MODUL 5
TEKNIK HEURISTIC SEARCH

5.1. Tujuan

Memperlihatkan kepada mahasiswa bagaimana menyelesaikan permasalahan pada game 8-puzzle dengan menggunakan algoritma heuristic search. Mahasiswa diharapkan mampu mengimplementasikan algoritma heuristic dengan menggunakan Java.

5.2. Dasar Teori

5.2.1. Teknik Pencarian Heuristic Search

Teknik blind search tidak selalu memecahkan masalah dengan baik. Waktu yang dibutuhkan ketika menemukan solusi atau memecahkan masalah terlalu lama dan juga memori yang dibutuhkan untuk menampung urutan-urutan solusi sangat besar akan menjadi kelemahan bagi algoritma ini. Kelemahan tersebut dapat diatasi ketika informasi-informasi tambahan yang diperoleh dari setiap langkah pencarian diidentifikasikan dan dijadikan sebagai penentu langkah berikutnya.

Salah satu teknik untuk meminimalisasikan kelemahan dari blind search adalah teknik heuristic. Heuristic merupakan suatu proses dimana pencarian solusi akan ditemukan dengan baik namun bisa juga kemungkinan tidak ada solusi. Teknik ini mmebutuhkan sebuah nilai untuk menentukan pencarian berikutnya. Nilai heuristic dapat ditentukan melalui fungsi heuristic.

Fungsi heuristic merupakan fungsi yang melakukan pemetaan dari diskripsi keadaan ke pengukur kebutuhan. Umumnya fungsi ini direpresentasikan ke dalam bentuk angka. Dalam ilmu Kecerdasan Buatan, heuristic dihadapkan dalam 2 keadaan dasar.

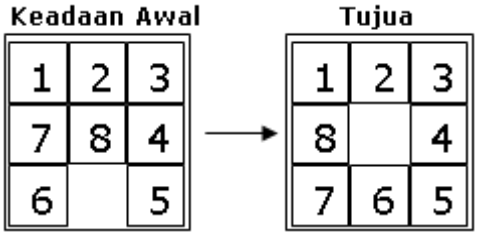
- Persoalan/problema yang mungkin memiliki solusi eksak, namun biaya perhitungan untuk menemukan solusi tersebut sangat tinggi dalam kebanyakan persoalan (seperti catur), ruang keadaan bertambah secara luar biasa seiring dengan jumlah.
- Persoalan yang mungkin tidak memiliki solusi eksak karena ambiquitas (ketidakpastian) mendasar dalam pernyataan persoalan atau data yang tersedia diagnosa medis merupakan salah satu contohnya.

Heuristi hanyalah sebuah cara menerka langkah berikutnya yang harus diambil dalam memecahkan suatu persoalan berdasarkan informasi yang ada/tersedia.

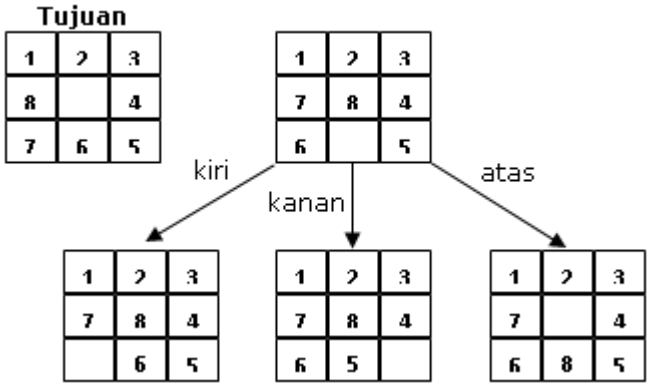
Pencarian Heuristik (*Heuristic Search*)

- Pencarian buta tidak selalu dapat diterapkan dengan baik, hal ini disebabkan waktu aksesnya yang cukup lama serta besarnya memori yang diperlukan.

