Nama: M. Farhan Aryo Lazuardi

NIM: 5311421016

Hasil Analisis Modul Praktikum 4

1. Tentukan bagaimana algoritma BFS di atas dapat menentukan node ke 8, 6, dan 7.

Algoritma BFS (Breadth-First Search) dalam kode di atas digunakan untuk menjelajahi graf yang diwakili oleh adjacency list dan mencari jarak terpendek dari sebuah node awal (dalam kasus ini, node 3) ke node-node lain dalam graf.

Untuk menentukan bagaimana algoritma BFS di atas dapat menentukan node 8, 6, dan 7, mari kita analisis langkah-langkahnya:

- Algoritma BFS dimulai dari node awal, yaitu node 3. Node ini diberi warna GRAY (sedang diproses) dan jaraknya diatur ke 0.
- Node-node yang bertetangga dengan node 3 adalah node 2 dan node 4. Node 2 dan 4 ditambahkan ke antrian (queue) untuk diproses lebih lanjut. Mereka juga diberi warna GRAY dan jaraknya diatur ke 1.
- Selanjutnya, algoritma akan memproses node 2, yang menjadi node saat ini. Node-node yang bertetangga dengan node 2 adalah node 1 dan node 3. Namun, node 1 telah diwarnai GRAY sebelumnya, jadi itu diabaikan. Node 3 juga telah diwarnai GRAY sebelumnya, jadi itu juga diabaikan.
- Setelah node 2 selesai diproses, node 4 menjadi node saat ini. Node-node yang bertetangga dengan node 4 adalah node 3, 5, dan 6. Node 3 sudah diwarnai GRAY, dan 5 serta 6 berwarna PUTIH. Jadi, node-node 5 dan 6 ditambahkan ke antrian untuk diproses lebih lanjut, dan mereka diberi warna GRAY dengan jarak yang diatur ke 2.
- Proses berlanjut dengan node 5 dan 6. Node-node yang bertetangga dengan mereka akan diberi warna GRAY dan diberi jarak yang sesuai. Proses ini akan terus berlanjut hingga semua node yang dapat dijangkau dari node 3 telah diproses.
- Setelah semua node telah diproses, algoritma BFS selesai. Node-node 7 dan 8 akan menjadi GRAY (sedang diproses) selama proses, tetapi mereka tidak akan ditambahkan ke antrian karena mereka hanya memiliki tepat satu edge yang menuju node 6, yang sudah berada dalam antrian.

Jadi, dengan algoritma BFS ini, node 8, 6, dan 7 akan menjadi BLACK (selesai diproses) setelah seluruh proses BFS selesai, dan kita dapat menentukan jarak terpendek dari node 3 ke node 8, 6, dan 7 dalam graf tersebut. Jarak dari node 3 ke node 8 adalah 3, jarak dari node 3 ke node 6 adalah 2, dan jarak dari node 3 ke node 7 adalah 3.

Hasil Program :

```
(3,d=0) (4,d=1) (2,d=1) (5,d=2) (6,d=2) (1,d=2) (7,d=3) (8,d=3)

Distance from n1 to Node 8: 3
Distance from n1 to Node 6: 2
Distance from n1 to Node 7: 3
```

2. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.4 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 5.

Algoritma BFS (Breadth-First Search) bekerja dengan menjelajahi graf dari simpul awal (dalam hal ini, simpul 1) secara berlapis-lapis. Tujuan utama dari BFS adalah untuk menemukan jarak terpendek antara simpul awal (s) dan semua simpul lain dalam graf. Jarak diukur dalam jumlah langkah atau tepatnya dalam jumlah tepi yang harus dilalui dari simpul awal ke simpul lainnya.

