Nama : Anggun Fia Febianingrum

NIM : 5311421010

Rombel : 1, Senin 09.00

PRAKTIKUM 5 ARTIFICIAL INTELLIGENCE

Tugas

1. Pelajari Class EighPuzzelSearch, EightPuzzleSpace, dan Node.
2. Ubahlah initila dan goal state dari program diatas sehingga bentuk initial dan goal statenya gambar 8. Kemudian tentukan langkah-langkah mana saja sehingga Puzzle nya mencapai goal state. Analisa dan bedakan dengan solusi pada point 1.
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.
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.
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 sehingga puzzlenya mencapai goal state.

Penyelesaian :

1. Berikut mengenai Class EighPuzzelSearch, EightPuzzleSpace, dan Node
   a) EightPuzzleSearch Class

      EightPuzzleSearch Class bertugas melakukan pencarian solusi untuk masalah puzzle 8 angka. Kelas ini bergantung pada kelas EightPuzzleSpace untuk mendapatkan informasi tentang ruang pencarian dan menggunakan kelas Node untuk mewakili simpul dalam pencarian. Selain itu, Kelas EightPuzzleSearch mengelola dua daftar, yaitu daftar terbuka (open) dan daftar tertutup (closed). Kelas ini menyediakan berbagai metode, termasuk untuk mendapatkan simpul terbaik dari daftar terbuka

(getBestNode), mendapatkan biaya sebelumnya dari simpul (getPreviousCost), mencetak jalur solusi (printPath), dan menjalankan algoritma pencarian (run).

b) EightPuzzleSpace Class

EightPuzzleSpace Class berperan sebagai wadah yang menangani operasi yang berhubungan dengan ruang pencarian untuk masalah puzzle 8 angka. Kelas ini menyediakan metode untuk mendapatkan simpul awal (getRoot), mengambil tujuan (getGoal), dan menghasilkan daftar suksesor dari suatu simpul tertentu (getSuccessors). Tugas utama kelas ini adalah mengelola konfigurasi awal dan tujuan dari puzzle 8 angka serta menentukan langkah-langkah yang dapat diambil dari suatu keadaan puzzle.

c) Node Class

Node Class adalah representasi dari simpul atau node dalam struktur data graf. Setiap simpul memiliki atribut state, yang merupakan array integer dengan panjang 9, yang menggambarkan keadaan dari puzzle 8 angka. Atribut lainnya mencakup biaya (cost) yang terkait dengan simpul, parent yang mengacu pada simpul pendahulu, dan daftar suksesor (successors) yang merupakan daftar simpul anak dari simpul saat ini. Kelas ini juga memiliki metode untuk mengubah simpul menjadi bentuk string, memeriksa kesetaraan, dan mengambil jalur dari simpul ke akar.

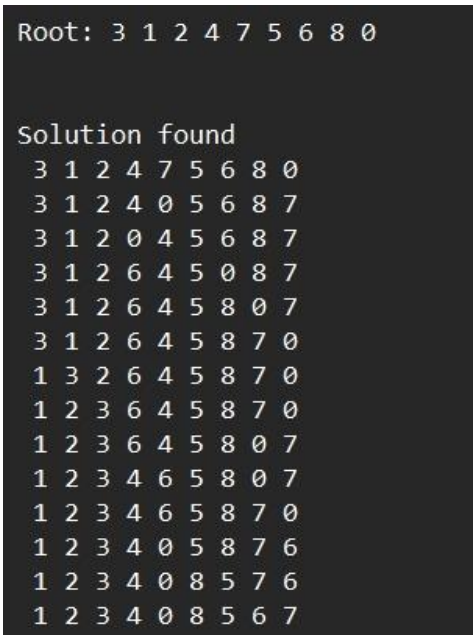
Run Program untuk mendapatkan hasil seperti dibawah ini :

```
PS C:\Users\user> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\user\AppData\Local
\Temp\vscodesws_66ada\jdt_ws\jdt.ls-java-project\bin' 'EightPuzzleSearch'

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
```

2. Mengubah initial dan goal state pada program seperti pada gambar 5.8 :

```
45
46  class EightPuzzleSpace {
47      public Node getRoot(){
48          return new Node(new int[]{3, 2, 1, 4, 7, 5, 6, 8, 0}, parent:null);
49      }
50
51      public Node getGoal() {
52          return new Node(new int[]{1, 2, 3, 4, 0, 8, 5, 6, 7}, parent:null);
53      }
```

