Nama : Anggun Fia Febianingrum

NIM : 5311421010

Rombel : 1, Senin 09.00

PRAKTIKUM 4 ARTIFICIAL INTELLIGENCE

1. Tentukan bagaimana algoritma BFS di atas dapat menentukan node ke 8, 6, dan 7. 2.
2. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.5 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 5.
3. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.6 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 9.
4. Ubahlah kode program di atas sehingga bentuk tree seperti Gambar 4.7 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node C.

Penyelesaian :

1. Algoritma BFS (Breadth-First Search) adalah salah satu algoritma pencarian dalam ilmu komputer yang digunakan untuk menjelajahi atau mencari informasi dalam graf atau struktur data berdasarkan tingkat jarak atau kedalaman dari node awal. Algoritma ini bekerja dengan cara menjelajahi node-node yang terhubung secara langsung dengan node awal sebelum mengeksplorasi node-node yang lebih dalam atau lebih jauh. Berikut merupakan langkah Pada Algoritma BFS
   a) menginisiasi node awal, yaitu node 3, kemudian dimasukkan ke dalam antrian.
   b) Selanjutnya, node 3 akan mengeksplorasi node-node yang terhubung langsung dengan itu, yaitu node 2 dan node 4, karena keduanya memiliki jarak 1 dari node 3. Node 4 akan melanjutkan dengan mengeksplorasi node 1, node 6, dan node 5 yang terhubung dengannya.
   c) Selanjutnya, node 5 akan mengeksplorasi node 8 dan node 7.
   d) Proses ini akan terus berlanjut sampai semua node yang terhubung dengan node awal telah diperiksa.

   Oleh karena itu, pada akhir proses BFS, node-node 8, 6, dan 7 akan ditemukan dan diproses sesuai dengan aturan algoritma BFS.

2. Kemudian untuk membuat tree seperti pada gambar 4.5 method static void main diubah seperti berikut:

```
public static void main(String[] args)
{
    AdjacencyList1 graph = new AdjacencyList1();
    Node n1 = new Node(data:0);
    Node n2 = new Node(data:1);
    Node n3 = new Node(data:2);
    Node n4 = new Node(data:3);
    Node n5 = new Node(data:4);
    Node n6 = new Node(data:5);
    Node n7 = new Node(data:6);
```

```
    graph.addEdge(n1, n2);
    graph.addEdge(n1, n3);

    graph.addEdge(n2, n1);
    graph.addEdge(n2, n4);
    graph.addEdge(n2, n5);

    graph.addEdge(n3, n1);
    graph.addEdge(n3, n6);
    graph.addEdge(n3, n7);

    graph.addEdge(n4, n2);

    graph.addEdge(n5, n2);

    graph.addEdge(n6, n3);

    graph.addEdge(n7, n3);

    graph.bfs(n1);
}
```

Selanjutnya Run program maka akan didapatkan hasil sebagai berikut:

```
(0,d=0) (1,d=1) (2,d=1) (3,d=2) (4,d=2) (5,d=2) (6,d=2)
PS C:\Users\Asus>
```

Hasil tersebut mengkonfirmasi kesesuaian hasil tree yang dihasilkan oleh program dengan gambar 4.5.

a) Untuk mencari node 5, algoritma BFS pertama-tama dimulai dengan langkah memasukkan node 1 ke dalam antrian.

b) Selanjutnya, node 1 akan mengeksplorasi node 2 dan node 3, yang secara langsung terhubung dengan node 1 atau memiliki kedalaman tingkat 1.

c) Kemudian, node 3 akan melanjutkan dengan mengeksplorasi node yang memiliki kedalaman 2, dimulai dengan mengecek node 4, node 5, node 6, dan node 7.

d) Setelah node 5 ditemukan, proses akan terus berlanjut hingga semua node yang terhubung dengan node awal telah diperiksa.

Oleh karena itu, pada akhir proses BFS, node 5 akan ditemukan dan diolah sesuai dengan ketentuan algoritma BFS.

