TUGAS JOBSHEET 7

Dosen pengampu :

Randi Proska Sandra, M.Sc

Disusun Oleh:

Rendi Aigo Brandon

NIM : 23343082

PROGRAM STUDI

INFORMATIKA(NK) DEPARTEMEN

ELEKTRONIKA FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

Source Code

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

//Created By Rendi Aigo Brandon_23343082

#define MAX_VERTICES 100

// Definisi struktur Queue
struct Queue {
    int items[MAX_VERTICES];
    int front;
    int rear;
};

// Fungsi untuk membuat queue baru
struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

// Fungsi untuk memeriksa apakah queue kosong
int isEmpty(struct Queue* q) {
    return q->rear == -1;
}

// Fungsi untuk menambahkan elemen ke dalam queue
void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX_VERTICES - 1)
        printf("Queue is full\n");
```

```
    else {
        if (q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}

// Fungsi untuk menghapus elemen dari queue
int dequeue(struct Queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        item = -1;
    } else {
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
    }
    return item;
}

// Definisi struktur Node
struct Node {
    int vertex;
    struct Node* next;
};

// Fungsi untuk membuat node baru
struct Node* createNode(int v) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->vertex = v;
```

```
    newNode->next = NULL;
    return newNode;
}

// Definisi struktur Graph
struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};

// Fungsi untuk membuat graph baru
struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

    // Mengalokasikan memori untuk array adjacency lists dan array visited
    graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));
    graph->visited = (int*)malloc(vertices * sizeof(int));

    // Menginisialisasi adjacency lists dan array visited
    int i;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0; // Set semua simpul sebagai belum dikunjungi
    }

    return graph;
}

// Fungsi untuk menambahkan edge (sisi) antara dua simpul dalam graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Tambahkan dest ke adjacency list dari src
    struct Node* newNode = createNode(dest);
```

```
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Jika graph tidak berarah, tambahkan juga edge dari dest ke src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Fungsi untuk melakukan Breadth First Search (BFS)
void bfs(struct Graph* graph, int startVertex) {
    // Membuat queue baru untuk menyimpan simpul yang akan dikunjungi
    struct Queue* q = createQueue();

    // Tandai startVertex sebagai telah dikunjungi dan tambahkan ke queue
    graph->visited[startVertex] = 1;
    enqueue(q, startVertex);

    // Melakukan penelusuran selama queue tidak kosong
    while (!isEmpty(q)) {
        // Mengambil simpul yang pertama kali dimasukkan ke dalam queue
        int currentVertex = dequeue(q);
        printf("Visited %d\n", currentVertex);

        // Menelusuri semua simpul yang bertetanggaan dengan currentVertex
        struct Node* temp = graph->adjLists[currentVertex];
        while (temp) {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
                // Jika simpul belum dikunjungi, tandai sebagai telah dikunjungi
dan masukkan ke queue
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
```

```
            temp = temp->next;
        }
    }
}

int main() {
    // Membuat graph baru dengan 6 simpul (vertices)
    struct Graph* graph = createGraph(6);

    // Menambahkan edge (sisi) antara simpul-simpul dalam graph
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);
    addEdge(graph, 3, 5);
    addEdge(graph, 4, 5);

    // Menampilkan hasil penelusuran BFS dimulai dari simpul 0
    printf("BFS traversal starting from vertex 0:\n");
    bfs(graph, 0);

