

**LAPORAN PRAKTIKUM  
STRUKTUR DATA  
MODUL 14  
“GRAPH”**



**Disusun Oleh:**  
**Dhiya Ulhaq Ramadhan 2211104053**  
**Kelas :**  
**S1SE-07-02**  
**Dosen :**  
**Wahyu Andi Saputra, S.Pd., M.Eng.**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY  
PURWOKERTO  
2024**

## 1. Tujuan

- Memahami konsep dasar graph dan implementasinya
- Mampu mengimplementasikan graph menggunakan pointer
- Mengetahui jenis-jenis graph dan metode penelusurannya

## 2. Landasan Teori

Graph merupakan struktur data non-linear yang terdiri dari himpunan tidak kosong dari node (vertex) dan garis penghubung (edge). Sebuah graph dapat diilustrasikan seperti menghubungkan dua lokasi, misalnya antara tempat kost dengan laboratorium komputer, dimana kedua lokasi tersebut merupakan node dan jalan penghubungnya merupakan edge.

Dalam implementasinya, graph dapat dibagi menjadi dua jenis utama yaitu graph berarah (directed graph) dan graph tidak berarah (undirected graph). Graph berarah memiliki edge dengan arah tertentu yang menunjukkan hubungan antara dua node, sedangkan pada graph tidak berarah, edge yang menghubungkan antar node tidak memiliki arah tertentu.

Representasi graph dapat dilakukan dengan dua pendekatan yaitu menggunakan matriks ketetanggaan (adjacency matrices) atau multilist. Matriks ketetanggaan menggunakan array 2 dimensi untuk merepresentasikan hubungan antar node, sementara multilist menggunakan struktur linked list yang bersifat dinamis sehingga lebih fleksibel dalam pengelolaan memori.

Dalam penelusuran graph terdapat dua metode utama yaitu Breadth First Search (BFS) dan Depth First Search (DFS). BFS melakukan penelusuran dengan mengunjungi semua node pada level yang sama terlebih dahulu sebelum berpindah ke level berikutnya, dimulai dari root node. Sementara DFS melakukan penelusuran dengan menelusuri satu jalur hingga ke node terakhir sebelum melakukan backtrack untuk menelusuri jalur lainnya.

### 3. Guided

Source code :

```

1  #include <iostream>
2  #include <queue>
3
4  using namespace std;
5
6  struct ElmNode;
7
8  struct ElmEdge {
9      ElmNode *Node;
10     ElmEdge *Next;
11 };
12
13 struct ElmNode {
14     char info;
15     bool visited;
16     ElmEdge *firstEdge;
17     ElmNode *Next;
18 };
19
20 struct Graph {
21     ElmNode *first;
22 };
23
24 void CreateGraph(Graph &G) {
25     G.first = NULL;
26 }
27
28 void InsertNode(Graph &G, char X) {
29     ElmNode *newNode = new ElmNode;
30     newNode->info = X;
31     newNode->visited = false;
32     newNode->firstEdge = NULL;
33     newNode->Next = NULL;
34
35     if (G.first == NULL) {
36         G.first = newNode;
37     } else {
38         ElmNode *temp = G.first;
39         while (temp->Next != NULL) {
40             temp = temp->Next;
41         }
42         temp->Next = newNode;
43     }
44 }
45
46 void ConnectNode(ElmNode *N1, ElmNode *N2) {
47     ElmEdge *newEdge = new ElmEdge;
48     newEdge->Node = N2;
49     newEdge->Next = N1->firstEdge;
50     N1->firstEdge = newEdge;
51 }
52
53 void PrintInfoGraph(Graph G) {
54     ElmNode *temp = G.first;
55     while (temp != NULL) {
56         cout << temp->info << " ";
57         temp = temp->Next;
58     }
59     cout << endl;
60 }
61
62 void ResetVisited(Graph &G) {
63     ElmNode *temp = G.first;
64     while (temp != NULL) {
65         temp->visited = false;
66         temp = temp->Next;
67     }
68 }
69
70 void PrintDFS(Graph G, ElmNode *N) {
71     if (N == NULL) {
72         return;
73     }
74     N->visited = true;
75     cout << N->info << " ";
76     ElmEdge *edge = N->firstEdge;
77     while (edge != NULL) {
78         if (!edge->Node->visited) {