Dalam hasil program yang Anda berikan, simpul 5 memiliki jarak (distance) 2 dari simpul awal 1. Berikut adalah langkah-langkah bagaimana BFS menemukan simpul 5:

- Pada awalnya, semua simpul dalam graf diberi warna putih (NodeColour.WHITE), yang menunjukkan bahwa mereka belum dieksplorasi.
- Kemudian, simpul awal (Node 1) diberi warna abu-abu (NodeColour.GRAY) dan diberi jarak 0, karena ini adalah simpul awal. Ini juga tidak memiliki predecesor.
- Simpul 1 dimasukkan ke dalam antrian (queue).
- Proses BFS dimulai. Pada setiap langkah, simpul yang ada dalam antrian dihapus dan dieksplorasi. Simpul yang dihapus dalam langkah pertama adalah simpul 1.
- Setelah simpul 1 dieksplorasi, semua simpul yang terhubung langsung dengan simpul 1 (yaitu simpul 2 dan simpul 3) diberi warna abu-abu dan jarak (distance) 1. Mereka juga memiliki simpul 1 sebagai predecesor.
- Simpul-simpul yang baru ini (2 dan 3) dimasukkan ke dalam antrian.

- Selanjutnya, simpul 2 dikeluarkan dari antrian dan dieksplorasi. Semua simpul yang terhubung langsung dengan simpul 2 (simpul 1, simpul 4, dan simpul 5) diberi warna abu-abu dan jarak 2.
- Simpul-simpul yang baru ini (4 dan 5) dimasukkan ke dalam antrian.
- Proses ini berlanjut hingga antrian kosong.
- Pada akhir algoritma, semua simpul akan dieksplorasi, dan masing-masing simpul akan memiliki warna hitam (NodeColour.BLACK), menunjukkan bahwa mereka sudah selesai dieksplorasi.

Jadi, simpul 5 ditemukan dalam 2 langkah (distance = 2) dari simpul awal 1. Ini adalah hasil dari algoritma BFS yang bekerja secara berlapis-lapis untuk menemukan jarak terpendek dari simpul awal ke semua simpul lain dalam graf.

Hasil Program:

```
(1,d=0) (2,d=1) (3,d=1) (4,d=2) (5,d=2) (6,d=2) (7,d=2) (8,d=3)
```

3. Ubahlah method static void main sehingga bentuk tree seperti Gambar 4.5 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 9.

Ini adalah hasil dari BFS yang dimulai dari node `n1` dan menandai setiap node dengan jaraknya dari node awal `n1`.

Untuk menemukan node 9 dalam graf tersebut menggunakan algoritma BFS, Anda dapat melihat pada jarak (`distance`) dari setiap node dari node awal. Dalam hasil BFS di atas, node 9 memiliki jarak (`distance`) 3 dari node awal `n1`. Jadi, Anda dapat mengatakan bahwa node 9 ditemukan dalam 3 langkah dari node `n1` dalam BFS. Anda juga dapat melihat bahwa node 9 dicapai melalui node 5 (dalam jarak 2) dan kemudian dari node 5 ke node 9 (dalam jarak 3).

Ini adalah cara algoritma BFS bekerja untuk menemukan node tertentu dalam graf dengan memberikan jarak dari node awal ke node yang dicari. Dalam kasus ini, node 9 ditemukan dalam 3 langkah dari node awal `n1`.

Hasil Program:

```
(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)
```

4. Ubahlah kode program di atas sehingga bentuk tree seperti Gambar 6 dapat dibentuk. Kemudian tentukan bagaimana algoritma BFS dapat menemukan node 3(C).

Jika Kita ingin menemukan bagaimana algoritma BFS menemukan node 3, maka Anda dapat melacak langkah-langkah BFS:

- BFS dimulai dari node awal, yaitu node 6 (dalam representasi graf Anda).
- Node 6 dicatat dengan jarak 0 (d=0), dan status warna node diubah menjadi GRAY (menunjukkan bahwa node tersebut sedang diproses).
- Node 6 memiliki dua tetangga: node 2 dan node 7. Kedua node ini ditambahkan ke antrian BFS.
- BFS kemudian memeriksa node 2 dari antrian. Node 2 diberi label dengan jarak 1 (d=1) dan status warna GRAY, dan tetangganya (node 1, 4, dan 6) ditambahkan ke antrian.
- Node 1 yang ada di antrian selanjutnya diberi label dengan jarak 2 (d=2), dan status warna GRAY. Kemudian, tetangganya (node 2) tidak ditambahkan ke antrian karena telah diproses sebelumnya.
- Node 4 yang ada di antrian selanjutnya diberi label dengan jarak 2 (d=2), dan status warna GRAY. Tetangganya (node 3, 5) ditambahkan ke antrian.
- Node 3 yang ada di antrian selanjutnya diberi label dengan jarak 3 (d=3), dan status warna GRAY.
- Setelah semua node di antrian diproses, BFS selesai, dan hasilnya adalah sebagai berikut: (6,d=0) (2,d=1) (7,d=1) (1,d=2) (4,d=2) (9,d=2) (3,d=3) (5,d=3) (8,d=3)

