DOCUMENTATION REPORT

Programming Project 1
Solving the 8-puzzle using A* algorithm

ITCS 6150/8150 - Intelligent Systems

Submitted by

Bhanu Prakash Reddy Chennam
801100084

Satabdhi Reddy Nalamalpu
801115132

Table of Contents

PROBLEM STATEMENT:

To implement A* search algorithm and apply it to 8-puzzle problem using programming language

8-puzzle problem

A 3x3 grid contains 8 tiles numbered 1 to 8 and tile adjacent to the empty space can be slide to the empty space finally reaching the goal state

A* ALGORITHM:

A* method uses admissible heuristic to determine the next state.
The heuristic used to evaluate distances in A* is:

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

where g(n) represents the cost (distance) of the path from the starting state to current state, and h(n) represents the estimated cost to the goal.

h(n) Manhattan Distance

The distance between two points measured along axes at right angles. In a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2), it is $|x_1 - x_2| + |y_1 - y_2|$.

h(n) Misplaced Tiles

The number of misplaced tiles comparing current state and goal state

Initial state

1	2	3
4	0	6
7	5	8

Goal state

6	2	3
4	5	1
7	8	0

```
heuristic_manhattan_dist=3+0+0+0+3+0+1+1 = 8
tiles 1,6,5,8 take 3,3,1,1 steps respectively to
reach their goal state.

heuristic_misplaced_dist=1+0+0+0+1+0+1+1= 4
tiles 1,6,5,8 are misplaced.
```

CODE:
Programming language:Python3
python interpreter version:3.6
additional libraries needed:numpy

Data structure used:
each node is a structured array and stored in fringe (open) or closed list