```

```

79         PrintDFS(G, edge->Node);
80     }
81     edge = edge->Next;
82 }
83 }
84
85 void PrintBFS(Graph G, ElmNode *N) {
86     queue<ElmNode*> q;
87     q.push(N);
88     N->visited = true;
89
90     while (!q.empty()) {
91         ElmNode *current = q.front();
92         q.pop();
93         cout << current->info << " ";
94
95         ElmEdge *edge = current->firstEdge;
96         while (edge != NULL) {
97             if (!edge->Node->visited) {
98                 edge->Node->visited = true;
99                 q.push(edge->Node);
100             }
101             edge = edge->Next;
102         }
103     }
104 }
105
106 int main() {
107     Graph G;
108     CreateGraph(G);
109
110     InsertNode(G, 'A');
111     InsertNode(G, 'B');
112     InsertNode(G, 'C');
113     InsertNode(G, 'D');
114     InsertNode(G, 'E');
115     InsertNode(G, 'F');
116     InsertNode(G, 'G');
117     InsertNode(G, 'H');
118
119     ElmNode *A = G.first;
120     ElmNode *B = A->Next;
121     ElmNode *C = B->Next;
122     ElmNode *D = C->Next;
123     ElmNode *E = D->Next;
124     ElmNode *F = E->Next;
125     ElmNode *G1 = F->Next;
126     ElmNode *H = G1->Next;
127
128     ConnectNode(A, B);
129     ConnectNode(A, C);
130     ConnectNode(B, D);
131     ConnectNode(B, E);
132     ConnectNode(C, F);
133     ConnectNode(C, G1);
134     ConnectNode(D, H);
135
136     cout << "DFS traversal: ";
137     ResetVisited(G);
138     PrintDFS(G, A);
139     cout << endl;
140
141     cout << "BFS traversal: ";
142     ResetVisited(G);
143     PrintBFS(G, A);
144     cout << endl;
145
146     return 0;
147 }

```

Output :

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

## UNGUIDED

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.
```

Jawaban :

Source code

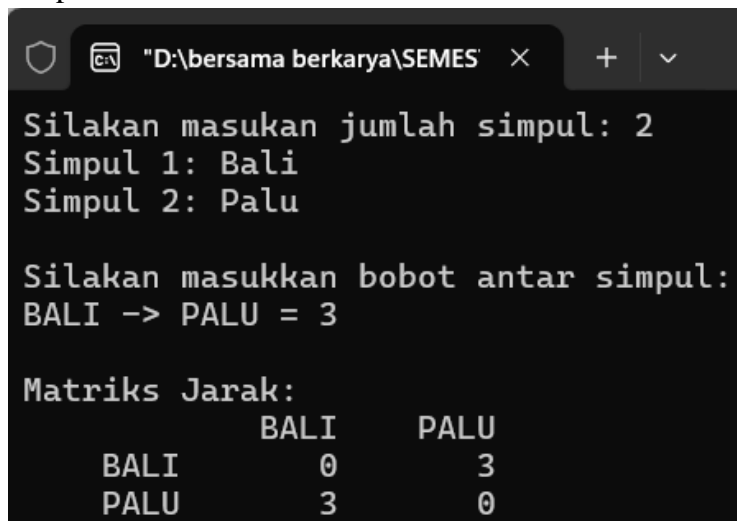
```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <map>
5  #include <iomanip>
6  #include <algorithm>
7
8  class Graph {
9  private:
10     std::vector<std::string> cities;
11     std::map<std::string, std::map<std::string, int>> distances;
12
13 public:
14     void addCity(const std::string& city) {
15         cities.push_back(city);
16         distances[city] = std::map<std::string, int>();
17         for (const auto& existingCity : cities) {
18             distances[city][existingCity] = 0;
19             distances[existingCity][city] = 0;
20         }
21     }
22
23     void addDistance(const std::string& from, const std::string& to, int distance) {
24         distances[from][to] = distance;
25         distances[to][from] = distance;
26     }
27
28     void displayMatrix() const {
29         std::cout << std::setw(8) << "";
30         for (const auto& city : cities) {
31             std::cout << std::setw(8) << city;
32         }
33         std::cout << "\n";
34
35         for (const auto& fromCity : cities) {
36             std::cout << std::setw(8) << fromCity;
37             for (const auto& toCity : cities) {
38                 std::cout << std::setw(8) << distances.at(fromCity).at(toCity);
39             }
40         }
41     }
42 }
```