- Kelemahan ini sebenarnya dapat diatasi jika ada informasi tambahan dari domain yang bersangkutan.
- Misalkan pada kasus 8-puzzle (Gambar 5.1)

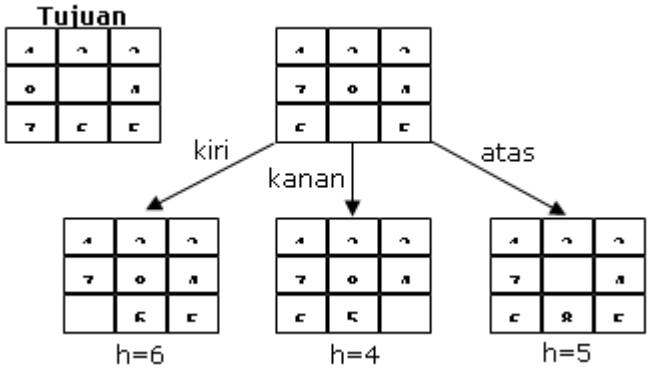


Gambar 5.1. Kasus 8-puzzle



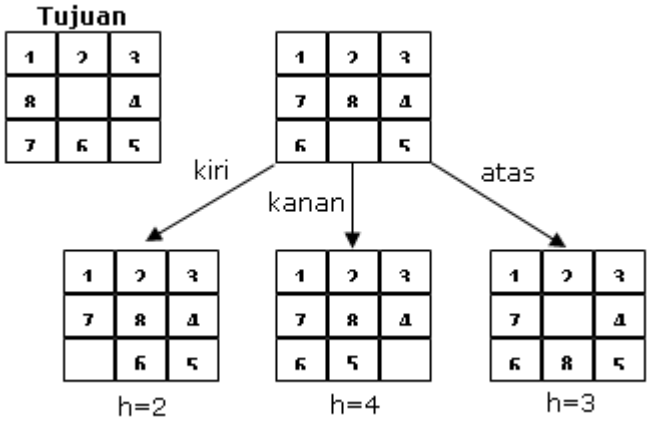
Gambar 5.2. Langkah awal kasus 8-puzzle

- Langkah pertama dari permainan tersebut seperti terlihat pada Gambar 5.2. Apabila digunakan pencarian buta, kita tidak perlu mengetahui operasi apa yang akan dikerjakan (sembarang operasi bisa digunakan).
- Pada pencarian heuristik perlu diberikan informasi khusus dalam domain tersebut.
- Informasi yang bisa diberikan, antara lain:
  a. Untuk jumlah ubin yang menempati posisi yang benar: jumlah yang lebih tinggi adalah yang lebih diharapkan (lebih baik), Gambar 5.3.



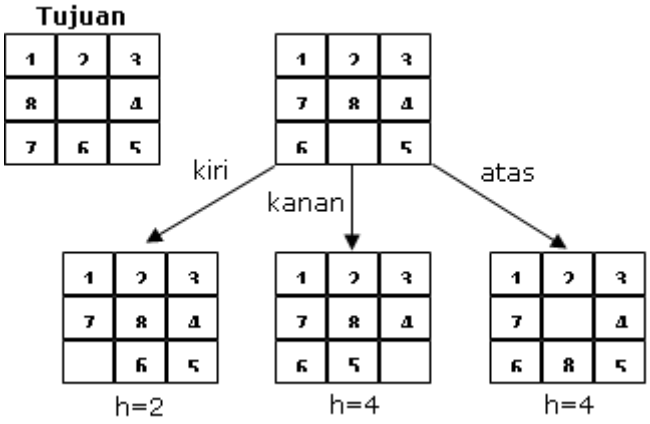
Gambar 5.3. Fungsi heuristik pertama kasus 8-puzzle

b. Untuk jumlah ubin yang menempati posisi yang salah: jumlah yang lebih kecil adalah yang diharapkan (lebih baik), Gambar 5.4.