Kemudian untuk mendapatkan hasil, maka Run Program dan akan dihasilkan sebagai berikut :



Setelah program di run, maka akan diketahui langkah-langkah dalam mencapai Goal State sebagai berikut:

a. Pada langkah awal menginisialisasi root state terlebih dahulu :

b. Kemudian, Algoritma akan mencari langkah-langkah dan menggeser angka yang bersebelahan dengan posisi kosong (0) ke arah yang seharusnya utnutk mrndapatkan Goal Statenya.
c. Pada setiap iterasi menampilkan langkah demi langkah yang di hasilkan
d. Kemudian langkah akan berhenti ketika sudah mendapatkan Goal State yang diinginkan:

Pada Setiap langkah yang diambil akan menunjukkan perpindahan angka ke angka untuk mencapai Goal State yang diinginkan. Pada hal ini Al goritma akan mencoba berbagai cara atau kombinasi dari perpindahan ubin-ubin angka untuk menemukan jalan yang paling tepat dan efesien untuk mrnuju pada goal State. Pada hasil diatas langkah yang diperlukan 13 langkah untuk menuju dari Root State ke Goal State.

Pada percobaan point 2 untuk mencapai goal state yang diinginkan diperlukan sebanyak 13 langkah sedangkan pada percobaan point 1 hanya diperlukan sebanyak 6 langkah untuk mencapai goal state yang diinginkan. Hal ini dikarenakan puzzle pada point 2 memiliki jarak yang lebih panjang dari root state ke goal state daripada point 1 atau memiliki posisi yang lebih teracak dari point 1. Oleh karena itu pada point 2 diperlukan langkah yang lebih banyak dalam mencapai goal state yang diinginkan.

3. Mengubah initial dan goal state pada program seperti pada gambar 5.9 :

```
46  class EightPuzzleSpace {
47      public Node getRoot(){
48          return new Node(new int[]{1, 5, 3, 4, 6, 8, 2, 7, 0}, parent:null);
49      }
50
51      public Node getGoal() {
52          return new Node(new int[]{7, 6, 5, 8, 0, 4, 1, 2, 3}, parent:null);
53      }
54
```

Kemudian untuk mendapatkan hasil, maka Run Program dan akan dihasilkan sebagai berikut :

```
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
```

Setelah program di run, maka akan diketahui langkah-langkah dalam mencapai Goal State sebagai berikut:

e. Pada langkah awal menginisialisasi root state terlebih dahulu :

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


Solution found
 1 5 3 4 6 8 2 7 0
```

f. Kemudian, Algoritma akan mencari langkah-langkah dan menggeser angka yang bersebelahan dengan posisi kosong (0) ke arah yang seharusnya utnutk mrndapatkan Goal Statenya.

g. Pada setiap iterasi menampilkan langkah demi langkah yang di hasilkan

h. Kemudian langkah akan berhenti ketika sudah mendapatkan Goal State yang diinginkan:

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

Pada Setiap langkah yang diambil akan menunjukkan perpindahan angka ke angka untuk mencapai Goal State yang diinginkan. Pada hal ini Al goritma akan mencoba

berbagai cara atau kombinasi dari perpindahan ubin-ubin angka untuk menemukan jalan yang paling tepat dan efesien untuk mrnuju pada goal State. Pada hasil diatas langkah yang diperlukan 20 langkah untuk menuju dari Root State ke Goal State.

Pada percobaan point 3 untuk mencapai goal state yang diinginkan diperlukan sebanyak 20 langkah sedangkan pada percobaan point 1 hanya diperlukan sebanyak 6 langkah dan percobaan point 2 diperlukan 13 langkah untuk mencapai goal state yang diinginkan. Hal ini dikarenakan puzzle pada point 3 memiliki jarak yang lebih panjang dari root state ke goal state daripada point 1 dan point 2 atau memiliki posisi yang lebih teracak dari point 1 dan point 2. Oleh karena itu pada point 3 diperlukan langkah yang lebih banyak dalam mencapai goal state yang diinginkan.

4. Mengubah initial dan Goal Staet pada program seperti pada gambar 5.10 :

