PENJELASAN MATERI-7

Queue/Antrian

Dosen pengampu:
Randi Proska Sandra, S.Pd, M.Sc.

Disusun oleh:
Muhammad Ghazian Tsaqif Zhafiri Andoz (23343057)

Algoritma Breadth First Search (BFS)

Algoritma Breadth-First Search (BFS) adalah algoritma yang digunakan untuk menelusuri semua simpul dalam sebuah graf secara melebar. Pencarian dimulai dari simpul awal dan kemudian berlanjut ke semua simpul yang bertetangga dengan simpul awal. Setelah itu, algoritma akan berlanjut ke simpul-simpul yang bertetangga dengan simpul-simpul yang telah dikunjungi sebelumnya, dan seterusnya, sampai semua simpul dalam graf telah dikunjungi. Berikut adalah penjelasan dari source code C mengenai Algoritma Breadth First Search (BFS):

STRUKTUR DATA

NO	Nama Fungsi	Penjelasan
1	**struct** queue* createQueue()	Fungsi ini digunakan untuk membuat dan menginisialisasi sebuah antrian (queue) baru. Ini mengalokasikan memori untuk antrian, menetapkan indeks depan dan belakang ke nilai awal.
2	**void** enqueue(struct queue* q, int)	Fungsi ini digunakan untuk menambahkan elemen ke dalam antrian. Jika antrian penuh, fungsi ini akan mencetak pesan kesalahan. Jika tidak, itu akan menambahkan elemen baru ke belakang antrian.
3	**int** dequeue(struct queue* q)	Fungsi ini digunakan untuk menghapus elemen dari depan antrian dan mengembalikan nilainya. Jika antrian kosong, fungsi ini akan mencetak pesan kesalahan dan mengembalikan nilai -1.
4	**int** isEmpty(struct queue* q)	Fungsi ini mengembalikan 1 jika antrian kosong dan 0 jika tidak.
5	**void** printQueue(struct queue* q)	Fungsi ini digunakan untuk mencetak isi dari antrian.
6	**struct** node* createNode(int v)	Fungsi ini digunakan untuk membuat node baru dengan nilai vertex tertentu.
7	**struct** Graph* createGraph(int vertices)	Fungsi ini digunakan untuk membuat dan menginisialisasi graf baru. Ini mengalokasikan memori untuk graf, daftar ke adjacency list, dan array visited.
8	**void** addEdge(struct Graph* graph, int src, int dest)	Fungsi ini digunakan untuk menambahkan sisi (edge) antara dua vertex tertentu dalam graf.
9	**void** bfs(struct Graph* graph, int startVertex)	Fungsi ini merupakan implementasi algoritma Breadth-First Search (BFS) untuk graf yang diberikan, dimulai dari vertex yang ditentukan.

PENJELASAN CARA KERJA

Algoritma ini bekerja dengan cara menjelajahi secara melebar dari simpul awal, sehingga umumnya digunakan untuk mencari jalur terpendek dari satu simpul ke simpul lainnya. Berikut adalah langkah-langkah kerja algoritma BFS:

1) Inisialisasi Graf:

- Pertama-tama, graf yang akan di-traversal harus dibuat dengan menggunakan fungsi createGraph().
- Setiap simpul dalam graf memiliki daftar node yang diwakili oleh struktur data linked list. Inisialisasi dilakukan untuk mengatur array dari linked list ini dan menyatakan bahwa belum ada simpul yang dikunjungi.

2) Penambahan Sisi (Edge):

- Setelah graf diinisialisasi, sisi-sisi graf ditambahkan menggunakan fungsi addEdge().
- Pada graf tak berarah seperti dalam contoh ini, setiap sisi harus ditambahkan dua kali, satu untuk setiap simpul yang terhubung.

3) Algoritma BFS:

- Pada fungsi bfs(), antrian (queue) dibuat menggunakan fungsi createQueue(). Antrian ini akan digunakan untuk melacak simpul-simpul yang telah dikunjungi dan simpul-simpul yang masih harus dikunjungi.
- Simpul awal yang akan dimulai traversalnya ditambahkan ke antrian dan ditandai sebagai "dikunjungi".
- Selama antrian tidak kosong, iterasi akan terus dilakukan:
- Simpul yang saat ini berada di depan antrian diambil sebagai simpul saat ini yang sedang diproses.
- Semua simpul tetangga yang belum dikunjungi dari simpul saat ini ditandai sebagai "dikunjungi" dan dimasukkan ke antrian.
- Proses ini terus berlanjut hingga semua simpul yang terhubung dengan simpul awal telah dikunjungi.