node Data format [[list],g(n),h(n),f(n),nodeID,parentID]
example: [[1,2,3,4,5,6,7,8,0],1,4,5,2,1]

```
# ;==========================================================================
# ; Title:  A* Algorithm using manhattan distance and misplaced tiles
# ; Author: Bhanu Prakash Reddy ,Satabdhi Reddy
# ; Date:   10 Feb 2019

# ;==========================================================================
import numpy as np
from copy import deepcopy
import collections

print("enter input by giving spaces example:1 2 3 4 5 6 7 8 0 ")
initial_node = list(map(int,input("Enter Input node:").split()))              #input initial node
initial_node = np.array(initial_node)
final_node = list(map(int,input("Enter Output node:").split()))               #input final node
final_node = np.array(final_node)
Astarmethod=0      #flag for manhatton or misplaced tiles Astarmethod=0 manhattan 1 for misplaced
tiles


# definition to calculate hn using manhattan distance
def manhattan_distance(mlist):
    copy = mlist
    mhtndist = 0
    for i, list_item in enumerate(copy):
        if list_item != 0:
            for j,list_item_final in enumerate(final_node):
                if list_item_final == list_item:
                    lr = j
                    break
            row1,col1 = int(i/ 3) , i% 3
            row2,col2 = int((lr) / 3), (lr) % 3
            mhtndist += abs(row1-row2) + abs(col1 - col2)
    return mhtndist

#definition to calculate hn using misplaced tiles
def misplaced_tiles(mlist):
    copy = mlist
    mis_dist=0
    for i, list_item in enumerate(copy):
        if list_item != 0 and final_node[i] != list_item:
            mis_dist = mis_dist+1
    return  mis_dist

#definition to calculate successor nodes of the node to be expanded
def successorNodes(board):
    global open_struct_array
    global closed_struct_array
```

```
    global nodeid
    moves = np.array(
        [
            ([0, 1, 2], -3),
            ([6, 7, 8], 3),
            ([0, 3, 6], -1),
            ([2, 5, 8], 1)
        ],
        dtype=[
            ('pos', list),
            ('ind', int)
        ]
    )

    gn=board[1]+1
    state = board[0]
    loc = int(np.where(state == 0)[0])
    parentid=board[4]
    for m in moves:
        if loc not in m['pos']:
            nodepresent = 0
            succ = deepcopy(state)
            delta_loc = loc + m['ind']
            succ[loc], succ[delta_loc] = succ[delta_loc], succ[loc]

            for i in closed_struct_array:            #checking if successor nodes are duplicates
                if(i[0]==succ).all():
                    nodepresent = 1

            for i in open_struct_array:              #checking if successor nodes are duplicates
                if (i[0] == succ).all():
                    nodepresent = 1

            if nodepresent == 0:
                #print("inloop")

                if (Astarmethod == 0):
                    hn = manhattan_distance(succ)
                else:
                    hn = misplaced_tiles(succ)
                fn=gn + hn
                nodeid=nodeid+1                          #increment value of nodeid for each node genereated
                #appending successor nodes to open_struct_array
                open_struct_array=np.append(open_struct_array, np.array([(succ, gn, hn, fn, nodeid,
parentid)], STATE), 0)

#definition to check if the node is final node
def solution(board):
    global STATE
```

```
    STATE = [
        ('board', list),
        ('gn', int),
        ('hn', int),
        ('fn', int),
        ('nodeid',int),
        ('parentid',int)
    ]
    global open_struct_array
    global closed_struct_array
    global nodeid
    nodeid = 0
    if(Astarmethod==0):
        hn=manhattan_distance(board)
    else:
        hn=misplaced_tiles(board)
    open_struct_array = np.array([(board, 0, hn, 0 + hn, 0, -1)], STATE)

    varran=np.array([0,0,0,0,0,0,0,0,0])                    #closed struct array 1 time initialization
    closed_struct_array=np.array([(varran, 0, 0, 0, 0, 0)], STATE)
    closed_struct_array=np.delete(closed_struct_array, 0, 0)

    while True:
        length_queques = len(open_struct_array) + len(closed_struct_array)    #checking if total nodes are
crossing the threshold value
        if length_queques >3000:
            break
        a=open_struct_array[0]
        s=a[0]
        if (s == final_node).all():             #comparing with final node
            return len(closed_struct_array), nodeid
        open_struct_array = np.delete(open_struct_array, 0, 0)
        closed_struct_array=np.append(closed_struct_array, np.array([(a[0], a[1], a[2], a[3], a[4], a[5])],
STATE), 0) #appending expanded node to closed node
        successorNodes(a)
        open_struct_array = np.sort(open_struct_array, kind='mergesort', order=['fn', 'nodeid'])  #sort
based on value of fn
    return 0,0

#definition to find the path of the final node
def solutionpath(open_structured_array, closedNode):
    storelastelement = open_structured_array[0][0]
    parentidd=open_structured_array[0][5]
    con = np.concatenate((open_structured_array, closedNode), axis=0)
    de = collections.deque([])
    de.append(storelastelement)
    while(parentidd != -1):
        for i in con:
            if i[4] == parentidd:
```

```
            de.appendleft(i[0])
            parentidd = i[5]
            break
    print('cost to reach final_node:',len(de)-1)

    for i in de:
        print(np.reshape(i,(3,3)),'\n')

#definintion to print output using both manhattan distance and misplaced tiles as hn
def main():
    global open_struct_array
    global closed_struct_array
    global Astarmethod


    comparearrays = (np.sort(initial_node) == np.sort(final_node)).all()     #checking if input is correct
    if not comparearrays:
        print('incorrect input')
        return
    else:
        nodes_expanded,nodes_generated=solution(initial_node)              #if correct input find path
        if(nodes_expanded==0 and nodes_generated ==0):
            print('no solution')
            return
    print("-------------------------------A* Manhattan DIstance------------------------------------------")
    # print(open_struct_array)
    # print(closed_struct_array)
    print('nodes_generated:',nodes_generated)
    print('nodes_expanded',nodes_expanded)

    solutionpath(open_struct_array, closed_struct_array)        #finding solution path hn= manhattan
distance
    print("-------------------------------A* Misplaced Tiles-----------------------------------------------")
    Astarmethod=1                                           #set Astarmethod=1
    open_struct_array=[]                                     #empty both open and closed
    closed_struct_array=[]
    nodes_expanded, nodes_generated = solution(initial_node)
    if (nodes_expanded == 0 and nodes_generated == 0):          #return 0,0 if no solution
        print('no solution')
        return
    # print(open_struct_array)
    # print(closedNode)
    print('nodes_generated:',nodes_generated)
    print('nodes_expanded',nodes_expanded)
    solutionpath(open_struct_array, closed_struct_array)         #finding solution path hn=misplaced tiles


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

Global variables used:

1. Nodeid ,each node has a unique id
2. open_struct_array ,storing expanded nodes
3. closed_struct_Array ,storing closed nodes

Functions used:

1. main()
2. solution()
3. solutionpath()
4. successorNodes()
5. manhattan_distance()
6. misplaced_tiles()

main function

- check for correct inputs from user
- sending initial_node to solution function
- toggle flag for finding path using misplaced tiles
- printing generated nodes,nodes expanded

solution function

- Runs a infinite while loop which breaks on finding final path or if length of fringe + closed exceeds a threshold.
- Delete node expanded from open and add it in closed
- sending the node to be expanded to successor node function
- sorting the fringe(open) based on f(n)

successornodes function

- generating successor nodes and adding it to open
- checking if successor nodes are duplicates

manhattan_distance function

- calculating manhattan distance for a node

misplaced_tiles function

- calculating number of misplaced tiles

solutionpath function

- backtracking the solution path using the parent node of found solution node

SAMPLE INPUT OUTPUT:

Example1:

```
enter input by giving spaces example:1 2 3 4 5 6 7 8 0
Enter Input node:1 0 8 2 5 7 4 6 3
Enter Output node:0 8 7 1 2 3 4 5 6
--------------------------------A* Manhattan DIstance-------------------------------------------
nodes_generated: 14
nodes_expanded 7
cost to reach final_node: 7
[[1 0 8]
 [2 5 7]
 [4 6 3]]

[[1 8 0]
 [2 5 7]
 [4 6 3]]

[[1 8 7]
 [2 5 0]
 [4 6 3]]

[[1 8 7]
 [2 5 3]
 [4 6 0]]

[[1 8 7]
 [2 5 3]
 [4 0 6]]

