Tugas Jobsheet 7

Praktikum Struktur Data

“Queue”

Dosen Pengampu :

Randi Proska Sandra, S.Pd, M.Sc

Disusun oleh

Nama : Carel Habsian Osagi

Nim : 23343061

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

Algoritma Breadth-First Search (BFS) adalah metode pencarian atau penelusuran pada graf yang dimulai dari node akar (atau node awal yang ditentukan) dan menjelajahi semua tetangganya pada level yang sama sebelum berpindah ke level berikutnya. BFS menggunakan struktur data antrian (queue) untuk menjaga agar penelusuran dilakukan secara teratur dari node yang lebih dalam.

Prinsip Kerja Algoritma BFS

1. **Inisialisasi**:
    - Mulai dari node awal (`start`), tandai node tersebut sebagai dikunjungi dan masukkan ke antrian (queue).
2. **Proses Traversal**:
    - Selama antrian tidak kosong, lakukan langkah berikut:
        - Keluarkan elemen depan dari antrian (`deQueue`).
        - Cetak elemen tersebut (menunjukkan bahwa elemen tersebut sedang dikunjungi).
        - Periksa semua node yang terhubung dengan node saat ini.
        - Jika node yang terhubung belum dikunjungi, tandai sebagai dikunjungi dan masukkan ke antrian (`enQueue`).
3. **Lanjutkan Hingga Antrian Kosong**:
    - Algoritma berlanjut sampai semua node yang bisa dijangkau dari node awal telah dikunjungi, yaitu ketika antrian kosong.

Implementasi Program BFS dalam C

Berikut adalah penjelasan dari program C di atas yang mengimplementasikan BFS menggunakan queue:

1. **Struktur Node dan Queue**:
    - `struct Node`: Merepresentasikan setiap elemen dalam antrian, dengan dua anggota: `data` (menyimpan nilai) dan `next` (pointer ke elemen berikutnya).
    - `struct Queue`: Merepresentasikan antrian dengan dua pointer: `front` (menunjuk ke elemen pertama) dan `rear` (menunjuk ke elemen terakhir).
2. **Fungsi `newNode`**:
    - Membuat node baru dengan nilai `data` yang diberikan dan menginisialisasi pointer `next` dengan `NULL`.
3. **Fungsi `createQueue`**:
    - Membuat dan menginisialisasi antrian kosong dengan `front` dan `rear` yang menunjuk ke `NULL`.
4. **Fungsi `enQueue`**:
    - Menambahkan elemen baru ke belakang antrian.
    - Jika antrian kosong (`rear` adalah `NULL`), elemen baru menjadi elemen pertama (`front` dan `rear` menunjuk ke elemen baru).
    - Jika antrian tidak kosong, elemen baru ditambahkan di belakang dan `rear` diperbarui.
5. **Fungsi `deQueue`**:

    - Menghapus elemen dari depan antrian dan mengembalikan nilai elemen tersebut.
    - Jika antrian kosong (`front` adalah `NULL`), fungsi mengembalikan `-1`.
    - Jika antrian tidak kosong, elemen pertama dihapus dan `front` diperbarui.
    - Jika setelah penghapusan antrian menjadi kosong (`front` menjadi `NULL`), `rear` juga diperbarui menjadi `NULL`.
6. **Fungsi `BFS`**:
    - Melakukan traversal BFS pada graf.
    - Argumen `graph` adalah matriks ketetanggaan yang merepresentasikan graf.
    - Argumen `start` adalah node awal untuk memulai BFS.
    - Argumen `n` adalah jumlah node dalam graf.
    - Menggunakan array `visited` untuk melacak node yang sudah dikunjungi.
    - Membuat antrian `q` untuk melacak node yang harus dikunjungi selanjutnya.
    - Node awal ditandai sebagai dikunjungi dan dimasukkan ke antrian.
    - Selama antrian tidak kosong, fungsi mengeluarkan elemen depan dari antrian dan mencetaknya.
    - Kemudian, fungsi mengecek semua node yang terhubung dengan node saat ini. Jika node belum dikunjungi, node tersebut ditandai sebagai dikunjungi dan dimasukkan ke antrian.
7. **Fungsi `main`**:
    - Mendefinisikan graf sebagai matriks ketetanggaan.
    - Memanggil `BFS` dengan node awal `0` dan jumlah node `4`.