```

40         std::cout << "\n";
41     }
42 }
43
44 const std::vector<std::string>& getCities() const {
45     return cities;
46 }
47 };
48
49 std::string toUpper(std::string str) {
50     std::transform(str.begin(), str.end(), str.begin(), ::toupper);
51     return str;
52 }
53
54 int main() {
55     Graph graph;
56     int numCities;
57
58     std::cout << "Silakan masukan jumlah simpul: ";
59     std::cin >> numCities;
60     std::cin.ignore();
61
62     for (int i = 1; i <= numCities; i++) {
63         std::string cityName;
64         std::cout << "Simpul " << i << ": ";
65         std::getline(std::cin, cityName);
66         graph.addCity(toUpper(cityName));
67     }
68
69     std::cout << "\nSilakan masukan bobot antar simpul:\n";
70     const auto& cities = graph.getCities();
71     for (size_t i = 0; i < cities.size(); i++) {
72         for (size_t j = i + 1; j < cities.size(); j++) {
73             int distance;
74             std::cout << cities[i] << " -> " << cities[j] << " = ";
75             std::cin >> distance;
76             graph.addDistance(cities[i], cities[j], distance);
77         }
78     }
79
80     std::cout << "\nMatriks Jarak:\n";
81     graph.displayMatrix();
82
83     return 0;
84 }

```

Output :



```

D:\bersama berkarya\SEMES
Silakan masukan jumlah simpul: 2
Simpul 1: Bali
Simpul 2: Palu

Silakan masukan bobot antar simpul:
BALI -> PALU = 3

Matriks Jarak:
          BALI  PALU
BALI      0      3
PALU      3      0

```

Penjelasan program

Class Graph: Kelas ini merupakan inti dari program yang mengelola representasi graf kota dan jaraknya. Memiliki dua data member utama:

- `vector<string> cities`: Menyimpan daftar nama-nama kota
- `map<string, map<string, int>> distances`: Struktur data 2D yang menyimpan jarak antar kota

Fungsi `addCity`:

**`void addCity(const std::string& city)`**

Fungsi ini menambahkan kota baru ke dalam graf. Saat kota ditambahkan, fungsi ini:

- Memasukkan nama kota ke dalam vector `cities`
- Membuat entri baru di map `distances` untuk kota tersebut
- Menginisialisasi jarak ke semua kota (termasuk dirinya sendiri) dengan nilai 0

Fungsi `addDistance`:

**`void addDistance(const std::string& from, const std::string& to, int distance)`**

Fungsi ini menyimpan jarak antara dua kota. Karena graf tidak berarah, jarak disimpan dua kali:

- Dari kota asal ke tujuan
- Dari kota tujuan ke asal dengan nilai yang sama

Fungsi `displayMatrix`:

**`void displayMatrix() const`**

Fungsi ini menampilkan matriks jarak dalam format tabel:

- Mencetak header dengan nama-nama kota
- Mencetak setiap baris dengan nama kota dan jaraknya ke kota lain
- Menggunakan `setw(8)` untuk format tabel yang rapi

Fungsi Utilitas `toUpper`:

**`std::string toUpper(std::string str)`**

Fungsi ini mengkonversi string input menjadi huruf kapital untuk konsistensi data

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

Jawaban :

Source code