Jadi, algoritma BFS menemukan node 3 dengan mengikuti lintasan dari node 6 ke node 2, dari node 2 ke node 1, dan dari node 2 ke node 4, lalu dari node 4 ke node 3. Node 3 diberi label dengan jarak 3 dari node awal (6).

Hasil Program:

```
(6,d=0) (2,d=1) (7,d=1) (1,d=2) (4,d=2) (9,d=2) (3,d=3) (5,d=3) (8,d=3)
```

Praktikum 5

1. Pelajari class EightPuzzleSearch, EightPuzzleSpace, dan Node.

Kelas Node:

- Kelas ini merepresentasikan sebuah simpul atau keadaan dalam ruang pencarian, khususnya dalam konteks penyelesaian masalah 8-puzzle.
- Bidang `state` adalah array yang berisi 9 bilangan bulat yang mewakili keadaan atau posisi ubin dalam papan permainan.
- Bidang `cost` adalah biaya yang terkait dengan mencapai simpul ini.
- Bidang `parent` adalah referensi ke simpul induk dalam pohon pencarian.
- Bidang `successors` adalah vektor yang berisi simpul-simpul anak.

Metode:

- Konstruktor `Node(int s[], Node parent)` menginisialisasi sebuah simpul dengan keadaan tertentu dan simpul induk.
- Metode `toString()` mengembalikan representasi string dari keadaan simpul.
- Metode `equals(Object node)` membandingkan dua simpul untuk memeriksa apakah mereka memiliki keadaan yang sama.
- Metode `getPath(Vector<Node> v)` mengembalikan jalur dari akar ke simpul saat ini dalam bentuk vektor.
- Metode `getPath()` mengembalikan jalur dalam vektor baru.

Kelas EightPuzzleSearch:

- Kelas ini bertanggung jawab untuk melakukan pencarian solusi untuk masalah 8-puzzle.
- Vektor `open` digunakan untuk menyimpan simpul-simpul yang perlu dieksplorasi, dan vektor `closed` menyimpan simpul-simpul yang telah dieksplorasi.
- Kelas ini memiliki dua fungsi heuristik, yaitu `h1Cost()` dan `h2Cost()`, untuk menghitung biaya simpul berdasarkan heuristik yang berbeda.
- Metode `hCost(Node node)` digunakan untuk memilih heuristik yang akan digunakan (h1 atau h2).

Metode:

- Metode `getBestNode(Vector nodes)` mengembalikan simpul dengan biaya terendah dari vektor simpul yang diberikan.
- Metode `getPreviousCost(Node node)` mengembalikan biaya simpul jika simpul tersebut ada dalam daftar `open` atau `closed`.
- Metode `printPath(Vector path)` mencetak jalur dari akar ke tujuan.

- Metode `run()` adalah metode utama yang menjalankan algoritma pencarian.

Kelas EightPuzzleSpace:

- Kelas ini mendefinisikan keadaan awal dan tujuan dalam masalah 8-puzzle, serta menghasilkan simpul-simpul penerus.
- Metode `getRoot()` mengembalikan keadaan awal sebagai simpul.
- Metode `getGoal()` mengembalikan keadaan tujuan sebagai simpul.
- Metode `getSuccessors(Node parent)` menghasilkan simpul-simpul penerus berdasarkan perpindahan ubin kosong.
- Metode `transformState()` digunakan untuk membuat simpul baru dengan menukar dua ubin dalam keadaan.