Penjelasan Matriks Ketetanggaan

Matriks ketetanggaan `graph[4][4]` digunakan untuk merepresentasikan graf:

```
int graph[4][4] = {
    {0, 1, 1, 0},
    {1, 0, 1, 1},
    {1, 1, 0, 1},
    {0, 1, 1, 0}
};
```

- Matriks ini menunjukkan koneksi antara node dalam graf.
- `graph[i][j]` bernilai `1` jika terdapat edge antara node `i` dan node `j`.
- `graph[i][j]` bernilai `0` jika tidak terdapat edge antara node `i` dan node `j`.

Contoh Output

Jika kita menjalankan program ini dengan graf yang diberikan, outputnya akan menunjukkan urutan node yang dikunjungi oleh BFS dimulai dari node `0`:

```
Breadth-First Search dimulai dari node 0:
0 1 2 3
```

Ini berarti BFS pertama kali mengunjungi node `0`, lalu node `1` dan `2` yang terhubung langsung dengan `0`, dan akhirnya node `3` yang terhubung dengan `1` dan `2`.

Prinsip Queue dalam BFS

Queue berperan penting dalam algoritma BFS untuk menjaga agar penelusuran dilakukan secara level-by-level. Queue memastikan bahwa node yang dijelajahi pertama kali adalah node yang terletak pada level yang sama sebelum melanjutkan ke level berikutnya. Pada dasarnya, queue mematuhi prinsip FIFO (First-In, First-Out) yang sesuai dengan kebutuhan BFS untuk melakukan penelusuran secara mendalam terlebih dahulu pada setiap level graf.

Berikut adalah Sorce Code Program :

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

// Struktur untuk merepresentasikan node dalam queue
struct Node {
    int data;
    struct Node* next;
};

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

// Fungsi untuk membuat node baru dalam queue
struct Node* newNode(int k) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = k;
    temp->next = NULL;
    return temp;
}

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

// Fungsi untuk menambah elemen ke queue
void enQueue(struct Queue* q, int k) {
    struct Node* temp = newNode(k);
    if (q->rear == NULL) {
        q->front = q->rear = temp;
        return;
```

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

// Fungsi untuk menghapus elemen dari queue
int deQueue(struct Queue* q) {
    if (q->front == NULL)
        return -1;
    struct Node* temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL)
        q->rear = NULL;
    int data = temp->data;
    free(temp);
    return data;
}

// Fungsi untuk melakukan Breadth-First Search
void BFS(int graph[][4], int start, int n) {
    int visited[4] = {0}; // Array untuk melacak node yang sudah dikunjungi
    struct Queue* q = createQueue(); // Membuat queue

    visited[start] = 1; // Menandai node awal sebagai dikunjungi
    enQueue(q, start); // Menambah node awal ke queue

    printf("Breadth-First Search dimulai dari node %d:\n", start);

    while (q->front != NULL) {
        int current = deQueue(q); // Mengambil elemen pertama dari queue
        printf("%d ", current);

        // Mengecek semua node yang terhubung dengan node saat ini
        for (int i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                visited[i] = 1; // Menandai node sebagai dikunjungi
                enQueue(q, i); // Menambah node ke queue
            }
        }
    }
    printf("\n");
}

int main() {
    int graph[4][4] = {
```

```
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 1},
        {0, 1, 1, 0}
    };

    BFS(graph, 0, 4); // Memulai BFS dari node 0

    return 0;
}
```