Matrix:\n";
    for (const auto& row : matrix) {
        for (int value : row) {
            cout << value << " ";
        }
        cout << endl;
    }
}

int main() {
    int numNodes, numEdges;
```

```
// Input jumlah simpul dan sisi
cout << "Masukkan jumlah simpul: ";
cin >> numNodes;
cout << "Masukkan jumlah sisi: ";
cin >> numEdges;

// Inisialisasi adjacency matrix dengan 0
vector<vector<int>> adjacencyMatrix(numNodes, vector<int>(numNodes, 0));

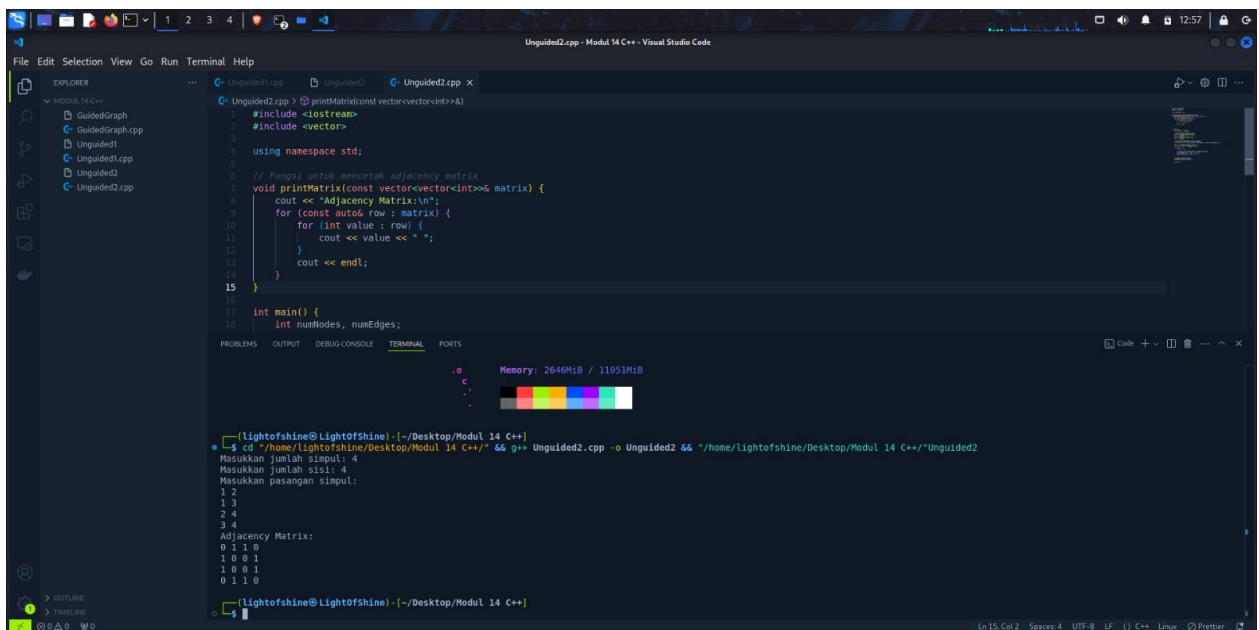
cout << "Masukkan pasangan simpul:\n";
for (int i = 0; i < numEdges; ++i) {
    int u, v;
    cin >> u >> v;

    // Karena graf tidak berarah, tandai kedua arah
    adjacencyMatrix[u - 1][v - 1] = 1;
    adjacencyMatrix[v - 1][u - 1] = 1;
}

// Cetak adjacency matrix
printMatrix(adjacencyMatrix);

return 0;
}
```

Output :



```
Unguided2.cpp - Modul 14 C++ - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  MODUL 14 C++
    GuidedGraph.cpp
    GuidedGraph.cpp
    Unguided1
    Unguided1.cpp
    Unguided2
    Unguided2.cpp
    Unguided2.cpp
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  Memory: 2640MiB / 11051MiB
  [Lightofshine@Lightofshine:~/Desktop/Modul 14 C++]
  $ cd "/home/Lightofshine/Desktop/Modul 14 C++/" && g++ Unguided2.cpp -o Unguided2 && ./Unguided2
  Masukkan jumlah simpul: 4
  Masukkan jumlah sisi: 4
  Masukkan pasangan simpul:
  1 2
  1 3
  2 4
  3 4
  Adjacency Matrix:
  0 1 0
  1 0 1
  1 0 0
  0 1 0
```

E. KESIMPULAN

Graph adalah struktur data yang merepresentasikan hubungan antara objek-objek dalam berbagai aplikasi, seperti jaringan komputer, transportasi, dan penjadwalan. Dengan komponen utama berupa simpul (node) dan sisi (edge), graph memungkinkan representasi hubungan yang kompleks, baik berarah maupun tidak berarah, berbobot maupun tidak berbobot.

Jenis-jenis representasi seperti matriks ketetanggaan, daftar ketetanggaan, dan multilist memberikan fleksibilitas untuk berbagai kebutuhan implementasi. Operasi dasar pada graph, seperti menambahkan simpul, menghubungkan simpul, dan penelusuran (BFS dan DFS), memudahkan eksplorasi dan manipulasi data yang disimpan.

Selain itu, metode seperti topological sorting digunakan untuk mengatasi masalah dependensi, menjadikan graph alat penting dalam penyelesaian berbagai permasalahan dunia nyata. Dengan algoritma dan representasi yang tepat, graph menjadi solusi komputasi yang efisien dan efektif untuk pengelolaan data terstruktur yang kompleks.

