Penjelasan Jurnal 10

Pekan ke-10

Dibuat oleh :

Muh Daffah Putra Muharsyah 607062330131

Implementasi Struktur Data

D3 Rekayasa Perangkat Lunak Aplikasi
Fakultas Ilmu Terapan
Tahun 2024

```
4       private Map<Character, List<Character>> adjList;
5
6       public Graph() {
7           adjList = new HashMap<>();
8       }
9
10      public void addEdge(char source, char destination) {
11          adjList.computeIfAbsent(source, k -> new ArrayList<>()).add(destination);
12      }
13
14      public List<Character> getAdjacentVertices(char vertex) {
15          return adjList.getOrDefault(vertex, new ArrayList<>());
16      }
```

Langkah awal, dengan membuat variabel untuk menyimpan node bertipe map list, lalu membuat konstraktor dengan nama Graph yang berisikan objek baru hashmap dari variabel yang telah kita buat sebelumnya. Kemudian membuat method untuk menambahkan edge. Dan terakhir kita membuat method lagi untuk mendapatkan daftar node tetangga dari suatu node tertentu.

```java
// Breadth First Search
public void BFS(char start) {
    Queue<Character> queue = new LinkedList<>();
    Set<Character> visited = new HashSet<>();
    queue.add(start);
    visited.add(start);

    while (!queue.isEmpty()) {
        char current = queue.poll();
        System.out.print(current + " ");

        for (char neighbor : getAdjacentVertices(current)) {
            if (!visited.contains(neighbor)) {
                queue.add(neighbor);
                visited.add(neighbor);
            }
        }
    }
}
```

Membuat method Breadth First Search (BFS)

Pertama kita membuat antrian untuk daftar node yang akan dikunjungi dengan nama variabel Queue, baris selanjutnya membuat tempat node yang telah dikunjungi dengan nama Visited. Kemudian node awal masuk pada antrian yang akan dikunjungi setelahnya menyusul masuk ke node yang telah dikunjungi.

Kemudian terjadi perulangan kondisi selama daftar yang akan dikunjungi tidak kosong akan melakukan

1. Mengambil dan menyimpan node paling depan antrian dalam variabel current
2. Mencetak variabel current
3. Melakukan perulangan lagi untuk mengecek semua tetangga dari node current. Perulangannya akan memiliki kondisi jika tetangga belum dikunjungi maka tetangga akan masuk ke antrian yang akan dikunjungi dan masuk ke tempat node yang telah dikunjungi

```java
// Depth First Search
public void DFS(char start, Set<Character> visited) {
    visited.add(start);
    System.out.print(start + " ");

    for (char neighbor : getAdjacentVertices(start)) {
        if (!visited.contains(neighbor)) {
            DFS(neighbor, visited);
        }
    }
}
```

Membuat method Depth First Search (DFS)

Method ini akan menerima dua parameter, yaitu node awal penelusuran dan node yang telah dikunjungi sebelumnya, selanjutnya kita akan menambahkan node start sebagai telah dikunjungi dengan memasukkannya dalam variabel visited. Setelah itu kita akan mencetak variabel start. Kemudian untuk mengunjungi variabel selanjutnya kita membuat perulangan untuk mengecek semua tetangga node saat ini. Dalam mengecek node tetangga, kita membuat kondisi jika tetangga belum dikunjungi maka akan melakukan rekursif dengan memanggil method DFS Kembali untuk mengunjungi tetangga yang belum dikunjungi tersebut.

```java
public static void main(String[] args) {
    Graph graph = new Graph();
    graph.addEdge(source:'A', destination:'D');
    graph.addEdge(source:'D', destination:'G');
    graph.addEdge(source:'G', destination:'H');
    graph.addEdge(source:'H', destination:'I');
    graph.addEdge(source:'I', destination:'F');
    graph.addEdge(source:'F', destination:'C');
    graph.addEdge(source:'C', destination:'B');
    graph.addEdge(source:'B', destination:'E');
    graph.addEdge(source:'E', destination:'H');
    graph.addEdge(source:'A', destination:'E');
    graph.addEdge(source:'A', destination:'B');
    graph.addEdge(source:'E', destination:'F');
    graph.addEdge(source:'F', destination:'H');

    System.out.println(x:"BFS traversal:");
    graph.BFS(start:'A');
    System.out.println(x:"\nDFS traversal:");
    Set<Character> visited = new HashSet<>();
    graph.DFS(start:'A', visited);
}
```

Main program terlebih dahulu membuat objek graph, kemudian menyambungkan tiap node satu sama lain dengan fungsi addEdge yang telah dibuat sebelumnya. Kemudian akan selanjutnya akan memanggil method BFS yang telah kita buat dengan start method pada node A. Method BFS akan mengunjungi semua tetangga nodenya terlebih dahulu sebelum mengunjungi node lain. Setelah itu kita membuat himpunan untuk melacak node yang telah dikunjungi bernama visited untuk dapat menggunakan method rekursif DFS yang telah kita buat. Method DFS akan mengunjungi satu tetangga secara mendalam sampai semua tetangga telah dikunjungi.

Adapun hasil Penelusuran atau Travers dari BFS dan DFS adalah sebagai berikut:
BFS : A D E B G H F I C

DFS: A D G H I F C B E