TUGAS JOBSHEET 7

Disusun Oleh:

Qadrian Zakhri

23343081

Dosen pengampu :

Randi Proska Sandra, M.Sc

PROGRAM STUDI

INFORMATIKA(NK) DEPARTEMEN

ELEKTRONIKA FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

Source Code

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

// Struktur untuk
antrian (queue) dalam
BFS
typedef struct Queue {
    int front, rear,
size;
    unsigned capacity;
    int* array;
} Queue;

// Fungsi untuk
membuat antrian dengan
kapasitas tertentu
Queue*
createQueue(unsigned
capacity) {
    Queue* queue =
(Queue*)malloc(sizeof(
Queue));
    queue->capacity =
capacity;
    queue->front =
queue->size = 0;
    queue->rear =
capacity - 1;
    queue->array =
(int*)malloc(queue-
>capacity *
sizeof(int));
    return queue;
}
```

```
// Fungsi untuk
memeriksa apakah
antrian penuh
int isFull(Queue*
queue) {
    return (queue-
>size == queue-
>capacity);
}

// Fungsi untuk
memeriksa apakah
antrian kosong
int isEmpty(Queue*
queue) {
    return (queue-
>size == 0);
}

// Fungsi untuk
menambahkan elemen ke
antrian
void enqueue(Queue*
queue, int item) {
    if (isFull(queue))
        return;
    queue->rear =
(queue->rear + 1) %
queue->capacity;
    queue-
>array[queue->rear] =
item;
    queue->size =
queue->size + 1;
}
```

```
// Fungsi untuk
menghapus elemen dari
antrian
int dequeue(Queue*
queue) {
    if
(isEmpty(queue))
        return -1;
    int item = queue-
>array[queue->front];
    queue->front =
(queue->front + 1) %
queue->capacity;
    queue->size =
queue->size - 1;
    return item;
}

// Fungsi untuk
melakukan BFS pada
grafik
void BFS(int
graph[][5], int
startVertex, int
vertices) {
    int* visited =
(int*)malloc(vertices
* sizeof(int));
    for (int i = 0; i
< vertices; i++)
        visited[i] =
0;

    Queue* queue =
createQueue(vertices);
```

```
visited[startVertex] =
1;
    enqueue(queue,
startVertex);

    while
(!isEmpty(queue)) {
        int
currentVertex =
dequeue(queue);

printf("Visited %d\n",
currentVertex);

        for (int i =
0; i < vertices; i++)
{
            if
(graph[currentVertex][
i] == 1 &&
!visited[i]) {

visited[i] = 1;

enqueue(queue, i);
            }
        }
    }

    free(visited);
    free(queue-
>array);
    free(queue);
}
```

```
int main() {
    int graph[5][5] =
{
        {0, 1, 1, 0,
0},
        {1, 0, 1, 1,
0},
        {1, 1, 0, 1,
1},
        {0, 1, 1, 0,
1},
        {0, 0, 1, 1,
0}
    };

    BFS(graph, 0, 5);

    return 0;
}
```

Algoritma Breadth-First Search (BFS)

BFS adalah algoritma untuk penelusuran atau pencarian dalam graf atau pohon. BFS dimulai dari suatu simpul (vertex) dan menjelajah tetangganya terlebih dahulu sebelum berpindah ke simpul pada level berikutnya. BFS sering digunakan untuk mencari jalur terpendek dalam graf yang tidak berbobot dan dalam berbagai aplikasi lainnya seperti menemukan konektivitas komponen dalam graf, pemodelan jaringan, dll.

Prinsip Queue dalam Algoritma BFS Queue adalah struktur data yang mengikuti prinsip First-In-First-Out (FIFO). Dalam konteks BFS, queue digunakan untuk menyimpan simpul yang akan dijelajahi selanjutnya. Berikut adalah prinsip kerja queue dalam BFS:

1. Enqueue: Ketika BFS dimulai dari simpul awal, simpul tersebut ditandai sebagai dikunjungi dan dimasukkan ke dalam queue.
2. Dequeue: Ambil simpul dari depan queue, jelajahi semua tetangganya yang belum dikunjungi, tandai sebagai dikunjungi, dan masukkan mereka ke dalam queue.
3. Repeat: Ulangi langkah dequeue dan enqueue sampai queue kosong, yang berarti semua simpul yang terhubung telah dijelajahi.

Implementasi Breadth-First Search (BFS) pada kode

1. Inisialisasi Struktur Data:
   - Graf: Graf direpresentasikan sebagai matriks ketetanggaan (adjacency matrix). Matriks ini menunjukkan hubungan antara simpul-simpul di dalam graf, di mana elemen `graph[i][j]` adalah 1 jika ada sisi (edge) yang menghubungkan simpul `i` dan simpul `j`.
   - Queue: Digunakan untuk melacak simpul-simpul yang perlu dijelajahi. Antrian ini diimplementasikan sebagai struktur dengan fungsi untuk menambah (enqueue) dan menghapus (dequeue) elemen.

2. Inisialisasi Status Kunjungan:
   - Array `visited` digunakan untuk melacak simpul-simpul mana yang telah dikunjungi. Semua elemen diinisialisasi dengan nilai 0 (belum dikunjungi).

3. Proses BFS:
   - BFS dimulai dari simpul awal (`startVertex`). Simpul ini ditandai sebagai dikunjungi dan dimasukkan ke dalam antrian.
   - Sebuah loop digunakan untuk terus menjalankan BFS selama antrian tidak kosong.
   - Di dalam loop, simpul diambil dari depan antrian (dequeue). Simpul ini dianggap sebagai simpul saat ini (`currentVertex`).
   - Tetangga-tetangga dari `currentVertex` diperiksa. Jika ada tetangga yang belum dikunjungi dan ada sisi yang menghubungkannya dengan `currentVertex`, tetangga tersebut ditandai sebagai dikunjungi dan dimasukkan ke dalam antrian.

4. Penjelajahan Tetangga:
   - Setiap tetangga dari simpul saat ini diperiksa menggunakan loop. Jika tetangga belum dikunjungi, ia ditandai sebagai dikunjungi dan dimasukkan ke dalam antrian untuk dijelajahi pada iterasi berikutnya.