MINGGU 10
Graphs

DESKRIPSI TEMA

A. Directed dan Undirected Graph

Directed Graph adalah sebuah graph yang memiliki arah pada node-nya. Sedangkan **Undirected Graph** adalah graph yang tidak memiliki arah pada node-nya.

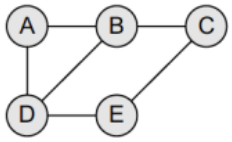


Figure 13.1 Undirected graph

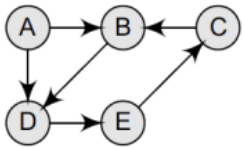


Figure 13.2 Directed graph

Pada contoh di atas, untuk Undirected Graph, B dapat menuju ke A, begitu juga sebaliknya. Sedangkan pada contoh Directed Graph, A dapat berkunjung ke B sedangkan B tidak dapat berkunjung ke A.

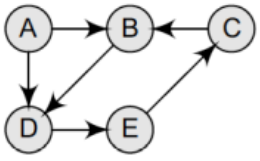
Terdapat 3 istilah dalam Directed Graph, yaitu:

- Out-degree : Jumlah egde (arah/garis) yang keluar dari sebuah node
- In-degree : Jumlah edge (arah/garis) yang masuk ke dalam node
- Degree : Jumlah edge yang terdapat dalam node. Dapat dihitung dengan Out-degree + In-degree

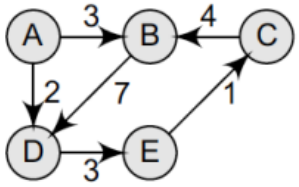
B. Weighted dan Unweighted Graph

Weighted Graph adalah graph yang memiliki beban/angka pada egde-nya. Sedangkan Unweighted Graph adalah graph yang tidak memiliki nilai beban/angka pada edge-nya.

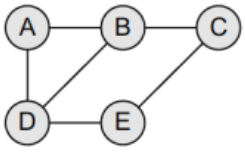
(A)



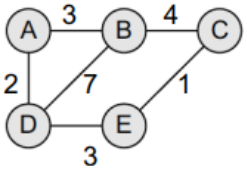
(B)



(C)



(D)



Pada contoh graph (A) dan (C) merupakan sebuah Unweighted Directed dan Undirected Graph. Sedangkan pada contoh graph (B) dan (D) merupakan sebuah Weighted Directed dan Undirected Graph. A ke B pada (B) dan (D) memiliki jarak 3. Namun pada (B), B ke A tidak memiliki jarak 3 karena panah pada B tidak mengarah ke A.

C. Graph Traversal Algorithm

Terdapat 2 cara dalam melakukan traversal pada sebuah graph, yaitu Breadth-first Search dan Depth-first Search. Breadth-first Search (BFS) adalah algoritma pencarian graph yang pencariannya dimulai dari node root dan menyebar ke node sekitarnya. BFS menggunakan queue saat melakukan pencarian. Depth-first Search (DFS) adalah algoritma pencarian graph yang pencariannya dimulai dari sebuah node lalu lebih dalam ke node di bawahnya. DFS menggunakan stack.

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

1. Mahasiswa mampu mengimplementasikan BFS menggunakan Bahasa C.
2. Mahasiswa mampu mengimplementasikan DFS menggunakan Bahasa C.

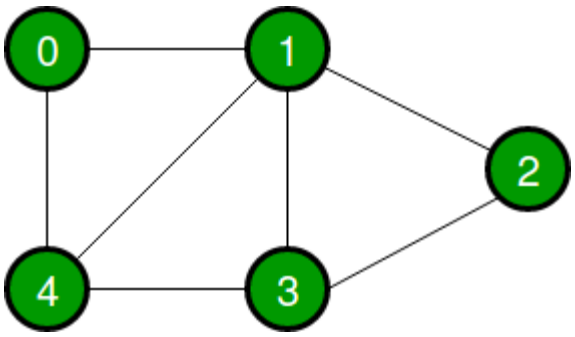
PENUNJANG PRAKTIKUM

1. CodeBlocks
2. Dev-C++ (alternatif)

LANGKAH-LANGKAH PRAKTIKUM

A. Graph Creation

Terdapat 2 cara dalam membuat graph. Cara pertama adalah dengan menggunakan Adjacency Matrix (2D Array) dan cara kedua adalah dengan menggunakan Adjacency List (Linked List). Perhatikan graph berikut ini.