Gambar 5.4. Fungsi heuristik kedua kasus 8-puzzle

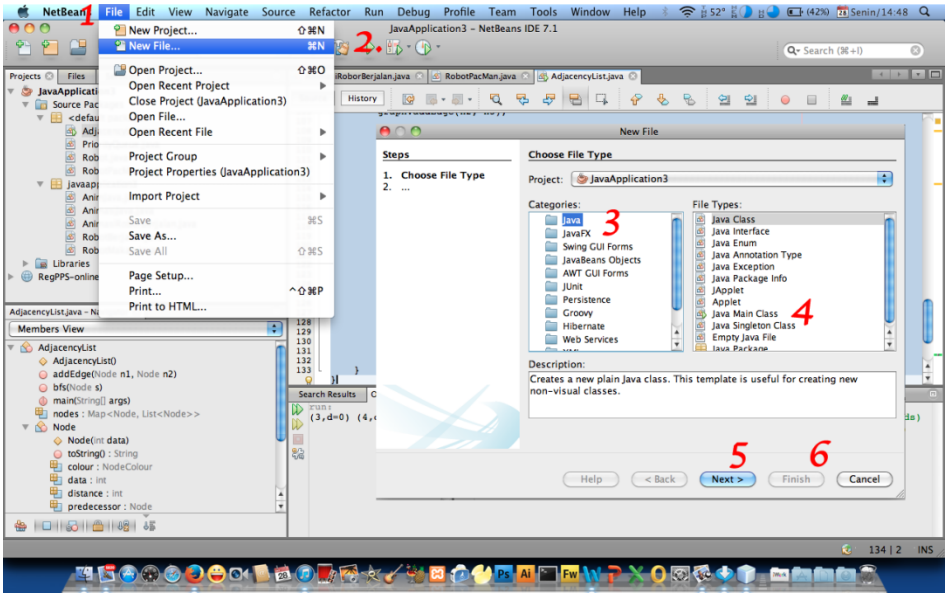
c. Menghitung total gerakan yang diperlukan untuk mencapai tujuan; jumlah yang lebih kecil adalah yang diharapkan (lebih baik), Gambar 5.5.



Gambar 5.5. Fungsi heuristik ketiga kasus 8-puzzle

5.3. Implementasi Algoritma Heurstic pada Permainan 8-Puzzle

Ketiklah *source code* Program 5.1 dan 5.2. pada perangkat lunak Netbeans 7.0 pada bagian teks editor Java Main Class. Pilih Menu “File”, lalu pilih submenu “New File”. Kemudian pilih Categories “Java” dengan FileTypes-nya adalah “Java Main Class”. Setelah itu, tekan tombol “Next” dan masukkan nama file EightPuzzleSearch, dan terakhir tekan tombol “Finish”. Alur langkah untuk membuat algoritma Heuristic dapat diikuti melalui Gambar 5.6



Gambar 5.6. Teks Editor Netbeans 7.0

File EightPuzzleSearch.java berisikan dua buah class yaitu class EightPuzzleSearch dan Node. Di dalam class Node memasukkan semua node ke dalam sebuah tree. Class ini juga dapat menghasilkan alur (path) dari root ke node tertentu yang diinginkan. Pemanggilan path ini dapat dilakukan melalui pemanggilan method getPath(). Berikut ini adalah pendeklarasian class Node.

```
class Node {
     int[] state = new int[9];
     int cost;
     Node parent = null;
     Vector<Node> successors = new Vector<Node>();

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

     public String toString() {
        String s = "";
        for (int i = 0; i < 9; i++) {
           s = s + state[i] + " "; }
        return s;
     }

     public boolean equals(Object node) {
        Node n = (Node)node;
        boolean result = true;
        for (int i = 0; i < 9; i++) {
           if (n.state[i] != state[i]) result = false; }
        return result;
     }

     Vector<Node> getPath(Vector<Node> v) {
        v.insertElementAt(this, 0);
        if (parent != null) v = parent.getPath(v);
        return v;
```

```
     }

     Vector<Node> getPath() {
        return getPath(new Vector<Node>());
     }
}
```

Program 5.1. Pendeklarasian Class Node.

Constructor Node menerima dua buah parameter yaitu urutan node children dan root dari children tersebut. Semua children dan node-node yang ada di dalam tree didefinisikan ke dalam tipe data integer (int). Class ini juga mempunyai method toString() yang berfungsi untuk mengubah node (dalam tipe data int) ke dalam bentuk string sehingga hasilnya akan berbentuk alur (path) dari initial state ke goal state dengan memanggil method *getPath()*.

