MENYELESAIKAN 8-PUZZLES DENGAN METODE INFORMED SEARCH

Anggota Kelompok :

Abdurrahman Farimza	*5025201125*
Kartika Diva Asmara Gita	*5025211039*
Hana Maheswari	*5025211182*

INFORMED SEARCH

Informed Search Algorithm / Heuristic Search merupakan algoritma pencarian menggunakan pengetahuan yang spesifik kepada permasalahan yang dihadapi selain dari definisi masalahnya itu sendiri sehingga lebih hemat waktu dan biaya. ada 2 metode Informed Search Algorithm yang terkenal yaitu, A* Search (A-Star Search) dan Greedy Best First Search.

A* Search

A* Search merupakan algoritma dengan tingkat keefektifan dan keakuratan yang tinggi serta bersifat complete. Hal ini karena A* Search juga mempertimbangkan biaya yang dibutuhkan dari **simpul awal menuju suatu simpul yang akan dibangkitkan.** Tidak hanya bergantung pada nilai heuristic yang dimiliki oleh tiap simpul yang akan dibangkitkan untuk mencapai simpul tujuan. Pencarian dengan A* melihat kepada kombinasi **nilai dari pathnya** yaitu g(n) dengan **nilai estimasi** yaitu h(n).

$$F(n) = g(n) + h(n)$$

$$F(n) = g(n) + h(n)$$

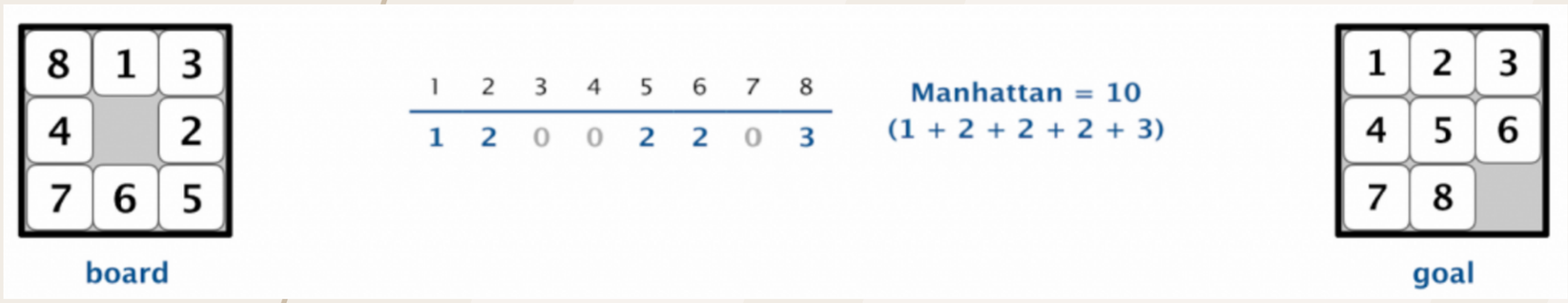
- g(n) : kedalaman / depth dari node
- h(n) : Manhattan distance