3. Pembuatan tree seperti pada gambar 4.6 menggunakan method static void main diubah seperti berikut:

```
public static void main(String[] args)
{
    AdjacencyList2 graph = new AdjacencyList2();
    Node n1 = new Node(data:1);
    Node n2 = new Node(data:2);
    Node n3 = new Node(data:3);
    Node n4 = new Node(data:4);
    Node n5 = new Node(data:5);
    Node n6 = new Node(data:6);
    Node n7 = new Node(data:7);
    Node n8 = new Node(data:8);
    Node n9 = new Node(data:9);
    Node n10 = new Node(data:10);
    Node n11 = new Node(data:11);
    Node n12 = new Node(data:12);
```

```
    graph.addEdge(n1, n2);
    graph.addEdge(n1, n3);
    graph.addEdge(n1, n4);

    graph.addEdge(n2, n1);
    graph.addEdge(n2, n5);
    graph.addEdge(n2, n6);

    graph.addEdge(n3, n1);

    graph.addEdge(n4, n1);
    graph.addEdge(n4, n7);
    graph.addEdge(n4, n8);

    graph.addEdge(n5, n2);
    graph.addEdge(n5, n9);
    graph.addEdge(n5, n10);

    graph.addEdge(n6, n2);

    graph.addEdge(n7, n4);
    graph.addEdge(n7, n11);
    graph.addEdge(n7, n12);

    graph.addEdge(n8, n4);

    graph.addEdge(n9, n5);

    graph.addEdge(n10, n5);

    graph.addEdge(n11, n7);

    graph.addEdge(n12, n7);


    graph.bfs(n1);

}
```

Kemudian Run program dan didapatkan hasil sebagai berikut:

```
(1,d=0) (2,d=1) (3,d=1) (4,d=1) (5,d=2) (6,d=2) (7,d=2) (8,d=2) (9,d=3) (10,d=3) (11,d=3) (12,d=3)
PS C:\Users\Asus>
```

Hasil tersebut sudah menunjukkan bahwa hasil tree yg diperoleh dari program sudah sesuai dengan gambar 4.6.

a) Untuk menemukan node 9 algoritma BFS pertama-tama dimulai dengan memasukkan node 1 ke dalam antrian.
b) Kemudian dilanjutkan node 1 akan memeriksa node 2, node 3, dan node 4 yang terhubung langsung dengan node 1 atau node yang memiliki tingkat kedalaman 1.
c) Dilanjutkan node 4 akan memeriksa node yang memiliki tingkat kedalaman 2 dimulai dari memeriksa node 5, node 6, node 7, dan node 8.
d) Kemudian node 8 akan memeriksa node 9, setelah menemukan node 9 proses akan terus berlanjut untuk memeriksa node 10, node 11, dan node 12 sampai semua node yang terhubung dengan node awal telah diperiksa.

Dengan demikian, pada akhir proses BFS, node 9 akan ditemukan dan diproses sesuai dengan aturan algoritma BFS.

4. Pembuatan tree seperti pada gambar 4.7, pada pembuatan tree kali ini akan dilakukan beberapa perubahan pada program.
    1) Pada kelas Node, untuk dapat menerima huruf maka Ganti tipe data variabel ’data’ dari yang awalnya ’int’ menjadi ’String’.

```
public static class Node
{
    String data;
    int distance;
    Node predecessor;
    NodeColour colour;

    public Node(String data)
    {
        this.data = data;
    }

    public String toString()
    {
     return "(" + data + ",d=" + distance + ")";
    }
}
```

    2) Kemudian pada Metode main, Masukkan nilai node dengan tipe data string sebagai berikut.

```
Run | Debug
public static void main(String[] args)
{
    AdjacencyList3 graph = new AdjacencyList3();
    Node n1 = new Node(data:"A");
    Node n2 = new Node(data:"B");
    Node n3 = new Node(data:"C");
    Node n4 = new Node(data:"D");
    Node n5 = new Node(data:"E");
    Node n6 = new Node(data:"F");
    Node n7 = new Node(data:"G");
    Node n8 = new Node(data:"H");
    Node n9 = new Node(data:"I");
```