Class EightPuzzleSearch memanggil fungsi utama *(main function)*, mendeskripsikan algoritma heuristic, menghitung cost *heuristic*, mencetak alur (path) dari root ke suatu node, dan menentukan node terbaik berdasarkan nilai dari fungsi heuristic.

```
public class EightPuzzleSearch {
     EightPuzzleSpace space = new EightPuzzleSpace();

     Vector<Node> open = new Vector<Node>();
     Vector<Node> closed = new Vector<Node>();

     int h1Cost(Node node) {
        int cost = 0;
        for (int i = 0; i < node.state.length; i++) {
           if (node.state[i] != i) cost++; }
        return cost;
     }
     int h2Cost(Node node) {
        int cost = 0;
        int state[] = node.state;
        for (int i = 0; i < state.length; i++) {
            int v0 = i, v1 = state[i];
            /*tidak menghitung ubin yang kosong */
            if (v1 == 0) continue;
            int row0 = v0 / 3, col0 = v0 % 3, row1 = v1 / 3, col1 = v1 % 3;
            int c=(Math.abs(row0-row1)+Math.abs(col0-col1));
            cost += c;
        }
        return cost;
     }

/*boleh diubah dengan memakai heuristic h1 atau h2 */
     int hCost(Node node) {
        return h2Cost(node);
     }

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

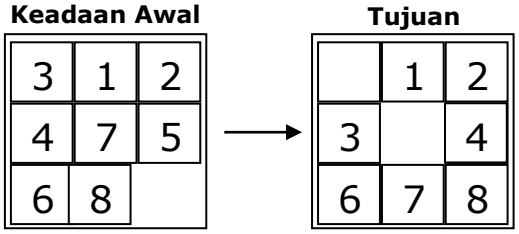
```
int getPreviousCost(Node node) {
    int i = open.indexOf(node);
    int cost = Integer.MAX_VALUE;
    if (i != -1) {
        cost = open.get(i).cost; }
    else {
        i = closed.indexOf(node);
        if (i != -1) cost = closed.get(i).cost; }
    return(cost);
}

void printPath(Vector path) {
    for (int i = 0; i < path.size(); i++) {
     System.out.print(" " + path.elementAt(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;
        }
        Vector<Node> successors =
                          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) {
       Vector path = solution.getPath();
       System.out.print("\nSolution found\n");
       printPath(path);
    }
 }

public static void main(String args[]) {
  // melakukan pencarian
```

```
        new EightPuzzleSearch().run();
    }
}
```

Program 5.2. Pendeklarasian Class EightPuzzleSearch.

Program 5.2 membentuk sebuah object space dengan bertipe class `EightPuzzleSpace`. Objek ini berfungsi untuk mendefinisikan initial state dan goal state sehingga penggua (user) dengan mudah menentukan contoh initial dan goal state pada 8-puzzle. Sebagai contoh pada kasus ini initial dan goal state dapat dilihat seperti Gambar 5.7. Bentuk initial dan goal state direpresentasikan ke dalam bentuk array berdimensi satu dan dideklarasikan seperti dibawah ini (lihat method *getRoot()* dan *getGoal()*).

```
int ex[] = {3, 1, 2, 4, 7, 5, 6, 8, 0};  // initial state
int state[] = {0, 1, 2, 3, 4, 5, 6, 7, 8}; // goal state
```



Gambar 5.7. Initial dan Goal state pada 8-puzzle

Kode lengkap dari class ini dapat dilihat pada Program 5.3. Ketiklah *source code* Program 5.3. pada perangkat lunak Netbeans 7.0 pada bagian teks editor Java Class. Pilih Menu “File”, lalu pilih submenu “New File”. Kemudian pilih Categories “Java” dengan FileTypes-nya adalah “Java Class”. Setelah itu, tekan tombol “Next” dan masukkan nama file EightPuzzleSpace, dan terakhir tekan tombol “Finish”.

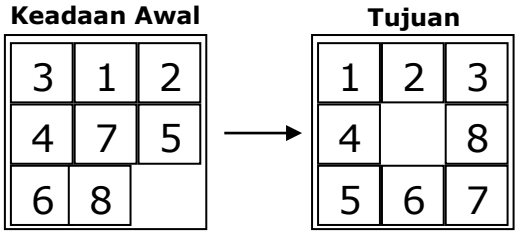
```
import java.util.Vector;
/*
 * Class EightPuzzleSpace dideklarasikan untuk menentukan
 * initial dan goal state serta mendapatkan path dari root
 * ke node tertentu
 */
/*
 * Modified by Irvanizam Zamanhuri
 */