4) Membuat Node:

- Fungsi createNode() digunakan untuk membuat node baru untuk linked list.
- Setiap node ini mewakili simpul dalam graf.

5) Membuat Antrian (Queue):

- Fungsi createQueue() digunakan untuk membuat antrian yang akan digunakan dalam algoritma BFS.
- Antrian dibuat dengan struktur data yang sederhana, dengan variabel penunjuk untuk penambahan dan penghapusan elemen.

6) **Operasi Antrian:**

- Operasi-operasi seperti enqueue() (menambahkan elemen ke antrian) dan dequeue() (menghapus elemen dari antrian) diimplementasikan sesuai dengan aturan FIFO (First In, First Out).
- Fungsi isEmpty() digunakan untuk memeriksa apakah antrian kosong.
- Fungsi printQueue() digunakan untuk mencetak isi dari antrian.

Setelah semua langkah ini dilakukan, algoritma BFS akan melakukan traversal pada graf dari simpul awal yang ditentukan, menampilkan semua simpul yang dapat dicapai dari simpul awal tersebut secara berurutan menurut jarak.

SOURCE CODE

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

#define SIZE 40

struct queue {
    int items[SIZE];
    int front;
    int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node {
    int vertex;
    struct node* next;
};

struct node* createNode(int);

struct Graph {
    int numVertices;
    struct node** adjLists;
    int* visited;
};

// Panggil Fungsi
void bfs(struct Graph* graph, int startVertex);
struct node* createNode(int v);
struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
struct queue* createQueue();
int isEmpty(struct queue* q);
void enqueue(struct queue* q, int value);
```

```c
int dequeue(struct queue* q);
void printQueue(struct queue* q);

int main() {
    struct Graph* graph = createGraph(6);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);

    bfs(graph, 0);

    return 0;
}

// ------------------------------------------------
// Algoritma BFS
void bfs(struct Graph* graph, int startVertex) {
    struct queue* q = createQueue();

    graph->visited[startVertex] = 1;
    enqueue(q, startVertex);

    while (!isEmpty(q)) {
        printQueue(q);
        int currentVertex = dequeue(q);
        printf("\n>> Dikunjungi: %d\n", currentVertex);

        struct node* temp = graph->adjLists[currentVertex];

        while (temp) {
            int adjVertex = temp->vertex;

            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }
}

// Membuat node
struct node* createNode(int v) {
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// Membuat graf
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

    graph->adjLists = malloc(vertices * sizeof(struct node*));
```

```
    graph->visited = malloc(vertices * sizeof(int));

    int i;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

// Menambahkan sisi
void addEdge(struct Graph* graph, int src, int dest) {
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Membuat antrian
struct queue* createQueue() {
    struct queue* q = malloc(sizeof(struct queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

// Memeriksa apakah antrian kosong
int isEmpty(struct queue* q) {
    if (q->rear == -1)
        return 1;
    else
        return 0;
}

// Menambahkan elemen ke dalam antrian
void enqueue(struct queue* q, int value) {
    if (q->rear == SIZE - 1)
        printf("\n----[ Antrian penuh ]----\n");
    else {
        if (q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}

// Menghapus elemen dari antrian
int dequeue(struct queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("\n----[ Antrian Kosong ]----\n");
        item = -1;
    } else {
        item = q->items[q->front];
        q->front++;
```

```
        if (q->front > q->rear) {
            printf("\n>> Merestart antrian...");
            q->front = q->rear = -1;
        }
    }
    return item;
}

// Mencetak antrian
void printQueue(struct queue* q) {
    int i = q->front;

    if (isEmpty(q)) {
        printf("\n----[ Antrian Kosong ]----\n");
    } else {
        printf("\n----[ Isi Antrian ]----\n");
        for (i = q->front; i <= q->rear; i++) {
            printf("~~> %d ", q->items[i]);
        }
    }
}
```