

Tugas Jobsheet 7



Dosen pengampu : Randi Proska Sandra, M.Sc

Kode Kelas : 202313430152

Disusun Oleh :

**Fajrul Huda Ash Shiddiq
23343063**

**PROGRAM STUDI INFORMATIKA (NK)
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2023**

1. Source Code

```
// Created by_Fajrul Huda Ash Shiddiq_23343063
#include <stdio.h>
#include <stdlib.h>

// Maksimum jumlah simpul/graf
#define MAX_NODES 100

// Struktur simpul
struct Node {
    int data;
    struct Node* next;
};

// Struktur untuk antrian simpul
struct Queue {
    struct Node *front, *rear;
};

// Inisialisasi antrian kosong
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct
Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Mengecek apakah antrian kosong
int isEmpty(struct Queue* queue) {
    return queue->front == NULL;
}
```

```

// Menambah simpul ke antrian
void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = data;
    newNode->next = NULL;
    if (isEmpty(queue))
        queue->front = newNode;
    else
        queue->rear->next = newNode;
    queue->rear = newNode;
}

// Menghapus simpul dari antrian
int dequeue(struct Queue* queue) {
    if (isEmpty(queue))
        return -1;
    struct Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;
    free(temp);
    return data;
}

// Fungsi untuk BFS
void BFS(int graph[MAX_NODES][MAX_NODES], int vertices, int start) {
    // Array untuk menandai apakah simpul telah dikunjungi atau
    belum
    int visited[MAX_NODES] = {0};

    // Inisialisasi antrian
    struct Queue* queue = createQueue();

```

```
    // Tandai simpul awal sebagai telah dikunjungi dan tambahkan ke antrian
```

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

```
    // Loop hingga antrian kosong
```

```
    while (!isEmpty(queue)) {
        // Ambil simpul dari antrian
        int current = dequeue(queue);
        printf("%d ", current);
```

```
        // Periksa semua simpul yang terhubung dengan simpul saat ini
```

```
        for (int i = 0; i < vertices; i++) {
            if (graph[current][i] && !visited[i]) {
                // Tandai simpul yang terhubung dan tambahkan ke antrian
                visited[i] = 1;
                enqueue(queue, i);
            }
        }
    }
```

```
    // Bebaskan memori antrian
    free(queue);
}
```

```
int main() {
    int vertices, edges, start;
    printf("Masukkan jumlah simpul: ");
    scanf("%d", &vertices);
    printf("Masukkan jumlah sisi: ");
```

```

scanf("%d", &edges);

// Matriks adjasensi untuk merepresentasikan graf
int graph[MAX_NODES][MAX_NODES] = {0};

// Baca sisi-sisi
printf("Masukkan sisi-sisi (contoh: simpul1 simpul2): \n");
for (int i = 0; i < edges; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    // Tandai bahwa simpul u dan v terhubung
    graph[u][v] = 1;
    graph[v][u] = 1; // Jika graf tidak berarah
}

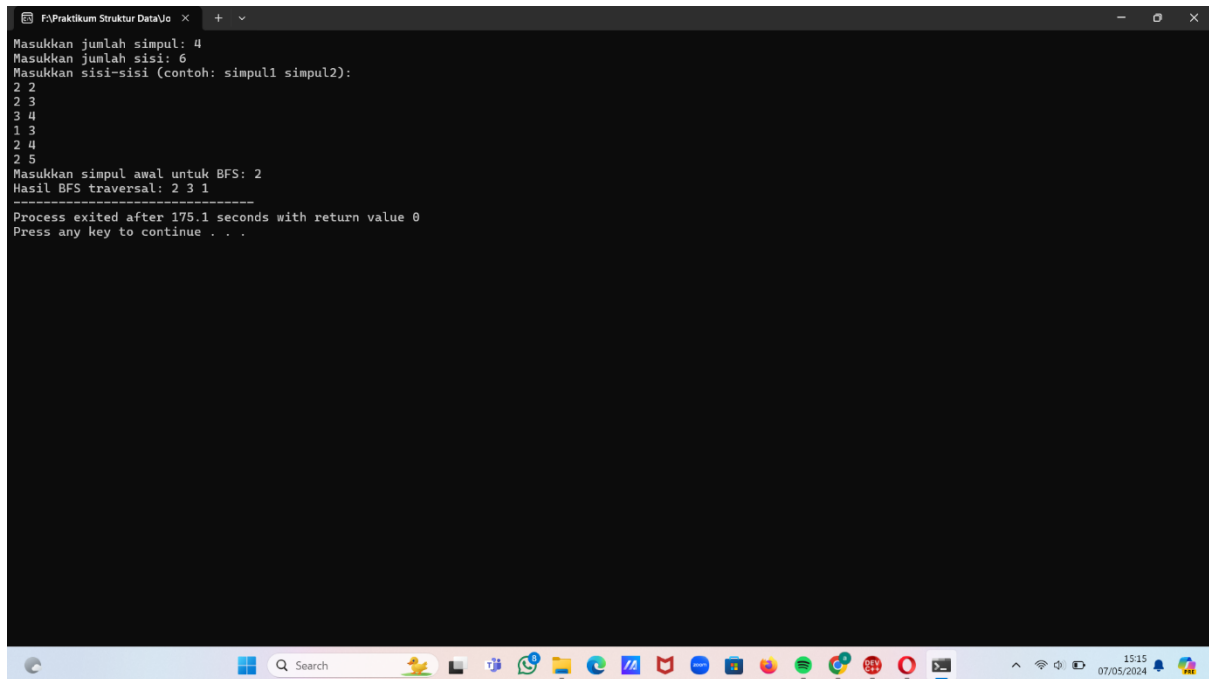
printf("Masukkan simpul awal untuk BFS: ");
scanf("%d", &start);

printf("Hasil BFS traversal: ");
BFS(graph, vertices, start);

return 0;
}