```
45
46  class EightPuzzleSpace {
47      public Node getRoot(){
48          return new Node(new int[]{1, 2, 3, 4, 5, 6, 7, 8, 0}, parent:null);
49      }
50
51      public Node getGoal() {
52          return new Node(new int[]{1, 2, 3, 4, 0, 5, 6, 7, 8}, parent:null);
53      }
```

Kemudian untuk mendapatkan hasil, maka Run Program dan akan dihasilkan sebagai berikut :

```
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
```

Setelah program di run, maka akan diketahui langkah-langkah dalam mencapai Goal State sebagai berikut:

a) Pada langkah awal menginisialisasi root state terlebih dahulu :

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


Solution found
 1 2 3 4 5 6 7 8 0
```

b) Kemudian, Algoritma akan mencari langkah-langkah dan menggeser angka yang bersebelahan dengan posisi kosong (0) ke arah yang seharusnya utnutk mrndapatkan Goal Statenya.

c) Pada setiap iterasi menampilkan langkah demi langkah yang di hasilkan

d) Kemudian langkah akan berhenti ketika sudah mendapatkan Goal State yang diinginkan:

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

Pada Setiap langkah yang diambil akan menunjukkan perpindahan angka ke angka untuk mencapai Goal State yang diinginkan. Pada hal ini Al goritma akan mencoba berbagai cara atau kombinasi dari perpindahan ubin-ubin angka untuk menemukan jalan

yang paling tepat dan efesien untuk mrnuju pada goal State. Pada hasil diatas langkah yang diperlukan 16 langkah untuk menuju dari Root State ke Goal State.

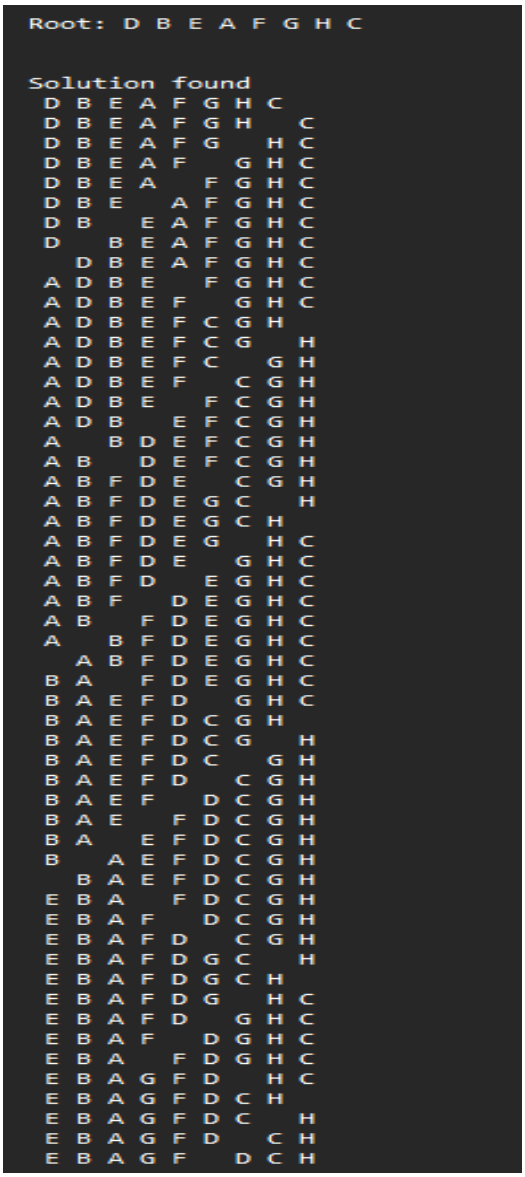
Pada percobaan ini dapat kita perhatikan bahwa pertama root dari keempat percobaan sangat berbeda. Root dalam percobaan 3 memiliki urutan angka yang lebih kompleks dibandingkan dengan Root dalam percobaan lainnya. Kedua, solusi dari keempat percobaan juga berbeda. Dalam percobaan 1, solusi adalah 1 6 5 8 0 4 2 7 3, dalam percobaan 2, solusi adalah 1 2 3 4 5 6 7 8 0, sedangkan pada percobaan 3, solusi adalah 7 6 5 8 0 4 1 2 3.

Kesimpulannya, meskipun keempat percobaan dapat mencapai solusi, perbedaan pada Root dan solusi menunjukkan bahwa program dimulai dari keadaan awal yang berbeda dan mencapai tujuan yang berbeda.