3) Selanjutnya pada Metode addEdge dan bfs, Masukkan logika pemrosesan data untuk tipe data String sebagai berikut.

```
graph.addEdge(n1, n2);

graph.addEdge(n2, n1);
graph.addEdge(n2, n4);
graph.addEdge(n2, n6);

graph.addEdge(n3, n4);

graph.addEdge(n4, n2);
graph.addEdge(n4, n3);
graph.addEdge(n4, n5);

graph.addEdge(n5, n4);

graph.addEdge(n6, n2);
graph.addEdge(n6, n7);

graph.addEdge(n7, n6);
graph.addEdge(n7, n9);

graph.addEdge(n8, n9);

graph.addEdge(n9, n8);

graph.bfs(n6);
```

Kemudian Run program maka akan didapatkan hasil sebagai berikut:

```
(F,d=0) (B,d=1) (G,d=1) (A,d=2) (D,d=2) (I,d=2) (C,d=3) (E,d=3) (H,d=3)
PS C:\Users\Asus>
```

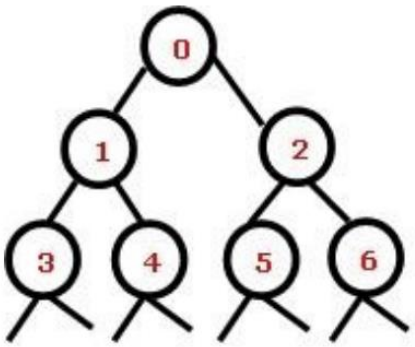
Hasil tersebut sudah menunjukkan bahwa hasil tree yg diperoleh dari program sudah sesuai dengan gambar 4.7.

a) Untuk menemukan node 3 (C) algoritma BFS pertama-tama dimulai dengan memasukkan node 6 (F) ke dalam antrian.
b) Kemudian dilanjutkan node 6 akan memeriksa node 2 (B), dan node 7 (G) yang terhubung langsung dengan node 6 atau node yang memiliki tingkat kedalaman 1.
c) selanjutnya node 7 akan memeriksa node yang memiliki tingkat kedalaman 2 dimulai dari memeriksa node 1 (A), node 4 (D), dan node 9 (I).
d) Kemudian node 9 akan memeriksa node 3, setelah menemukan node 3 (C) proses akan terus berlanjut untuk memeriksa node 5 (E), dan node 8 (H) sampai semua node yang terhubung dengan node awal telah diperiksa.

Dengan demikian, pada akhir proses BFS, node 3 (C) akan ditemukan dan diproses sesuai dengan aturan algoritma BFS.

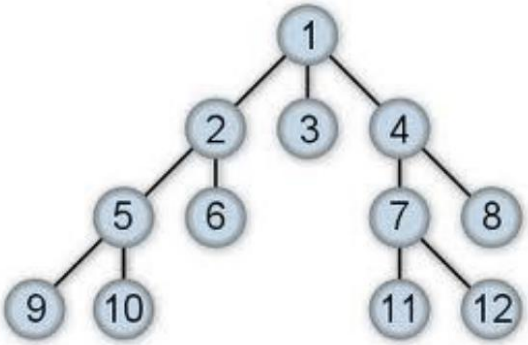
LAMPIRAN

1. Lampiran 1:



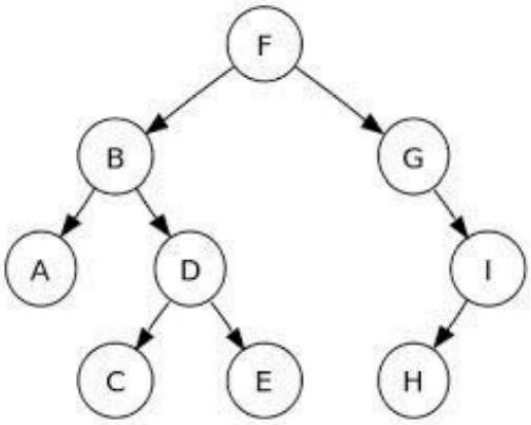
Gambar 4.5 Tree 1

2. Lampiran 2:



Gambar 4.6 Tree 2

3. Lampiran 3:



Gambar 4.7 Tree 3