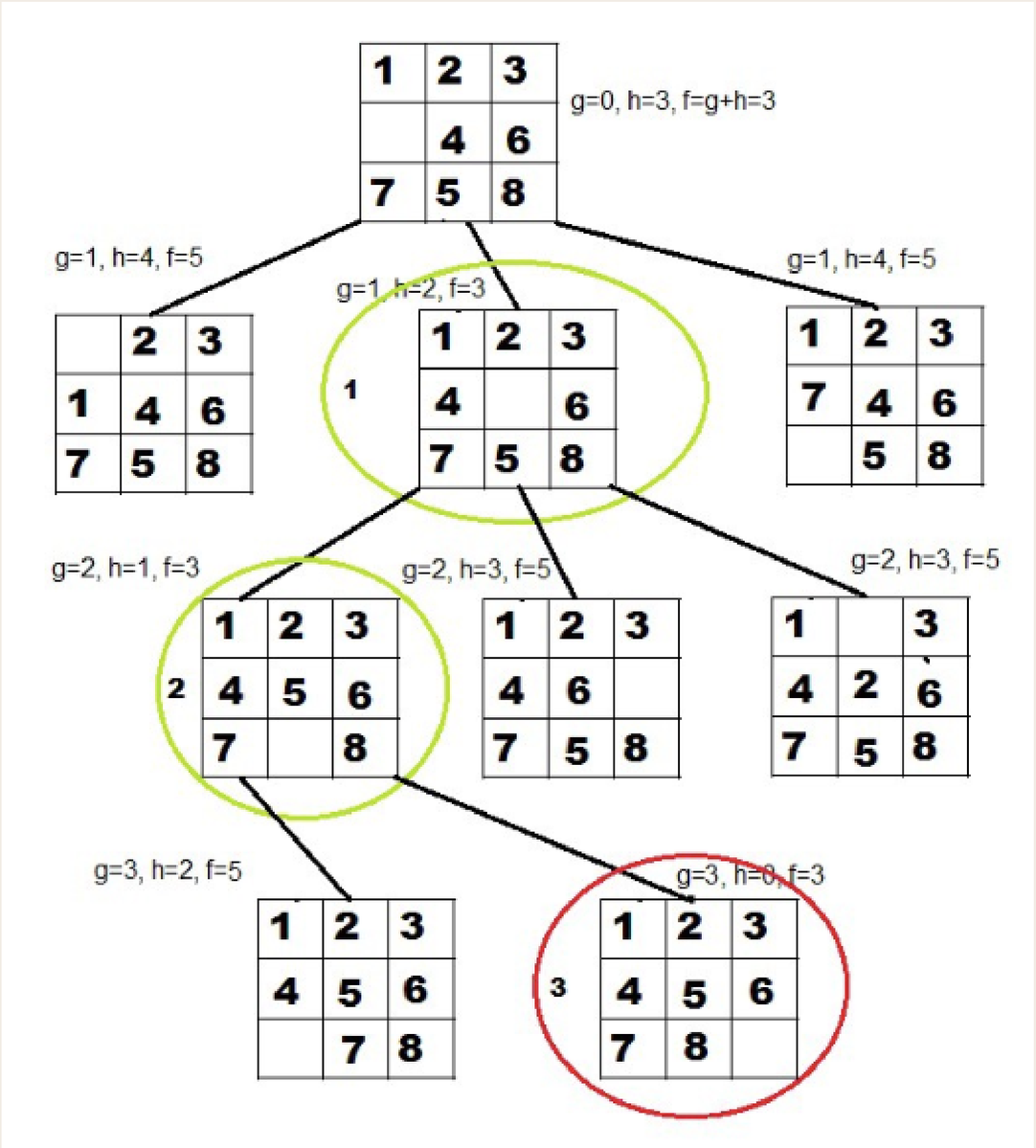
function manhattan_distance(node, goal) =

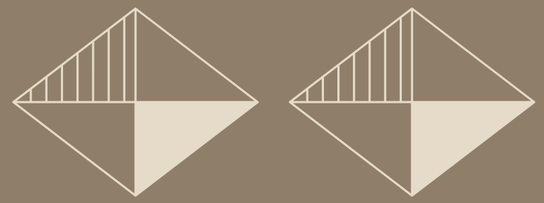
dx = abs(node.x - goal.x)

dy = abs(node.y - goal.y)

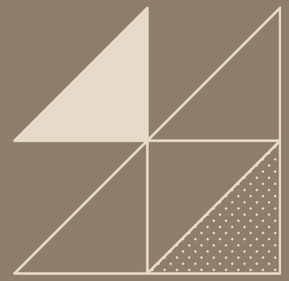
return dx + dy




g=0, h=3, f=g+h=3
g=1, h=4, f=5
g=1, h=2, f=3
g=1, h=4, f=5
1
g=2, h=1, f=3
g=2, h=3, f=5
g=2, h=3, f=5
2
g=3, h=2, f=5
g=3, h=0, f=3
3