5. Untuk menyelesaikan puzzle seperti pada gambar 5.11 perlu dilakukan perubahan tipe data pada program yang awalnya ’int’ menjadi tipe data ’String’ seperti pada lampiran.
Ubah initial dan goal pada program seperti dengan gambar 5.10 sebagai berikut:

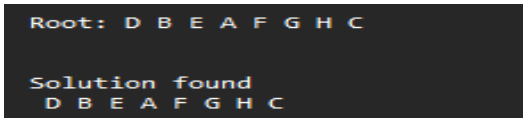
```
51  class EightPuzzleSpace {
52      public Node getRoot() {
53          return new Node(new String[] { "D", "B", "E", "A", "F", "G", "H", "C", " " }, parent:null);
54      }
55
56      public Node getGoal() {
57          return new Node(new String[] { "A", "H", "G", "B", " ", "F", "C", "D", "E" }, parent:null);
58      }
```

Run program dan didapatkan hasil sebagai berikut:



Setelah program di run, maka akan diketahui langkah-langkah dalam mencapai Goal State sebagai berikut:

a) Pada langkah awal menginisialisasi root state terlebih dahulu :



b) Kemudian, Algoritma akan mencari langkah-langkah dan menggeser angka yang bersebelahan dengan posisi kosong (0) ke arah yang seharusnya utnutk mrndapatkan Goal Statenya.

c) Pada setiap iterasi menampilkan langkah demi langkah yang di hasilkan

d) Kemudian langkah akan berhenti ketika sudah mendapatkan Goal State yang diinginkan, namun pada langkah ini tidak menemukan hasil dikarenakan node yang sangat teracak dan jarak antara root dan goal state sangat jauh.

Setiap langkah mewakili langkah perpindahan ubin yang dilakukan untuk mencapai goal state. Algoritma mencoba berbagai kombinasi perpindahan ubin untuk menemukan jalur yang paling efisien menuju goal state.

Pada percobaan ini untuk mencapai nilai goal state diperlikan langkah yang panjang, dikarenakan posisi pada root state dan goal state sangat jauh, hal ini juga disebabkan karena node sangat teracak. Sehingga pada percobaan ini belum ada hasil yang diperoleh (gagal).