```

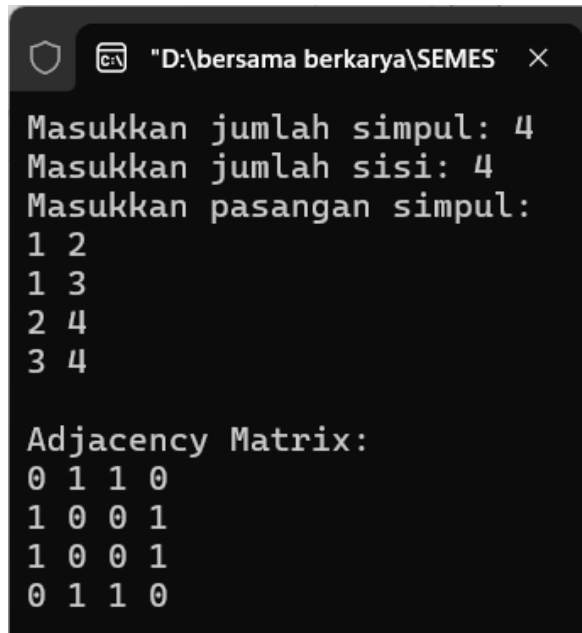
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  class Graph {
6  private:
7      vector<vector<int>> adjacencyMatrix;
8      int vertices;
9
10 public:
11     // Constructor untuk inisialisasi matrix dengan ukuran vertices x vertices
12     Graph(int v) : vertices(v) {
13         adjacencyMatrix.resize(v, vector<int>(v, 0));
14     }
15
16     // Fungsi untuk menambahkan edge antara dua vertex
17     void addEdge(int v1, int v2) {
18         // Karena graf tidak berarah, set kedua arah dengan nilai 1
19         adjacencyMatrix[v1-1][v2-1] = 1;
20         adjacencyMatrix[v2-1][v1-1] = 1;
21     }
22
23     // Fungsi untuk menampilkan adjacency matrix
24     void displayMatrix() {
25         cout << "\nAdjacency Matrix:\n";
26         for(int i = 0; i < vertices; i++) {
27             for(int j = 0; j < vertices; j++) {
28                 cout << adjacencyMatrix[i][j] << " ";
29             }
30             cout << endl;
31         }
32     }
33 };
34
35 int main() {
36     int numVertices, numEdges;
37
38     cout << "Masukkan jumlah simpul: ";
39     cin >> numVertices;

```



```
40
41     cout << "Masukkan jumlah sisi: ";
42     cin >> numEdges;
43
44     Graph graph(numVertices);
45
46     cout << "Masukkan pasangan simpul:\n";
47     for(int i = 0; i < numEdges; i++) {
48         int v1, v2;
49         cin >> v1 >> v2;
50         graph.addEdge(v1, v2);
51     }
52
53     graph.displayMatrix();
54
55     return 0;
56 }
```

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
```

Penjelasan program

Program dimulai dengan meminta input jumlah simpul (vertices) dari pengguna. Setelah itu, program meminta jumlah sisi (edges) yang akan menghubungkan antar simpul. Constructor dari class Graph akan membuat matrix dengan ukuran vertices x vertices dan menginisialisasi semua nilai dengan 0.

Selanjutnya, program akan meminta input pasangan simpul yang terhubung. Untuk setiap pasangan simpul yang diinput, fungsi addEdge() akan dipanggil. Fungsi ini mengatur nilai 1 pada matrix untuk kedua arah karena ini adalah graf tidak berarah. Misalnya jika ada edge antara simpul 1 dan 2, maka nilai matrix[0][1] dan matrix[1][0] akan diset menjadi 1.

Setelah semua input selesai, program memanggil fungsi displayMatrix() untuk menampilkan adjacency matrix. Fungsi ini menampilkan matrix dalam bentuk grid dimana nilai 1 menunjukkan ada hubungan antara dua simpul, dan nilai 0 menunjukkan tidak ada hubungan.

### **Kesimpulan**

Graph merupakan struktur data yang memiliki peran penting dalam merepresentasikan hubungan antar objek atau entitas. Implementasi graph dapat dilakukan dengan dua pendekatan utama yaitu menggunakan array 2 dimensi dan multilist. Dalam praktikum yang dibahas, pendekatan multilist lebih diutamakan karena sifatnya yang dinamis dan lebih efisien dalam penggunaan memori.

Penelusuran graph dapat dilakukan dengan dua metode yaitu Breadth First Search (BFS) dan Depth First Search (DFS). BFS melakukan penelusuran berdasarkan level (breadth) sedangkan DFS melakukan penelusuran berdasarkan kedalaman (depth) terlebih dahulu. Kedua metode ini memiliki kegunaannya masing-masing tergantung pada kasus yang dihadapi.