H(1)
MISPLACED TILES

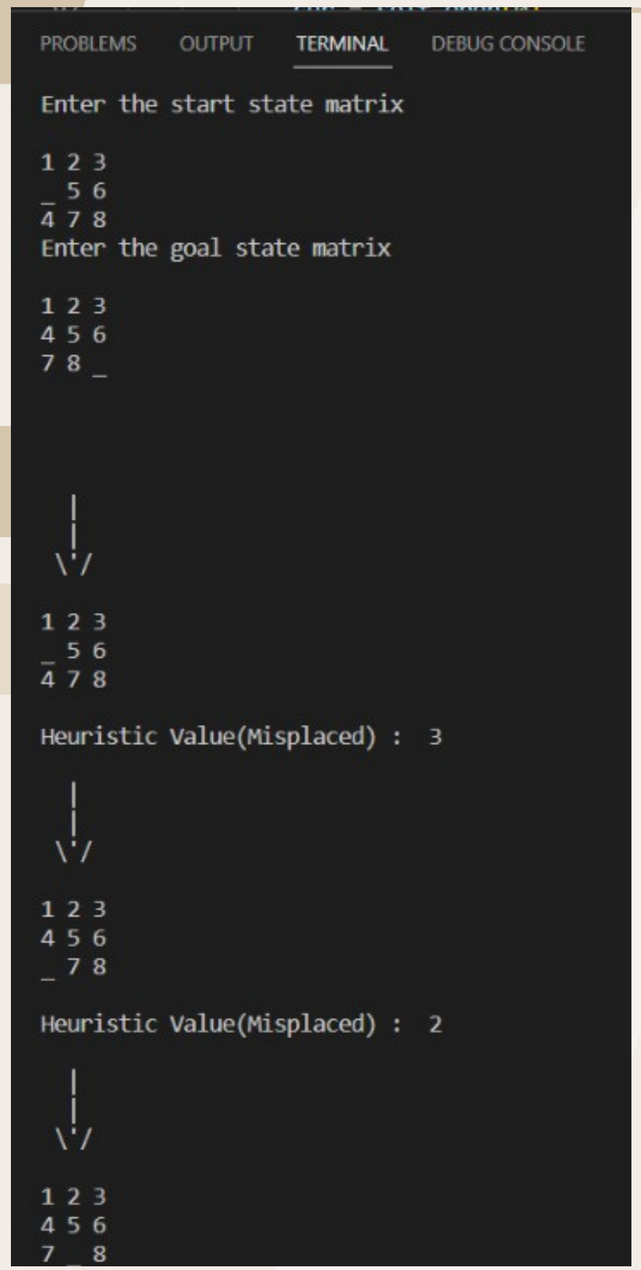


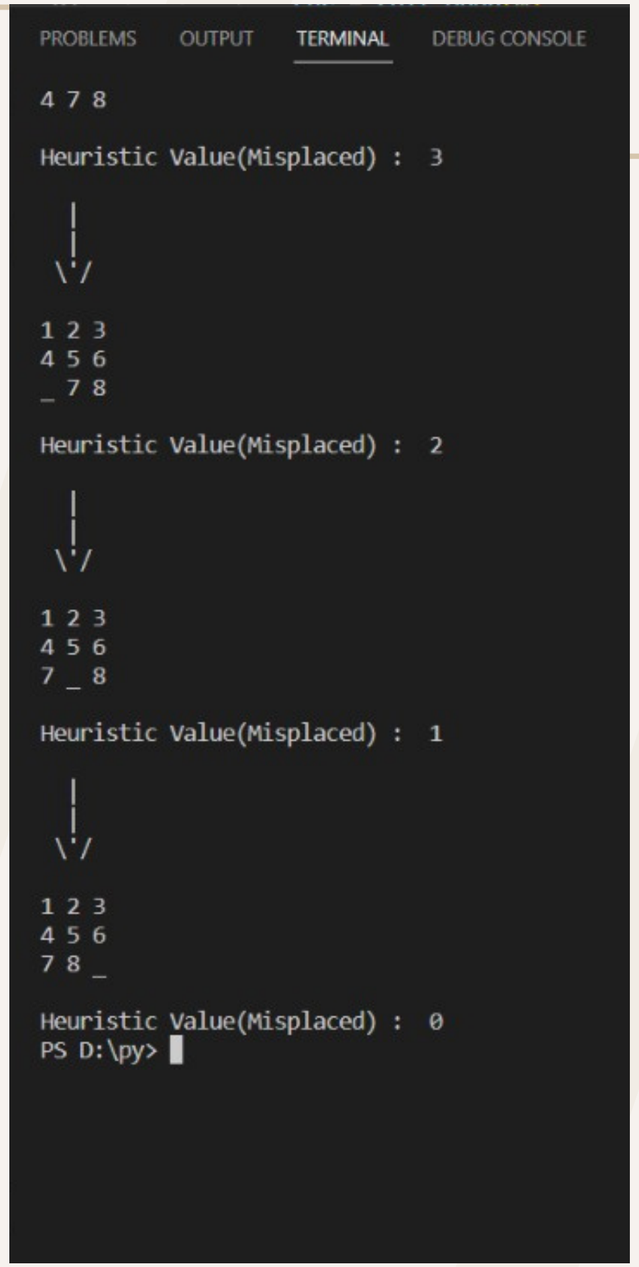
```python
class Node:
    def __init__(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        x,y = self.find(self.data,'_')
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children
```

```python
def shuffle(self,puz,x1,y1,x2,y2):
    if x2 ≥ 0 and x2 < len(self.data) and y2 ≥ 0 and y2 < len(self.data):
        temp_puz = []
        temp_puz = self.copy(puz)
        temp = temp_puz[x2][y2]
        temp_puz[x2][y2] = temp_puz[x1][y1]
        temp_puz[x1][y1] = temp
        return temp_puz
    else:
        return None

def copy(self,root):
    temp = []
    for i in root:
        t = []
        for j in i:
            t.append(j)
        temp.append(t)
    return temp

def find(self,puz,x):
    for i in range(0,len(self.data)):
        for j in range(0,len(self.data)):
            if puz[i][j] == x:
                return i,j
```

```python
class Puzzle:
    def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,start,goal):
        return self.h(start.data,goal)+start.level

    def h(self,start,goal):
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    temp += 1
        return temp
```

```python
    def process(self):
        print("Enter the start state matrix \n")
        start = self.accept()
        print("Enter the goal state matrix \n")
        goal = self.accept()

        start = Node(start,0,0)
        start.fval = self.f(start,goal)
        self.open.append(start)
        print("\n\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            print("\nHeuristic Value(Misplaced) : ",self.h(cur.data,goal))
            if(self.h(cur.data,goal) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,goal)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]

            self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)
puz.process()
```


PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Enter the start state matrix
1 2 3
_ 5 6
4 7 8
Enter the goal state matrix
1 2 3
4 5 6
7 8 _
| | \'/
1 2 3
_ 5 6
4 7 8
Heuristic Value(Misplaced) : 3
| | \'/
1 2 3
4 5 6
_ 7 8
Heuristic Value(Misplaced) : 2
| | \'/
1 2 3
4 5 6
7 _ 8


PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
4 7 8
Heuristic Value(Misplaced) : 3
| | \'/
1 2 3
4 5 6
_ 7 8
Heuristic Value(Misplaced) : 2
| | \'/
1 2 3
4 5 6
7 _ 8
Heuristic Value(Misplaced) : 1
| | \'/
1 2 3
4 5 6
7 8 _
Heuristic Value(Misplaced) : 0
PS D:\py>

H(2)
MANHATTAN DISTANCE

```python
import argparse
import time
import timeit

#Information
class PuzzleState:
    def __init__(self, state, parent, move, depth, cost, key):
        self.state = state
        self.parent = parent
        self.move = move
        self.depth = depth
        self.cost = cost
        self.key = key
        if self.state:
            self.map = ''.join(str(e) for e in self.state)
    def __eq__(self, other):
        return self.map == other.map
    def __lt__(self, other):
        return self.map < other.map
    def __str__(self):
        return str(self.map)

#Global variables
GoalState = [0, 1, 2, 3, 4, 5, 6, 7, 8]
GoalNode = None # at finding solution
NodesExpanded = 0 #total nodes visited
MaxSearchDeep = 0 #max deep
```

```python
#AST
def ast(startState):

    global MaxSearchDeep, GoalNode

    #transform initial state to calculate Heuritic
    node1 = ""
    for poss in startState:
        node1 = node1 + str(poss)

    #calculate Heuristic and set initial node
    key = Heuristic(node1)
    boardVisited= set()
    Queue = []
    Queue.append(PuzzleState(startState, None, None, 0, 0, key))
    boardVisited.add(node1)

    while Queue:
        Queue.sort(key=lambda o: o.key)
        node = Queue.pop(0)
        if node.state == GoalState:
            GoalNode = node
            return Queue
        posiblePaths = subNodes(node)
        for path in posiblePaths:
            thisPath = path.map[:]
            if thisPath not in boardVisited:
                key = Heuristic(path.map)
                path.key = key + path.depth
                Queue.append(path)
                boardVisited.add(path.map[:])
                if path.depth > MaxSearchDeep:
                    MaxSearchDeep = 1 + MaxSearchDeep
```

```python
#Heuristic: distance to root numbers
values_0 = [0,1,2,1,2,3,2,3,4]
values_1 = [1,0,1,2,1,2,3,2,3]
values_2 = [2,1,0,3,2,1,4,3,2]
values_3 = [1,2,3,0,1,2,1,2,3]
values_4 = [2,1,2,1,0,1,2,1,2]
values_5 = [3,2,1,2,1,0,3,2,1]
values_6 = [2,3,4,1,2,3,0,1,2]
values_7 = [3,2,3,2,1,2,1,0,1]
values_8 = [4,3,2,3,2,1,2,1,0]

def Heuristic(node):

    global values_0,values_1,values_2,values_3,values_4,values_5,values_6,values_7,values_8
    v0=values_0[node.index("0")]
    v1=values_1[node.index("1")]
    v2=values_2[node.index("2")]
    v3=values_3[node.index("3")]
    v4=values_4[node.index("4")]
    v5=values_5[node.index("5")]
    v6=values_6[node.index("6")]
    v7=values_7[node.index("7")]