LAMPIRAN

Kode program data type string

```
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

class Node {
   String[] state = new String[9];
   int cost;
   Node parent = null;
   List<Node> successors = new ArrayList<>();

   Node(String s[], Node parent) {
      this.parent = parent;
      for (int i = 0; i < 9; i++)
         state[i] = s[i];
   }

   public String toString() {
      StringBuilder s = new StringBuilder();
      for (String value : state) {
         s.append(value).append(" ");
      }
      return s.toString();
   }
```

```
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        Node node = (Node) obj;
        for (int i = 0; i < 9; i++) {
            if (node.state[i] != state[i])
                return false;
        }
        return true;
    }

    List<Node> getPath(List<Node> v) {
        v.add(0, this);
        if (parent != null)
            v = parent.getPath(v);
        return v;
    }

    List<Node> getPath() {
        return getPath(new ArrayList<>());
    }

}

class EightPuzzleSpace {
    public Node getRoot() {
        return new Node(new String[] { "D", "B", "E", "A", "F", "G", "H", "C", " " }, null);
    }

    public Node getGoal() {
        return new Node(new String[] { "A", "H", "G", "B", " ", "F", "C", "D", "E" }, null);
    }

    List<Node> getSuccessors(Node node) {
        List<Node> successors = new ArrayList<>();
        String[] state = node.state;

        // Cari indeks ubin kosong
        int emptyTileIndex = 0;
        for (int i = 0; i < state.length; i++) {
```

```
            if (state[i].equals(" ")) {
                emptyTileIndex = i;
                break;
            }
        }

        // Gerakan atas jika mungkin
        if (emptyTileIndex > 2) {
            String[] newState = state.clone();
            newState[emptyTileIndex] = newState[emptyTileIndex - 3];
            newState[emptyTileIndex - 3] = " ";
            successors.add(new Node(newState, node));
        }

        // Gerakan bawah jika mungkin
        if (emptyTileIndex < 6) {
            String[] newState = state.clone();
            newState[emptyTileIndex] = newState[emptyTileIndex + 3];
            newState[emptyTileIndex + 3] = " ";
            successors.add(new Node(newState, node));
        }

        // Gerakan kiri jika mungkin
        if (emptyTileIndex % 3 != 0) {
            String[] newState = state.clone();
            newState[emptyTileIndex] = newState[emptyTileIndex - 1];
            newState[emptyTileIndex - 1] = " ";
            successors.add(new Node(newState, node));
        }

        // Gerakan kanan jika mungkin
        if (emptyTileIndex % 3 != 2) {
            String[] newState = state.clone();
            newState[emptyTileIndex] = newState[emptyTileIndex + 1];
            newState[emptyTileIndex + 1] = " ";
            successors.add(new Node(newState, node));
        }

        return successors;
    }

}
```

```
public class EightPuzzleSearch {
   // Menambahkan definisi EightPuzzleSpace jika diperlukan

   EightPuzzleSpace space = new EightPuzzleSpace();

   List<Node> open = new ArrayList<>();
   List<Node> closed = new ArrayList<>();

   int h1Cost(Node node) {
      int cost = 0;
      for (int i = 0; i < node.state.length; i++) {
         if (node.state[i].equals(String.valueOf(i)))
            cost++;
      }
      return cost;
   }

   int h2Cost(Node node) {
      int cost = 0;
      String[] state = node.state;
      for (int i = 0; i < state.length; i++) {
         String v0 = String.valueOf(i), v1 = state[i];
         /* tidak menghitung ubin yang kosong */
         if (v1.equals(" "))
            continue;
         int row0 = i / 3, col0 = i % 3, row1 = getIndex(state, v1) / 3, col1 = getIndex(state, v1)
% 3;
         int c = (Math.abs(row0 - row1) + Math.abs(col0 - col1));
         cost += c;
      }
      return cost;
   }

   int getIndex(String[] arr, String value) {
      for (int i = 0; i < arr.length; i++) {
         if (arr[i].equals(value)) {
            return i;
         }
      }
      return -1;
   }

   /* boleh diubah dengan memakai heuristic h1 atau h2 */
```

```
int hCost(Node node) {
   return h2Cost(node);
}

Node getBestNode(List<Node> nodes) {
   int index = 0, minCost = Integer.MAX_VALUE;
   for (int i = 0; i < nodes.size(); i++) {
      Node node = nodes.get(i);
      if (node.cost < minCost) {
         minCost = node.cost;
         index = i;
      }
   }
   Node bestNode = nodes.remove(index);
   return (bestNode);
}

int getPreviousCost(Node node) {
   int cost = Integer.MAX_VALUE;
   for (Node n : open) {
      if (n.equals(node) && n.cost < cost) {
         cost = n.cost;
      }
   }
   for (Node n : closed) {
      if (n.equals(node) && n.cost < cost) {
         cost = n.cost;
      }
   }
   return cost;
}

void printPath(List<Node> path) {
   for (int i = 0; i < path.size(); i++) {
      System.out.print(" " + path.get(i) + "\n");
   }
}

void run() {
   Node root = space.getRoot();
   Node goal = space.getGoal();
   Node solution = null;
   open.add(root);
```

```
    System.out.print("\nRoot: " + root + "\n\n");
    while (open.size() > 0) {
      Node node = getBestNode(open);
      int pathLength = node.getPath().size();
      closed.add(node);
      if (node.equals(goal)) {
        solution = node;
        break;
      }

      List<Node> successors = space.getSuccessors(node);
      space.getSuccessors(node);
      for (int i = 0; i < successors.size(); i++) {
        Node successor = successors.get(i);
        int cost = hCost(successor) + pathLength + 1;
        int previousCost;
        previousCost = getPreviousCost(successor);
        boolean inClosed;
        inClosed = closed.contains(successor);
        boolean inOpen = open.contains(successor);

        if (!(inClosed || inOpen) || cost < previousCost) {
          if (inClosed)
            closed.remove(successor);
          if (!inOpen)
            open.add(successor);
          successor.cost = cost;
          successor.parent = node;
        }
      }
    }
    // new TreePrint(getTree(root));
    if (solution != null) {
      List<Node> path = solution.getPath();
      System.out.print("\nSolution found\n");
      printPath(path);
    }
  }

  public static void main(String[] args) {
    // melakukan pencarian
    new EightPuzzleSearch().run();
  }
```

}