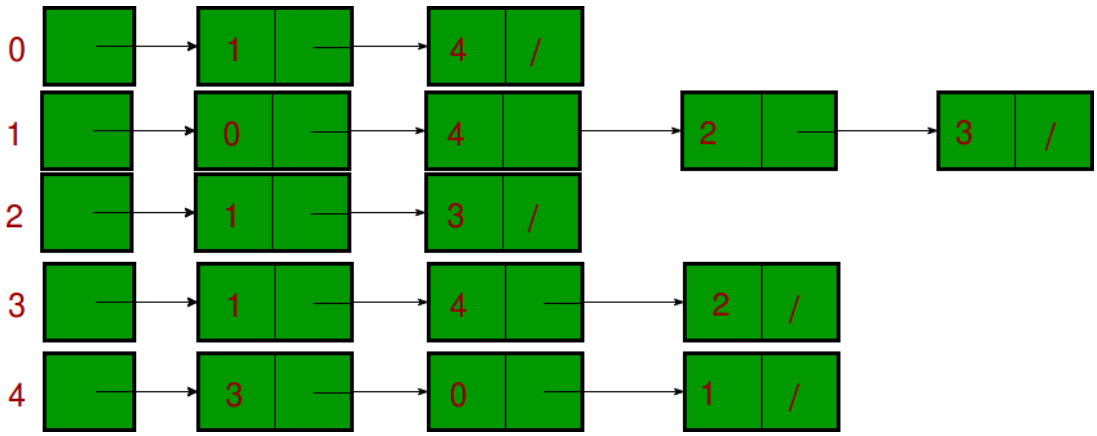


Graph di atas adalah Unweighted Undirected Graph. Jika digambarkan menggunakan Adjacency Matrix, maka isi dari Matrix-nya adalah sebagai berikut.

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Angka 1 pada matriks tersebut mengindikasikan hubungan antar node.

Jika graph di atas digambarkan menggunkan Adjacency List, maka gambaran dari List-nya adalah sebagai berikut.



B. Breadth-first Search

Tutorial 1.1 – BFS

1) Buatlah sebuah file dengan nama W10_BFS.c
2) Salin potongan code berikut. Perhatikan juga nomor baris dan komentar di dalamnya (jika ada).

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Edge{
    int dest;
    struct Edge *next;
} Edge;

typedef struct Queue {
    int data;
    struct Queue *next;
} Queue;

void addEdge(Edge *adjList[], int src, int dest){
    Edge *temp;

    temp = (Edge*) malloc(sizeof(Edge));
    temp->dest = dest;
    temp->next = NULL;

    if(adjList[src] == NULL){
        adjList[src] = temp;
    } else {
        Edge *ptr = adjList[src];
        while(ptr->next != NULL){
            ptr = ptr->next;
        }
        ptr->next = temp;
    }
}
```

```c
int isEmpty(Queue *queue){
  if (queue == NULL) return 1;
  return 0;
}

void enqueue(Queue **queue, int start){
  Queue *data = (Queue*) malloc(sizeof(Queue));
  data->data = start;
  data->next = NULL;

  if (isEmpty(*queue)) *queue = data;
  else{
    Queue *temp = *queue;
    while(temp->next != NULL){
      temp = temp->next;
    }
    temp->next = data;
  }
}

void dequeue(Queue **head){
  if(isEmpty(*head)){
    return;
  }

  Queue *trash = *head;
  *head = trash->next;
  free(trash);
}

int front(Queue *queue){
  if (queue == NULL) return 0;
  return queue->data;
}
```

```c
void BFS(int start, int visited[], Edge *adjList[]){
  Queue *queue = NULL;

  // Tandai vertex start-nya sudah dikunjungi, lalu masukkan ke queue
  visited[start] = 1;
  enqueue(&queue, start);

  while(!isEmpty(queue)){
    // Print lalu buang vertex dari queue
    int v = front(queue);
    printf("%d ", v);
    dequeue(&queue);

    // Dapatkan semua adjacent vertex yang sudah di dequeue
    // Kalau belum di visit, tandai visited-nya dan di enqueue
    Edge *it;
    for(it = adjList[v]; it!=NULL; it=it->next){
      if (!visited[it->dest]){
        visited[it->dest] = 1;
        enqueue(&queue, it->dest);
      }
    }
  }
}
```

```c
int main(){
  int i;
  int src, dest;
  int start;

  int V; // Jumlah vertex dalam graph
  // Inisialisasi graph
  printf("Jumlah vertex = "); scanf("%d", &V);
  Edge *adjList[V]; // Representasi Graph dengan Adjacency List

  for(i=0; i<V; i++){
    adjList[i] = NULL;
  }

  i = 0;
  while(1){
    printf("Adjacency List ke-%d\n", ++i);
    printf("Source: "); scanf("%d", &src);
    printf("Destination: "); scanf("%d", &dest);
    printf("\n");

    if(src <= -1 || dest <= -1) break;
    else {
      addEdge(adjList, src, dest);
      addEdge(adjList, dest, src);
    }
  }

  // Inisialisasi nilai visited
  int visited[V+2];
  for(i=0; i<V+2; i++){
    visited[i] = 0;
  }

  // Mulai BFS
  printf("Starting node: "); scanf("%d", &start);
  BFS(start, visited, adjList);

  return 0;
}
```

C. Depth-first Search

Tutorial 2.1 – DFS

1) Buatlah sebuah file dengan nama W10_DFS.c
2) Salin potongan code berikut. Perhatikan juga nomor baris dan komentar di dalamnya (jika ada).

```c
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

typedef struct Edge{
  int dest;
  struct Edge *next;
} Edge;

void addEdge(Edge *adjList[], int src, int dest){
  Edge *temp;

  temp = (Edge*) malloc(sizeof(Edge));
  temp->dest = dest;
  temp->next = NULL;

  if(adjList[src] == NULL){
    adjList[src] = temp;
  } else {
    Edge *ptr = adjList[src];
    while(ptr->next != NULL){
      ptr = ptr->next;
    }
    ptr->next = temp;
  }
}

void DFS(int v, int visited[], Edge *adjList[]){
  int i, j;

  visited[v] = 1;
  printf("%d ", v);

  Edge *it;
  for(it = adjList[v]; it!=NULL; it=it->next){
    if (!visited[it->dest])
      DFS(it->dest, visited, adjList);
  }
}
```

```c
int main(){
  int i;
  int src, dest;
  int start;

  int V; // Jumlah vertex dalam graph
  // Inisialisasi graph
  printf("Jumlah vertex = "); scanf("%d", &V);
  Edge *adjList[V]; // Representasi Graph dengan Adjacency List

  for(i=0; i<V; i++){
    adjList[i] = NULL;
  }

  i = 0;
  while(1){
    printf("Adjacency List ke-%d\n", ++i);
    printf("Source: "); scanf("%d", &src);
    printf("Destination: "); scanf("%d", &dest);
    printf("\n");

    if(src <= -1 || dest <= -1) break;
    else {
      addEdge(adjList, src, dest);
      addEdge(adjList, dest, src);
    }
  }

  // Inisialisasi nilai visited
  int visited[V+2];
  for(i=0; i<V+2; i++){
    visited[i] = 0;
  }

  // Mulai DFS
  printf("Starting node: "); scanf("%d", &start);
  DFS(start, visited, adjList);

  return 0;
}
```

REFERENSI