    v8=values_8[node.index("8")]
    valorTotal = v0+v1+v2+v3+v4+v5+v6+v7+v8
    return valorTotal
```

```python
#Obtain Sub Nodes
def subNodes(node):

    global NodesExpanded
    NodesExpanded = NodesExpanded+1

    nextPaths = []
    nextPaths.append(PuzzleState(move(node.state, 1), node, 1, node.depth + 1, node.cost + 1, 0))
    nextPaths.append(PuzzleState(move(node.state, 2), node, 2, node.depth + 1, node.cost + 1, 0))
    nextPaths.append(PuzzleState(move(node.state, 3), node, 3, node.depth + 1, node.cost + 1, 0))
    nextPaths.append(PuzzleState(move(node.state, 4), node, 4, node.depth + 1, node.cost + 1, 0))
    nodes=[]
    for procPaths in nextPaths:
        if(procPaths.state!=None):
            nodes.append(procPaths)
    return nodes

#Next step
def move(state, direction):
    #generate a copy
    newState = state[:]

    #obtain poss of 0
    index = newState.index(0)

    if(index==0):
        if(direction==1):
            return None
        if(direction==2):
            temp=newState[0]
            newState[0]=newState[3]
            newState[3]=temp
        if(direction==3):
            return None
```

```python
#MAIN
def main():

    global GoalNode

    #Obtain information from calling parameters
    parser = argparse.ArgumentParser()
    parser.add_argument('initialBoard')
    args = parser.parse_args()
    data = args.initialBoard.split(",")

    #Build initial board state
    InitialState = []
    InitialState.append(int(data[0]))
    InitialState.append(int(data[1]))
    InitialState.append(int(data[2]))
    InitialState.append(int(data[3]))
    InitialState.append(int(data[4]))
    InitialState.append(int(data[5]))
    InitialState.append(int(data[6]))
    InitialState.append(int(data[7]))
    InitialState.append(int(data[8]))

    #Start operation
    start = timeit.default_timer()

    ast(InitialState)

    stop = timeit.default_timer()
    time = stop-start
```

```python
    #Save total path result
    deep=GoalNode.depth
    moves = []
    while InitialState != GoalNode.state:
        if GoalNode.move == 1:
            path = 'Up'
        if GoalNode.move == 2:
            path = 'Down'
        if GoalNode.move == 3:
            path = 'Left'
        if GoalNode.move == 4:
            path = 'Right'
        moves.insert(0, path)
        GoalNode = GoalNode.parent

    #Print results
    print("path: ",moves)
    print("cost: ",len(moves))
    print("nodes expanded: ",str(NodesExpanded))
    print("search_depth: ",str(deep))
    print("MaxSearchDeep: ",str(MaxSearchDeep))
    print("running_time: ",format(time, '.8f'))

if __name__ == '__main__':
    main()
```

```
PS C:\Farim\Kuliah\KB\8puzzle\8puzzle> python 8puzzle.py 1,2,3,0,5,6,4,7,8
path:  ['Up', 'Right', 'Down', 'Right', 'Up', 'Left', 'Left', 'Down', 'Right', 'Right', 'Down', 'Left', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up'
, 'Up', 'Left', 'Left']
cost:  21
nodes expanded:  1327
search_depth:  21
MaxSearchDeep:  21
running_time:  0.06265610
```

TERIMA KASIH