PROGRAM ALGORITMA BFS

Disusun Oleh :
M. Fakhrul Zuhdi

BP/NIM: 2023/23343074

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

FAKULTAS TEKNIK
DEPARTEMEN TEKNIK ELEKTRONIKA

PROGRAM STUDI S1 TEKNIK INFORMATIKA

UNIVERSITAS NEGERI PADANG

2024

Penjelasan Algoritma BFS

Breadth First Search (BFS) adalah algoritma untuk melakukan traversal atau pencarian pada graf atau pohon. Algoritma ini bekerja dengan cara mengeksplorasi semua tetangga dari sebuah node sebelum bergerak ke node berikutnya.

Cara kerja BFS dengan Queue:

1. Inisialisasi: Mulai dengan memasukkan node awal (startVertex) ke dalam queue dan menandai node tersebut sebagai dikunjungi.

2. Iterasi:

- Selama queue tidak kosong, ambil node dari depan queue (dequeue) dan kunjungi node tersebut.

- Masukkan semua tetangga yang belum dikunjungi dari node tersebut ke dalam queue (enqueue) dan tandai sebagai dikunjungi.

3. Selesai: Algoritma berakhir ketika semua node telah dikunjungi dan queue kosong.

Queue digunakan untuk menyimpan node yang akan dikunjungi selanjutnya, memastikan bahwa setiap node dieksplorasi dalam urutan yang benar (semua tetangga dari satu node dieksplorasi sebelum beralih ke node lainnya).

1. `swap(int *a, int *b)`

```
void swap(int *a, int *b) {
    int temp = *b;
    *b = *a;
    *a = temp;
}
```

- Fungsi: Menukar nilai dari dua variabel yang ditunjuk oleh pointer `a` dan `b`.

- Penggunaan: Tidak digunakan dalam implementasi BFS, tapi biasanya digunakan dalam algoritma yang melibatkan pengurutan atau penyesuaian heap.

2. `initQueue(Queue *q)`

```
void initQueue(Queue *q) {
    q->front = -1;
    q->rear = -1;
}
```

- Fungsi: Menginisialisasi queue dengan mengatur nilai `front` dan `rear` menjadi -1, menunjukkan bahwa queue kosong.

- Penggunaan: Digunakan saat pertama kali membuat atau menginisialisasi queue.

3. `isEmpty(Queue *q)`

```
int isEmpty(Queue *q) {
    return (q->rear == -1);
}
```

- Fungsi: Mengecek apakah queue kosong dengan memeriksa apakah `rear` bernilai -1.

- Penggunaan: Digunakan sebelum men-dequeue atau meng-enqueue elemen untuk memastikan queue tidak kosong.

4. `isFull(Queue *q)`

```
int isFull(Queue *q) {
    return (q->rear == MAX - 1);
}
```

- Fungsi: Mengecek apakah queue penuh dengan memeriksa apakah `rear` sudah mencapai kapasitas maksimum (MAX - 1).

- Penggunaan: Digunakan sebelum meng-enqueue elemen untuk memastikan queue tidak penuh.

5. `enqueue(Queue *q, int value)`

```
void enqueue(Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue penuh\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
}
```

- Fungsi: Menambahkan elemen `value` ke dalam queue.
- Penggunaan: Digunakan untuk menambah node ke dalam antrian saat melakukan BFS.

6. `dequeue(Queue *q)`

```
int dequeue(Queue *q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue kosong\n");
        return -1;
    }
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}
```

- Fungsi: Menghapus dan mengembalikan elemen dari depan queue.

- Penggunaan: Digunakan untuk mengambil node berikutnya yang akan dikunjungi dalam BFS.

7. `createGraph(int vertices)`

```
Graph *createGraph(int vertices) {
    Graph *graph = (Graph *)malloc(sizeof(Graph));
    graph->numVertices = vertices;
    graph->adjMatrix = (int )malloc(vertices * sizeof(int *));
    graph->visited = (int *)malloc(vertices * sizeof(int));

    for (int i = 0; i < vertices; i++) {
        graph->adjMatrix[i] = (int *)malloc(vertices * sizeof(int));
        graph->visited[i] = 0;
        for (int j = 0; j < vertices; j++) {
            graph->adjMatrix[i][j] = 0;
        }
    }
    return graph;
}
```

- Fungsi: Membuat graph dengan sejumlah vertex tertentu. Mengalokasikan memori untuk matriks ketetanggaan (adjacency matrix) dan array visited.

- Penggunaan: Digunakan untuk menginisialisasi graph dengan vertex yang diberikan.

8. `addEdge(Graph *graph, int src, int dest)`

```
void addEdge(Graph *graph, int src, int dest) {
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1;
}
```

- Fungsi: Menambahkan edge antara dua vertex `src` dan `dest` dalam graph.

- Penggunaan: Digunakan untuk membangun graph dengan menambahkan hubungan antar vertex.

9. `bfs(Graph *graph, int startVertex)`

```
void bfs(Graph *graph, int startVertex) {
    Queue q;
    initQueue(&q);

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

    while (!isEmpty(&q)) {
        int currentVertex = dequeue(&q);
        printf("Visited %d\n", currentVertex);

        for (int i = 0; i < graph->numVertices; i++) {
            if (graph->adjMatrix[currentVertex][i] == 1 && !graph-
>visited[i]) {
                graph->visited[i] = 1;
                enqueue(&q, i);
            }
        }
    }
}
```

- Fungsi: Melakukan traversal BFS dari node `startVertex`. Menggunakan queue untuk melacak node mana yang akan dikunjungi selanjutnya.

- Penggunaan: Digunakan untuk menampilkan semua node yang dikunjungi dalam urutan BFS.

10. `main()`

```
int main() {
    Graph *graph = createGraph(6);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);
    addEdge(graph, 3, 5);
    addEdge(graph, 4, 5);

    bfs(graph, 0);

    return 0;
}
```

- Fungsi: Fungsi utama yang menginisialisasi graph dengan 6 vertex dan menambahkan beberapa edge. Kemudian, memulai traversal BFS dari vertex 0.

- Penggunaan: Menjalankan program dan menunjukkan bagaimana BFS berjalan pada graph yang telah dibuat.