    return 0;
}
```

Penjelasan :

1. Algoritma Breadth First Search (BFS):

- BFS adalah algoritma pencarian graf yang digunakan untuk menemukan semua simpul atau node pada level tertentu dari sebuah graf.
- Algoritma ini melakukan pencarian secara bertahap, dimulai dari simpul awal (start vertex), kemudian menelusuri semua simpul yang bertetanggaan dengan simpul tersebut sebelum melanjutkan ke simpul-simpul yang lebih jauh.

- Dalam pencarian ini, setiap simpul dikunjungi tepat satu kali, sehingga mencegah terjadinya pengulangan dalam penelusuran.

2. Prinsip Queue dalam BFS:

- Queue digunakan dalam BFS untuk menyimpan simpul-simpul yang akan dikunjungi.
- Prinsip FIFO (First In First Out) pada queue memastikan bahwa simpul-simpul akan dikunjungi berdasarkan urutan masuknya ke dalam antrian.
- Saat menjelajahi graf, simpul yang sudah dikunjungi ditandai sebagai telah "dikunjungi" untuk menghindari pengulangan, kemudian simpul-simpul yang bertetangga dengan simpul yang sedang dikunjungi dimasukkan ke dalam queue.
- Proses penelusuran dimulai dari simpul awal, lalu berlanjut dengan menelusuri semua simpul yang bertetanggaan dengan simpul tersebut sebelum melanjutkan ke simpul-simpul yang lebih jauh.
- Simpul yang telah dikunjungi secara berurutan dihapus dari queue setelah dilakukan penelusuran.

3. Implementasi Prinsip Queue dalam BFS:

- Dalam implementasi, queue biasanya direpresentasikan sebagai struktur data yang terdiri dari elemen-elemen yang tersusun secara linier, dengan dua operasi utama: enqueue (penambahan elemen) dan dequeue (penghapusan elemen).
- Saat melakukan penelusuran BFS, simpul-simpul yang akan dikunjungi dimasukkan ke dalam queue menggunakan operasi enqueue.
- Setiap kali simpul dikunjungi, simpul tersebut dihapus dari depan queue menggunakan operasi dequeue, dan kemudian simpul-simpul yang bertetangga dengannya dimasukkan ke dalam queue.
- Proses ini berlanjut sampai tidak ada lagi simpul yang tersisa di dalam queue atau semua simpul telah dikunjungi.

Implementasi Dalam Program :

1. Struktur Data Queue:

- Untuk menyimpan simpul-simpul yang akan dikunjungi secara berurutan, kita menggunakan struktur data queue.
- Queue didefinisikan dengan struktur `struct Queue` yang memiliki array `items` untuk menyimpan elemen-elemen queue, serta dua variabel `front` dan `rear` untuk menunjukkan posisi depan dan belakang queue.
- Fungsi-fungsi yang terkait dengan queue, seperti `createQueue(), isEmpty(), enqueue(),` dan `dequeue(),` telah diimplementasikan.

2. Struktur Data Node:

- Untuk merepresentasikan simpul-simpul dalam graf, kita menggunakan struktur data node.
- Setiap node memiliki dua bagian, yaitu `vertex` yang menyimpan nomor simpul dan `next` yang merupakan pointer ke simpul berikutnya dalam adjacency list.
- Struktur `struct Node` digunakan untuk merepresentasikan simpul-simpul tersebut.

3. Struktur Data Graph:

- Graf direpresentasikan menggunakan struktur data `struct Graph.`
- Struktur ini memiliki atribut `numVertices` yang menyimpan jumlah total simpul dalam graf, array `adjLists` yang menyimpan adjacency list untuk setiap simpul, serta array `visited` yang digunakan untuk menandai simpul mana yang sudah dikunjungi.

4. Fungsi-fungsi Utilitas:

- Terdapat beberapa fungsi utilitas seperti `createNode()` untuk membuat node baru, `createGraph()` untuk membuat graf baru, dan `addEdge()` untuk menambahkan edge (sisi) antara simpul-simpul dalam graf.

5. Algoritma BFS:

- Fungsi `bfs()` merupakan implementasi dari algoritma BFS.
- Algoritma ini melakukan penelusuran graf dimulai dari simpul awal (start vertex).
- Saat menelusuri graf, simpul-simpul yang bertetanggaan dengan simpul saat ini dimasukkan ke dalam queue.
- Setiap simpul yang dikunjungi ditandai sebagai telah dikunjungi dalam array `visited.`
- Proses penelusuran berlanjut sampai tidak ada lagi simpul yang tersisa dalam queue.

6. Main Function:

- Pada fungsi `main()`, sebuah graf baru dibuat dengan beberapa simpul dan edge yang telah ditentukan.
- Fungsi `bfs()` kemudian dipanggil untuk melakukan penelusuran BFS, dimulai dari simpul 0.
- Hasil penelusuran BFS ditampilkan ke layar.