public class EightPuzzleSpace {

    Node getRoot() {
        int ex[] = {3, 1, 2, 4, 7, 5, 6, 8, 0};
        // the Russell and Norvig eg
        int rn[] = {7, 2, 4, 5, 0, 6, 8, 3, 1};
        return new Node(ex, null);
    }

    Node getGoal() {
        int state[] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
        return new Node(state, null);
    }
```

```
    Vector<Node> getSuccessors(Node parent) {
        Vector<Node> successors = new Vector<Node>();
        for (int r = 0; r < 3; r++) {
           for (int c = 0; c < 3; c++) {
              /* ubin kosong disini */
              if (parent.state[(r * 3) + c] == 0) {
                 /* memindahkan ubin ke kiri */
                 if (r > 0) {
                    successors.add(transformState(r-1, c, r, c, parent)); }
                 /* memindahkan ubin ke kanan */
                 if (r < 2) {
                    successors.add(transformState(r+1, c, r, c, parent)); }
                  /* memindahkan ubin dari bawah */
                 if (c > 0) {
                    successors.add(transformState(r, c-1, r, c, parent)); }
                  /* memindahkan ubin dari atas */
                 if (c < 2) {
                    successors.add(transformState(r, c+1, r, c, parent)); }
              }
           }
        }
       /* used in getTree */
        parent.successors = successors;
        return successors;
    }

    Node transformState(int r0, int c0, int r1, int c1, Node parent) {
        int[] s = parent.state;
        int[] newState = {s[0], s[1], s[2], s[3], s[4], s[5], s[6], s[7],
s[8]};
        newState[(r1 * 3) + c1] = s[(r0 * 3) + c0];
        newState[(r0 * 3) + c0] = 0;
        return new Node(newState, parent);
     }
}
```

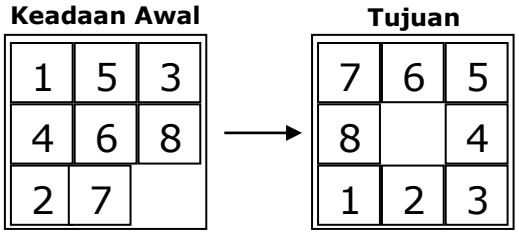
Program 5.3. Pendeklarasian Class EightPuzzleSpace.

Tugas:

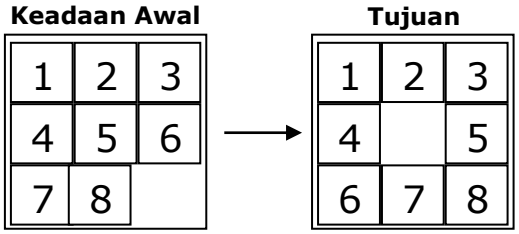
1. Pelajari class EightPuzzleSearch, EightPuzzleSpace, dan Node.
2. Ubahlah initial dan goal state dari program di atas sehingga bentuk initial dan goal statenya Gambar 8. Kemudian tentukan langkah-langkah mana saja sehingga puzzlenya mencapai goal state. Analisa dan bedakan dengan solusi pada point 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.



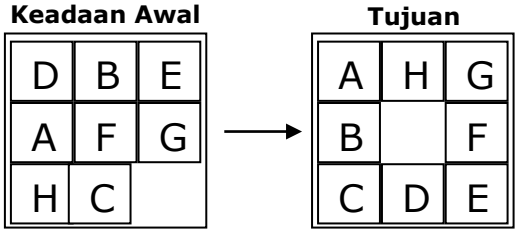
Gambar 5.8. Initial dan Goal state pada 8-puzzle 2



Gambar 5.9. Initial dan Goal state pada 8-puzzle 3



Gambar 5.10. Initial dan Goal state pada 8-puzzle 4



Gambar 5.11. Initial dan Goal state pada 8-puzzle 5