```

2. Screenshot



```
F:\Praktikum Struktur Data\Jo x + v
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 6
Masukkan sisi-sisi (contoh: simpul1 simpul2):
2 2
2 3
3 4
1 3
2 4
2 5
Masukkan simpul awal untuk BFS: 2
Hasil BFS traversal: 2 3 1
-----
Process exited after 175.1 seconds with return value 0
Press any key to continue . . .
```

3. Penjelasan

Algoritma Breadth First Search (BFS):

BFS adalah algoritma pencarian yang digunakan untuk traversing atau mencari semua simpul dari graf atau pohon secara berlapis.

Berikut adalah langkah-langkah utama dari algoritma BFS:

1. Mulai dari simpul awal (start node).
2. Tandai simpul awal sebagai sudah dikunjungi.
3. Masukkan simpul awal ke dalam antrian.
4. Lakukan loop hingga antrian kosong:
 - Keluarkan simpul pertama dari antrian.
 - Untuk setiap simpul terhubung dengan simpul yang dikeluarkan:
 - Jika simpul tersebut belum dikunjungi, tandai sebagai sudah dikunjungi dan masukkan ke dalam antrian.
5. Ulangi langkah 4 hingga semua simpul yang terhubung telah dikunjungi.

Prinsip Antrian (Queue) dalam BFS:

1. Pelacakan Simpul yang Telah Dikunjungi: Antrian digunakan untuk melacak simpul-simpul yang telah dikunjungi. Ketika sebuah simpul dikunjungi, ia dikeluarkan dari antrian dan ditandai sebagai sudah dikunjungi.
2. Penyimpanan Simpul yang Akan Dikunjungi: Antrian juga digunakan untuk menyimpan simpul-simpul yang akan dikunjungi selanjutnya. Ketika sebuah simpul dikeluarkan dari antrian, semua simpul yang terhubung dengan simpul tersebut

ditambahkan ke antrian. Ini memastikan bahwa simpul-simpul yang terhubung dengan simpul saat ini akan dikunjungi setelah semua simpul pada tingkat saat ini telah selesai dikunjungi.

3. Pencarian Berlapis (Layer-by-Layer): Dengan menggunakan antrian, algoritma BFS memastikan bahwa simpul-simpul yang lebih dekat dengan simpul awal akan dikunjungi terlebih dahulu sebelum simpul-simpul yang lebih jauh. Ini memastikan pencarian secara berlapis dari simpul-simpul graf, yang berguna misalnya dalam menemukan jalur terpendek dari simpul awal ke simpul lain dalam graf.

Dengan memanfaatkan prinsip antrian (queue), algoritma BFS dapat mencari semua simpul dari graf atau pohon secara sistematis dan efisien, dengan mengunjungi simpul-simpul secara berlapis dari simpul awal.