Kode ini mengimplementasikan algoritma pencarian untuk mencari solusi dalam masalah 8-puzzle dengan mengeksplorasi kemungkinan keadaan, mengevaluasi biaya berdasarkan heuristik, dan memilih simpul-simpul yang paling menjanjikan untuk dieksplorasi lebih lanjut. Pencarian berlanjut sampai solusi ditemukan atau semua kemungkinan telah dieksplorasi.

Hasil Program:

```
Root: 3 1 2 4 7 5 6 8 0


Solution found
 3 1 2 4 7 5 6 8 0
 3 1 2 4 7 5 6 0 8
 3 1 2 4 0 5 6 7 8
 3 1 2 0 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.8. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1.

Hasil Program:

```
Root: 3 1 2 4 7 5 6 8 0

Solution found

 3 1 2 4 7 5 6 8 0
 3 1 2 4 7 5 6 0 8
 3 1 2 4 0 5 6 7 8
 3 1 2 0 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 1 0 2 3 4 5 6 7 8
 1 2 0 3 4 5 6 7 8
 1 2 5 3 4 0 6 7 8
 1 2 5 3 0 4 6 7 8
 1 2 5 0 3 4 6 7 8
 0 2 5 1 3 4 6 7 8
 2 0 5 1 3 4 6 7 8
 2 3 5 1 0 4 6 7 8
 2 3 5 1 4 0 6 7 8
 2 3 0 1 4 5 6 7 8
 2 0 3 1 4 5 6 7 8
 0 2 3 1 4 5 6 7 8
 1 2 3 0 4 5 6 7 8
 1 2 3 4 0 5 6 7 8
BUILD SUCCESSFUL (total time: 1 second)
```

3. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.9. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1 dan 2.

Hasil Program:

```
Root: 1 5 3 4 6 8 2 7 0

Solution found
 1 5 3 4 6 8 2 7 0
 1 5 3 4 6 0 2 7 8
 1 5 0 4 6 3 2 7 8
 1 0 5 4 6 3 2 7 8
 1 6 5 4 0 3 2 7 8
 1 6 5 4 7 3 2 0 8
 1 6 5 4 7 3 2 8 0
 1 6 5 4 7 0 2 8 3
 1 6 0 4 7 5 2 8 3
 1 0 6 4 7 5 2 8 3
 1 7 6 4 0 5 2 8 3
 1 7 6 0 4 5 2 8 3
 0 7 6 1 4 5 2 8 3
 7 0 6 1 4 5 2 8 3
 7 6 0 1 4 5 2 8 3
 7 6 5 1 4 0 2 8 3
 7 6 5 1 0 4 2 8 3
 7 6 5 1 8 4 2 0 3
 7 6 5 1 8 4 0 2 3
 7 6 5 0 8 4 1 2 3
 7 6 5 8 0 4 1 2 3
BUILD SUCCESSFUL (total time: 52 minutes 23 seconds)
```

4. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 5.10. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1, 2, dan 3.

Hasil Program:

```
Root: 1 2 3 4 5 6 7 8 0


Solution found
 1 2 3 4 5 6 7 8 0
 1 2 3 4 5 6 7 0 8
 1 2 3 4 0 6 7 5 8
 1 2 3 4 6 0 7 5 8
 1 2 0 4 6 3 7 5 8
 1 0 2 4 6 3 7 5 8
 0 1 2 4 6 3 7 5 8
 4 1 2 0 6 3 7 5 8
 4 1 2 6 0 3 7 5 8
 4 1 2 6 5 3 7 0 8
 4 1 2 6 5 3 0 7 8
 4 1 2 0 5 3 6 7 8
 0 1 2 4 5 3 6 7 8
 1 0 2 4 5 3 6 7 8
 1 2 0 4 5 3 6 7 8
 1 2 3 4 5 0 6 7 8
 1 2 3 4 0 5 6 7 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Ubahlah initial dan goal state dari program dan class-class di atas sehingga bentuk initial dan goal statenya Gambar 5.11. Kemudian tentukan langkah-langkah mana saja sehinggapuzzlenya mencapai goal state.

Hasil Program: