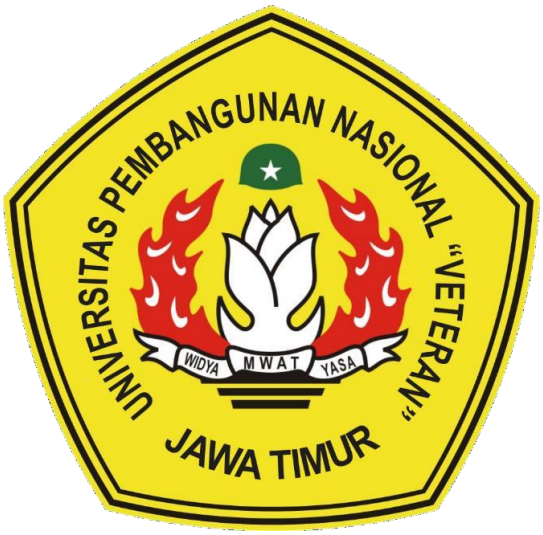
LAPORAN FINAL PROGRES

KECERDASAN BUATAN



NAMA KELOMPOK :

Achmad Rizky Fatur R. [19081010035]

Alfinas Agung Mujiono [19081010069]

Auliya Arkhan [19081010118]

Dede Ikhsan Dwi S. [19081010150]

M. Norman Hakam A. [19081010154]

PRODI INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"JAWA TIMUR

2021/2022

SURAT PERNYATAAN

Saya/kami yang bertanda tangan di bawah ini:

1) Achmad Rizky Fatur R. [19081010035]
2) Alfinas Agung Mujiono [19081010069]
3) Auliya Arkhan [19081010118]
4) Dede Ikhsan Dwi S. [19081010150]
5) M. Norman Hakam A. [19081010154]

Dengan ini menyatakan bahwa hasil pekerjaan yang saya/kami serahkan sebagai bagian dari penilaian tugas mata kuliah Kecerdasan Buatan adalah benar-benar karya orisinal saya/kami, bukan milik orang lain, dan tidak pernah digunakan dalam penilaian tugas yang lain dalam mata kuliah apapun, baik secara keseluruhan ataupun sebagian, di Universitas Pembangunan Nasional "Veteran" Jawa Timur ataupun di institusi lainnya.

Surabaya, 21 Desember 2021

TTD	TTD	TTD
		
Achmad Rizky Fatur R.	Alfinas Agung Mujiono	Auliya Arkhan

TTD	TTD
	
Dede Ikhsan Dwi S.	M. Norman Hakam A.

A. TUGAS I

1. Representasi Permasalahan (Problem Representation):

   a. Objek

      Objek yang ada dalam permasalahan adalah sebuah puzzle yang terdiri dari 24 objek yang teracak

   b. Keadaan (State)

      State pada permasalahan 24 puzzle ini adalah kombinasi dari 24 objek yang tersusun dalam matriks 5x5. Cara merepresentasikan state adalah dengan sebuah matriks berukuran 5x5.

   c. Ruang Keadaan (State Space)

      Dari sebuah state sembarang, terdapat sejumlah kemungkinan gerakan, yaitu kea rah kanan, kiri, atas, atau bawah.

   d. State Awal (Initial State)

      State awal dari permasalahan 24 puzzle ini adalah sebuah puzzle yang terdiri dari 24 objek yang teracak kemudian disusun pada matriks 5x5.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
0	17	18	19	20
16	21	22	23	24

Gambar 1 (state awal)

   e. State Tujuan (Goal State)

      State tujuan dari permasalahan 24 puzzle ini adalah terurutnya puzzle dari node 1 (node pertama), 2, 3, sampai dengan node 24 dan 0 (note terakhir) pada sebuah matriks 5x5.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	0

Gambar 2 (state tujuan)

   f. Biaya (Path Cost)

Setiap langkah/gerakan membutuhkan biaya sebesar sesuai dengan kedalaman dari state puzzle tersebut.

2. Fungsi Heuristik yang Anda gunakan:
   a. Deskripsi

      Fungsi heuristic yang kami gunakan adalah H2A dan H4, yaitu menghitung total jarak Manhattan dari state awal ke state tujuan dengan menggunakan algoritma A*.

   b. Pseudocode

      Manhattan Distance

```
DEFINE FUNCTION heuristic_manhattan_distance(self):
        SET distance TO 0
        FOR i IN range(self.PUZZLE_NUM_ROWS):
            FOR j IN range(self.PUZZLE_NUM_COLUMNS):
                SET i1, j1 TO
self._get_coordinates(self.position[i][j],
self.PUZZLE_END_POSITION)
                distance += abs(i - i1) + abs(j - j1)
        RETURN distance
```

   c. Penjelasan singkat pseudocode

      - Manhattan Distance

        Pertama, definisikan variable distance dengan nilai 0, lalu lakukan perulangan didalam perulangan dengan variable i dan j dengan nomor kolom dan nomor baris dari puzzle. Lalu set variable i1 dan j1 dengan nilai koordinat dari masing-masing node. Maka manhattan distance dihasilkan dari hasil pengurangan dari i dan j awal ke nilai i dan j yang benar.

   d. Implementasi pseudocode

```
    def _calculate_new_heuristic(move, end_node):
        return move.heuristic_manhattan_distance() -
end_node.heuristic_manhattan_distance()
    def do_algorithm(self):
        queue = [[self.start.heuristic_manhattan_distance(), self.start]]
        expanded = []
        num_expanded_nodes = 0
        path = None
        while queue:
            i = 0
            for j in range(1, len(queue)):
                if queue[i][0] > queue[j][0]:  # minimum
                    i = j
            path = queue[i]
            queue = queue[:i] + queue[i + 1:]
            end_node = path[-1]
            if end_node.position == end_node.PUZZLE_END_POSITION:
                break
            if end_node.position in expanded:
                continue
            for move in end_node.get_moves():
                if move.position in expanded:
                    continue
                new_path = [path[0] + self._calculate_new_heuristic(move,
end_node)] + path[1:] + [move]
                queue.append(new_path)
                expanded.append(end_node.position)
            num_expanded_nodes += 1
        self.num_expanded_nodes = num_expanded_nodes
        self.solution = path[1:]
class Puzzle:
    def _init_(self, position):
        """
        :param position: a list of lists representing the puzzle matrix
        """
        self.position = position
        self.PUZZLE_NUM_ROWS = len(position)
        self.PUZZLE_NUM_COLUMNS = len(position[0])
        self.PUZZLE_END_POSITION = self._generate_end_position()
```

3. Algoritma Informed Search (A*):
    a. Deskripsi

    Algoritma A* adalah salah satu algoritma yang dapat digunakan untuk menentukan total minimum biaya lintasan. Algoritma ini dapat memberikan solusi yang optimal jika digunakan saat kondisi yang tepat. Pada studi kasus

24 puzzle ini, rute terpendeknya adalah jarak tiap-tiap puzzle ke kotak kosong pada puzzle.

b. Pseudocode

```
countRow <- 0
countCol <- 0
For i <- 0 sampai 4
For j <- 0 sampai 4
//mengubah posisi kosong menjadi 25
//untuk mempermudah program
If tiles = 0
tiles = 25
EndIf
Case berdasarkan j
Case j = 0
If tiles-1>0 atau (tiles-1) tidak habis dibagi 5
countCol <- countCol+1
EndIf
Case j = 4
If tiles tidak habis dibagi 5
countCol <- countCol+1
EndIf
EndCase
Case berdasarkan i
Case i = 0
If tiles>5
countRow <- countRow+1
EndIf
```

```
Case i = 1
If tiles>10 atau tiles<6
countRow <- countRow+1
EndIf
Case i = 2
If tiles>15 atau tiles<11
countRow <- countRow+1
EndIf
Case i = 3
If tiles>20 atau tiles<16
countRow <- countRow+1
EndIf
Case i = 4
If tiles<21
countRow <- countRow+1
EndIf
EndCase
EndFor
EndFor
return countCol + countRow
```

c. Penjelasan singkat pseudocode

Pertama definisikan variabel count yang bernilai 0, lalu dilakukan perulangan di dalamperulangan yang masing masing iterasinya sebanyak 5 kali (mulai dari 0 sampai 4), ditandai dengan variabel i dan j, lakukan pengujian apakah matriks i,j bernilai 0, jika iya maka akan diubah nilainya menjadi 25 untuk memudahkan perhitungan. Dalam perulangan lakukan pengujian nilai j dan i, jika i adalah 0, nilai countRow bertambah 1 jika nilai matriks lebih dari 5. Ketika i adalah 1, nilai countRow bertambah 1 jika nilai matriks lebih dari 10 dan kurang dari 6. Ketika i adalah 2, nilai countRow bertambah 1 jika nilai matriks lebih dari 15 dan kurang dari 11.Ketika i adalah 3, nilai countRow bertambah 1 jika nilai matriks lebih dari 20 dan kurang dari 16.Ketika i adalah 1, nilai countRow bertambah 1 jika nilai matriks kurang dari 21. Ketika j adalah 0, nilai countCol bertambah 1 jika nilai matriks dikurang 1 lebih besar dari 0 atau nilai matriks dikurang 1 tidak habis dibagi 5. Ketika j adalah 1, nilai countCol bertambah 1 jika nilai matriks dikurang 2 lebih besar dari 0 atau nilai matriks dikurang 2 tidak habis dibagi 5.Ketika j adalah 2, nilai countCol bertambah 1 jika nilai matriks dikurang 3 lebih besar dari 0 atau nilai matriks dikurang 3 tidak habis dibagi 5.Ketika j adalah 3, nilai countCol bertambah 1 jika nilai matriks dikurang 4 lebih besar dari 0 atau nilai matriks dikurang 4 tidak habis dibagi 5.Ketika j adalah 4, nilai countCol bertambah 1 jika nilai matriks tidak habis dibagi 5. Setelah perulangan selesai, nilai countCol dan countRow akan ditambah dan dikembalikan.

d. Implementasi pseudocode

```
public int wrongPiecePropagation(){
        int countRow = 0;
        int countCol = 0;
        for (int i=0; i<5; i++) {
            for (int j=0; j<5; j++) {

                int tempTiles = tiles[i][j];
                if (tempTiles == 0){
                    tempTiles = 25;
                }

                switch (j){
                    case 0: if (tempTiles-1>0 ||
(tempTiles-1)%5!=0) countCol++;
                        break;
                    case 1: if (tempTiles-2>0 ||
(tempTiles-2)%5!=0) countCol++;
                        break;
                    case 2: if (tempTiles-3>0 ||
(tempTiles-3)%5!=0) countCol++;
                        break;
                    case 3: if (tempTiles-4>0 ||
(tempTiles-4)%5!=0) countCol++;
                        break;
                    case 4: if ((tempTiles)%5!=0)
countCol++;
                        break;
                }
                switch (i){
                    case 0: if (tempTiles>5) countRow++;
                            break;
                    case 1: if (tempTiles>10 ||
tempTiles<6) countRow++;
                            break;
                    case 2: if (tempTiles>15 ||
tempTiles<11) countRow++;
                            break;
                    case 3: if (tempTiles>20 ||
tempTiles<16) countRow++;
                            break;
                    case 4: if (tempTiles<21) countRow++;
                            break;
                }
            }
        }
        return countCol+countRow;
    }
```

4. Penanganan State Berulang (Repeated States):
    a. Deskripsi

    State berulang adalah sebuah kondisi dimana kondisi tersebut sudah pernah dikunjungi atau terjadi sebelumnya, sehingga jika state berulang ini terjadi, program tidak akan melakukan pengecekan lebih pada state tersebut.

b. Penjelasan State Berulang pada 24 Puzzle

State berulang pada 24 puzzle akan terjadi ketika sebuah kombinasi puzzle sudah dilakukan pengecekan sebelumnya, maka program tidak akan menambahkan state tersebu ke dalam list.

c. Pseudocode

```
Function checkRepeat(closedList, child)
    noRepeated <- {}
    For child
        If child not in closedList
            noRepeated.Add <- child
    return noRepeated
end function
```

d. Penjelasan singkat pseudocode

Penjelasan dari pseudocode untuk penanganan state berulang, pertama didefinisikan variabel closedList, list state yang sudah dilalui oleh program dan variabel child yang berisi anak anak dari state sebelumnya. Buat variabel baru bernama noRepeated yang akan digunakan untuk menampung anak yang bukan state berulang. Dilakukan perulangan sebanyak jumlah anak, lalu diuji apakah anak ada di dalam closedList, jika tidak maka anak akan ditambahkan ke list noRepeated, lalukan sampai anaknya habis. Lalu list noRepeated akan dikembalikan.

e. Implementasi pseudocode

```
public static ArrayList<Node>
checkRepeat(HashMap<Integer,Node> closedList,List<Node>
expanded){

   ArrayList<Node> nodes = new ArrayList<>();

   for (Node expand :

            expanded) {

       if
(!closedList.containsKey(expand.getState().hashCode())){

            nodes.add(expand);

       }

   }

   return nodes;

}
```

5. Evaluasi

a. Effective Branching Factors. Masukkan hasil perhitungan dari fungsi `effectiveBranchingFactor` dari program Anda pada tabel ini.

kedalaman	***H2A***	***H4A****
2	***2.1924***	***2.1924***
4	***1.6068***	***1.5706***
6	***1.3253***	***1.3253***
8	***1.2604***	***1.9960***
10	***1.3110***	***1.5399***
12	***1.1792***	***1.4270***

Kedalaman merupakan kedalaman tree dari solusi yang ditemukan. Program yang Anda buat harus dapat memberikan keterangan kedalaman dari solusi.

b. Hasil Analisis

c. Kesimpulan

Fungsi heuristik digunakan untuk mengoptimalkan sebuah algoritma, namun ada kalanya fungsi heuristik justru menjadi pedang bermata dua untuk programnya sendiri jika sebuah heuristik tidak dibuat dengan baik dan tidak sesuai dengan kondisi yang memenuhi dari setiap algoritma. Membuat heuristik yang baik dan optimal dapat meningkatkan kinerja dari sebuah program, baik dari segi waktu komputasi maupun pada manajemen penyimpanan. Untuk membuat perhitungan heuristik yang baik perlu dilakukan research dan pemahaman mendalam terkait kasus yang ingin diselesaikan dan harus sesuai dengan kondisi dari setiap studi kasus yang ada.

B. TUGAS II

1. Tujuan

    1) Project ini bertujuan untuk mengetahui hasil dari training dataset

    2) Jaringan syaraf tiruan atau *Neural Networks* ditujukan untuk memprediksi apakah hutan di Algeria berpotensi terjadi kebakaran atau tidak berdasarkan sejumlah informasi lain pada atribut, seperti: suhu, kelembaban relatif, kecepatan angin, indeks *Fine Fuel Moisture Code*, dsb. Serta *Hyperparameter* yang digunakan adalah *Hidden Layer* dan Variasi Data Training dan Testing untuk mengoptimalisasi hasil akurasi dari data training.

2. Penjelasan tentang optimisasi hasil yang dilakukan

    Dengan menggunakan *hyperparameter Hidden Layer* (HL) dan Variasi Data Training dan Testing, dilakukan beberapa kali percobaan, yaitu dengan mengubah jumlah Hidden Layer, jumah node, dan iterasi percobaan untuk mendapatkan hasil akurasi dari data training dan data test yang berbeda.

3. Penjelasan tentang kinerja system pada data training dan data test dengan optimisasi hasil yang telah dilakukan (sajikan dalam bentuk tabel dan beri penjelasan)

No. Percobaan	Hyperparameter 1 (Hidden Layer)	Hyperparameter 2 (VT)	Average Training Error	Akurasi
1	Hidden Layer 1 = 32 node	Data Training = 200 Data Test = 200		94%
2	Hidden Layer 1 = 32 node Hidden Layer 2 = 24 node	Data Training = 122 Data Test = 122		86.066%

Percobaan pertama dilakukan dengan menggunakan 1 Hidden Layer dan Variasi Data Training sebanyak 200 & Data Testing sebanyak 122 dengan iterasi percobaan sebanyak 200 kali. Didapatkan tingkat akurasi training cukup tinggi yaitu 94% dan Average Training Error sebanyak

Percobaan kedua dilakukan dengan menggunakan 2 Hidden Layer dan Variasi Data Training sebanyak 122 & Data Testing sebanyak 122 dengan iterasi percobaan sebanyak 300 kali. Didapatkan tingkat akurasi training yang sama dengan percobaan sebelumnya yaitu 86.006%, hal. Tetapi untuk percobaan kedua didapatkan Average Training Error yang lebih kecil dari yang sebelumnya yaitu 0.384512.

4. Kesimpulan

Optimisasi hasil *training* dataset dengan menggunakan algoritma *Neural Network* bertujuan untuk meningkatkan hasil akurasi. Hal ini dapat dilakukan dengan mengubah sejumlah nilai dari *hyperparameter*, yaitu *learning rate*, epoch, variasi training, dan jumlah hidden layer. Pada project tugas II ini dihasilkan nilai *Average Training Error* paling rendah dan tingkat Akurasi paling tinggi dengan nilai 14% untuk *Average Training Error* dan 95% untuk Akurasi dengan jumlah variasi training sebanyak 0,5 dan epoch 50. Maka, dapat disimpulkan semakin besar nilai Variasi Training dan Epoch akan meningkatkan Akurasi dan menghasilkan *Average Training Error* yang kecil.