MODUL 14: GRAPH

1.1. Deskripsi Singkat

Graph adalah kumpulan node (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan edge (garis). Graph dapat digunakan untuk merepresentaikan objek-objek diskrit (direpresentasikan sebagai node) dan hubungan antara objek-objek tersebut(direpresentasikan sebagai edge). Secara matematis, graph dinyatakan sebagai G= (V,E) dimana G adalah Graph, V adalah Vertex atau Node atau Simpul, atau Titik dan E adalah Edge atau Busur atau Garis.

Terdapat berbagai jenis graph seperti Graph berarah (directed graph), Graph tak berarah (undirected graph), graph berbobot (weighted graph), dan graph tak berbobot (unweighted Graph). Baik Graph berarah maupun graph tak berarah dapat memiliki bobot maupun tidak memiliki bobot.

Graph dapat direpresentasikan dengan beberap cara diantaranya yaitu dengan Adjacency Matrix dan Adjacency List. Adjacency Matrix merepresentasikan graph ke dalam sebuah matriks (array 2 dimensi) dan Adjacency List merepresentasikan graph dengan sebuah array dari linked list. Pada Praktikum kali ini akan dibahas representasi graph menggunakan adjacency matrix dan adjacency list.

1.2. Tujuan Praktikum

1) Mahasiswa mampu merepresentasikan struktur data graph tak berarah tak berbobot dengan menggunakan adjacency matriks
2) Mahasiswa mampu merepresentasikan struktur data graph berarah tak berbobot dengan menggunakan adjacency list
3) Mahasiswa mampu merepresentasikan struktur data graph berarah berbobot dengan menggunakan adjacency list
4) Mahasiswa mampu memahami cara kerja program untuk graph traversal dan topological sort.

1.3. Material Praktikum

Kegiatan pada modul ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

1.4. Kegiatan Praktikum

A. Graf Tak Berarah Tak Berbobot

Adjacency matriks atau matriks ketetanggaan adalah cara untuk merepresentasikan graf sebagai matriks boolean (0 dan 1). Graf berhingga dapat direpresentasikan dalam bentuk matriks bujur sangkar di komputer, di mana nilai boolean matriks bernilai 1 menunjukkan jika

ada jalur langsung antara dua simpul. Jika Anda tahu cara membuat array dua dimensi, Anda juga tahu cara membuat matriks ketetanggaan.

Berikut ini adalah contoh program untuk merepresentasikan graph dalam adjacency matriks:

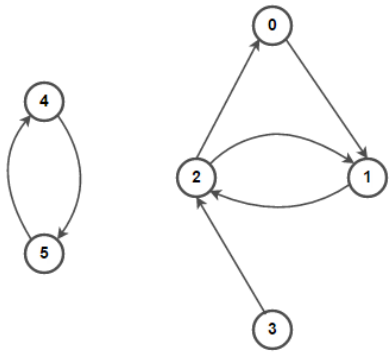
```
// Adjacency Matrix representation in C

#include <stdio.h>
#define V 4

// Initialize the matrix to zero
void init(int arr[][V]) {
  int i, j;
  for (i = 0; i < V; i++)
    for (j = 0; j < V; j++)
      arr[i][j] = 0;
}

// Add edges
void addEdge(int arr[][V], int i, int j) {
  arr[i][j] = 1;
  arr[j][i] = 1;
}

// Print the matrix
void printAdjMatrix(int arr[][V]) {
  int i, j;

  for (i = 0; i < V; i++) {
    printf("%d: ", i);
    for (j = 0; j < V; j++) {
      printf("%d ", arr[i][j]);
    }
    printf("\n");
  }
}
```

```
int main() {
  int adjMatrix[V][V];

  init(adjMatrix);
  addEdge(adjMatrix, 0, 1);
  addEdge(adjMatrix, 0, 2);
  addEdge(adjMatrix, 1, 2);
  addEdge(adjMatrix, 2, 0);
  addEdge(adjMatrix, 2, 3);

  printAdjMatrix(adjMatrix);

  return 0;
}
```

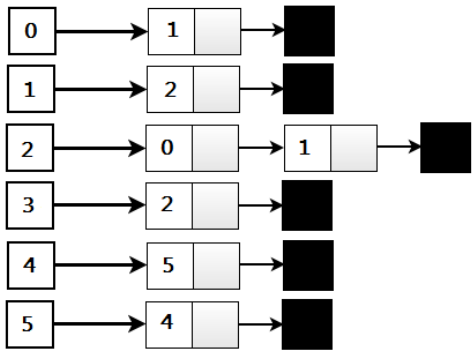
Cobalah untuk menjalankan program tersebut. Anda dapat menyimpannya dengan nama **modul14a.c**. Apakah output yang tampil pada layar Anda? Pastikan Anda memahami cara kerja program tersebut.

B. Graph Berarah Tak Berbobot

Pada representasi graph menggunakan adjacency list, setiap node pada graph diasosiasikan dengan kumpulan node lain yang terhubung dengan node tersebut. Dengan kata lain setiap node menyimpan list node lain yang terhubung. Sebagai contoh graph di bawah ini:



Memiliki gambaran representasi adjacency list sebagai berikut:



Berikut ini adalah contoh listing program untuk merepresentasikan graph tersebut. Anda dapat menyimpannya pada file **modul14b.c.**

```
#include <stdio.h>
#include <stdlib.h>
#define N 6 //misal maksimum node adalah 6

// Struktur data untuk menyimpan adjacency list dari node pada
graph
struct Node{
    int dest;
    struct Node* next;
};
typedef struct Node *ptrNode;

//Struktur data untuk menyimpan objek graph
struct Graph{
    // array pointer ke node untuk representasi adjacency list
    ptrNode head[N];
};
typedef struct Graph *ptrGraph;

// Struktur data untuk menyimpan edge graph
struct Edge {
    int src, dest;
};

// Fungsi untuk membuat adjacency list dari edge tertentu
```

```
ptrGraph createGraph(struct Edge edges[], int n){
    // alokasi memori untuk menyimpan struktur data graph
    ptrGraph graph = (ptrGraph)malloc(sizeof(struct Graph));

    // inisialiasi semua pointer head ke null
    for (int i = 0; i < N; i++) {
        graph->head[i] = NULL;
    }

    // menambahkan edge satu demi satu
    for (int i = 0; i < n; i++)
    {
        // ambil source dan destination dari node
        int src = edges[i].src;
        int dest = edges[i].dest;

        // buat node baru dari src ke dest
        ptrNode newNode = (ptrNode)malloc(sizeof(struct
Node));
        newNode->dest = dest;

        // point node baru ke head
        newNode->next = graph->head[src];

        // point head ke node baru
        graph->head[src] = newNode;
    }

    return graph;
}

// Fungsi print representasi adjacency list
void printGraph(ptrGraph graph){
    int i;
    for (i = 0; i < N; i++){
        // print node dan semua yang terhubung
```

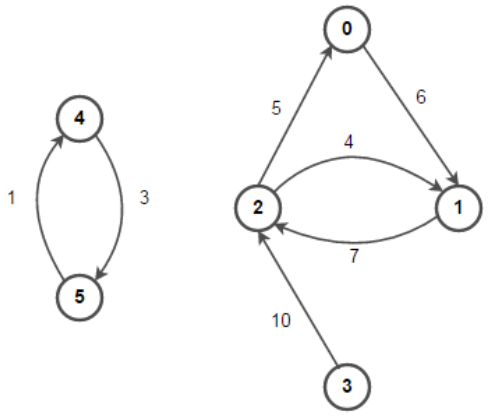
```
        ptrNode ptr = graph->head[i];
        while (ptr != NULL){
            printf("(%d -> %d)\t", i, ptr->dest);
            ptr = ptr->next;
        }

        printf("\n");
    }
}

void main(){
    //input array pasangan dari x ke y
    struct Edge edges[] =
    {{ 0, 1 }, { 1, 2 }, { 2, 0 }, { 2, 1 }, { 3, 2 }, { 4, 5
}, { 5, 4 }};

    // menghitung jumlah edge
    int n = sizeof(edges)/sizeof(edges[0]);

    // membuat graph
    ptrGraph graph = createGraph(edges, n);

    // print graph
    printGraph(graph);

}
```

Berikut adalah tampilan ketika program dijalankan:

```
0 -> 1 (6)
1 -> 2 (7)
2 -> 1 (4)      2 -> 0 (5)
3 -> 2 (10)
4 -> 5 (3)
5 -> 4 (1)

Process returned 10 (0xA)   execution time : 0.280 s
Press any key to continue.
```

C. Graph Berarah Berbobot

Pada graph berbobot, setiap edge memiliki weight. Misal kita akan mereprestasikan graph di bawah ini:



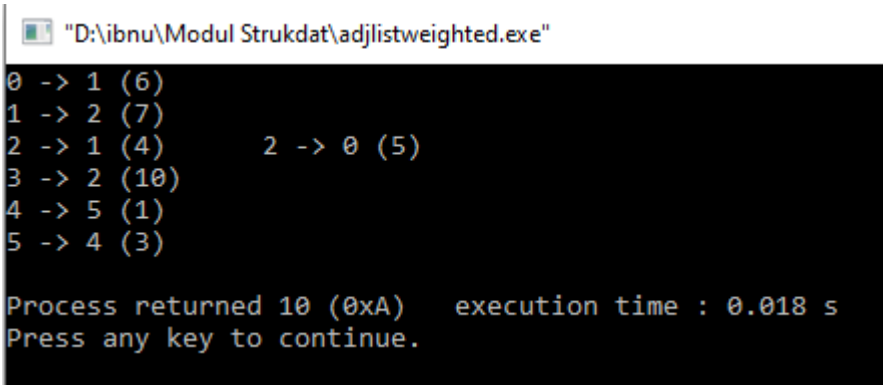
Maka pada struktur node selain kita menyimpan destination, kita juga menyimpan weight. Berikut adalah contoh implementasinya. Anda bisa menyimpannya pada file **modul13c.c.**

```
#include <stdio.h>
#include <stdlib.h>
#define N 6 //misal maksimum node adalah 6

// Struktur data untuk menyimpan adjacency list dari node pada graph
struct Node{
    int dest, weight;
    struct Node* next;
};
typedef struct Node *ptrNode;

//Struktur data untuk menyimpan onjek graph
struct Graph{
    // array pointer ke node untuk representasi adjacency list
    ptrNode head[N];
};
typedef struct Graph *ptrGraph;

// Struktur data untuk menyimpan edge graph
```

```
struct Edge {
    int src, dest, weight;
};

// Fungsi untuk membuat adjacency list dari edge tertentu
ptrGraph createGraph(struct Edge edges[], int n){
    // alokasi memori untuk menyimpan struktur data graph
    ptrGraph graph = (ptrGraph)malloc(sizeof(struct Graph));

    // inisialiasi semua pointer head ke null
    for (int i = 0; i < N; i++) {
        graph->head[i] = NULL;
    }

    // menambahkan edge satu demi satu
    for (int i = 0; i < n; i++){
        // ambil source dan destination dari node
        int src = edges[i].src;
        int dest = edges[i].dest;
        int weight = edges[i].weight;

        // buat node baru dari src ke dest
        ptrNode newNode = (ptrNode)malloc(sizeof(struct
Node));
        newNode->dest = dest;
        newNode->weight = weight;

        // point node baru ke head
        newNode->next = graph->head[src];

        // point head ke node baru
        graph->head[src] = newNode;
    }

    return graph;
}
```

```
// Fungsi print representasi adjacency list
void printGraph(ptrGraph graph){
    int i;
    for (i = 0; i < N; i++){
        // print node dan semua yang terhubung
        ptrNode ptr = graph->head[i];
        while (ptr != NULL){
            printf("%d -> %d (%d)\t", i, ptr->dest, ptr-
>weight);
            ptr = ptr->next;
        }

        printf("\n");
    }
}

void main(){
    //input array pasangan dari x ke y
    struct Edge edges[] =
    {{ 0, 1, 6 }, { 1, 2, 7 }, { 2, 0, 5 }, { 2, 1, 4 }, { 3,
2, 10 }, { 4, 5, 3 }, { 5, 4, 1 }};

    // menghitung jumlah edge
    int n = sizeof(edges)/sizeof(edges[0]);

    // membuat graph
    ptrGraph graph = createGraph(edges, n);

    // print graph
    printGraph(graph);

}
```

Berikut adalah tampilan ketika program dijalankan:



1.5. Penugasan

Terdapat beberapa cara untuk menulusuri graf dan menampilkan isinya. Diantaranya dengan cara Depth First Search (DFS) dan Breadth First Search (BFS). Pada sesi teori kita telah mempelajari cara kerja masing-masing algoritma traversal tersebut. DFS dapat diimplementasikan secara rekursif atau secara iteratif dengan menggunakan bantuan stack. BFS pun dapat diimplementasikan secara rekursif, dan dapat juga secara iteratif dengan menggunakan bantuan queue.

Bentuklah kelompok, lalu pelajari dan diskusikan bagaimana cara mengimplementasikan DFS dan BFS pada adjacency matrix dan adjacency list. Sebagai bahan diskusi, Anda boleh menggunakan source code yang terdapat pada tautan di bawah ini. Anda juga boleh menggunakan pseudocode/source code lain yang terdapat pada buku teks, internet, atau referensi lainnya.

DFS pada Adjacency Matrix:

- https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal_in_c.htm
- https://www.geeksforgeeks.org/implementation-of-dfs-using-adjacency-matrix/

BFS pada Adjacency Matrix:

- https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal_in_c.htm

DFS pada Adjacency List:

- https://www.programiz.com/dsa/graph-dfs

BFS pada Adjacency List:

- https://www.programiz.com/dsa/graph-bfs