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

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

[[0 8 7]
 [1 2 3]
 [4 5 6]]

-------------------------------A* Misplaced Tiles------------------------------------------------
nodes_generated: 14
nodes_expanded 7
cost to reach final_node: 7

[[1 0 8]
 [2 5 7]
 [4 6 3]]

[[1 8 0]
 [2 5 7]
 [4 6 3]]

[[1 8 7]
 [2 5 0]
 [4 6 3]]

[[1 8 7]
 [2 5 3]
 [4 6 0]]

[[1 8 7]
 [2 5 3]
 [4 0 6]]

[[1 8 7]
 [2 0 3]
 [4 5 6]]

[[1 8 7]
 [0 2 3]
 [4 5 6]]

[[0 8 7]
 [1 2 3]
 [4 5 6]]


Process finished with exit code 0
```

Example 2 :

```
enter input by giving spaces example:1 2 3 4 5 6 7 8 0
Enter Input node:1 2 7 6 0 8 4 3 5
Enter Output node:1 2 0 4 8 7 3 6 5
-------------------------------A* Manhattan DIstance------------------------------------------
nodes_generated: 17
nodes_expanded 8
cost to reach final_node: 6
[[1 2 7]
 [6 0 8]
 [4 3 5]]

[[1 2 7]
 [0 6 8]
 [4 3 5]]

[[1 2 7]
 [4 6 8]
 [0 3 5]]

[[1 2 7]
 [4 6 8]
 [3 0 5]]

[[1 2 7]
 [4 0 8]
 [3 6 5]]

[[1 2 7]
 [4 8 0]
 [3 6 5]]

[[1 2 0]
 [4 8 7]
 [3 6 5]]


-------------------------------A* Misplaced Tiles-----------------------------------------------
nodes_generated: 19
nodes_expanded 9
cost to reach final_node: 6
[[1 2 7]
 [6 0 8]
 [4 3 5]]
```

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

[[1 2 7]
 [4 6 8]
 [0 3 5]]

[[1 2 7]
 [4 6 8]
 [3 0 5]]

[[1 2 7]
 [4 0 8]
 [3 6 5]]

[[1 2 7]
 [4 8 0]
 [3 6 5]]

[[1 2 0]
 [4 8 7]
 [3 6 5]]


Process finished with exit code 0
```

Example 3:

```
enter input by giving spaces example:1 2 3 4 5 6 7 8 0
Enter Input node:6 3 5 8 7 0 2 1 4
Enter Output node:6 5 7 8 3 4 2 1 0
--------------------------------A* Manhattan DIstance-------------------------------------------
nodes_generated: 12
nodes_expanded 6
cost to reach final_node: 5
[[6 3 5]
 [8 7 0]
 [2 1 4]]

[[6 3 5]
 [8 0 7]
 [2 1 4]]

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

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

[[6 5 7]
 [8 3 0]
 [2 1 4]]

[[6 5 7]
 [8 3 4]
 [2 1 0]]


-------------------------------A* Misplaced Tiles-------------------------------------------------
nodes_generated: 13
nodes_expanded 7
cost to reach final_node: 5
[[6 3 5]
 [8 7 0]
 [2 1 4]]

[[6 3 5]
 [8 0 7]
 [2 1 4]]

[[6 0 5]
 [8 3 7]
 [2 1 4]]

[[6 5 0]
 [8 3 7]
 [2 1 4]]

[[6 5 7]
 [8 3 0]
 [2 1 4]]

[[6 5 7]
 [8 3 4]
 [2 1 0]]


Process finished with exit code 0
```

Example 4:

```
enter input by giving spaces example:1 2 3 4 5 6 7 8 0
Enter Input node:2 1 8 3 0 4 6 7 5
```

```
Enter Output node:2 7 1 3 0 8 6 5 4
-------------------------------A* Manhattan DIstance-----------------------------------------
nodes_generated: 12
nodes_expanded 6
cost to reach final_node: 6
[[2 1 8]
 [3 0 4]
 [6 7 5]]

[[2 1 8]
 [3 7 4]
 [6 0 5]]

[[2 1 8]
 [3 7 4]
 [6 5 0]]

[[2 1 8]
 [3 7 0]
 [6 5 4]]

[[2 1 0]
 [3 7 8]
 [6 5 4]]

[[2 0 1]
 [3 7 8]
 [6 5 4]]

[[2 7 1]
 [3 0 8]
 [6 5 4]]

-------------------------------A* Misplaced Tiles---------------------------------------------
nodes_generated: 19
nodes_expanded 10
cost to reach final_node: 6
[[2 1 8]
 [3 0 4]
 [6 7 5]]

[[2 1 8]
 [3 7 4]
 [6 0 5]]

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

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

[[2 1 0]
 [3 7 8]
 [6 5 4]]

[[2 0 1]
 [3 7 8]
 [6 5 4]]

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

OBSERVATIONS:

Heuristic misplaced tiles only considers whether a tile is misplaced or not unlike manhattan distance which takes into consideration the distance of the misplaced tile. We see through experimental results that number of nodes generated are more when misplaced tiles heuristic is used

Example - 1

Initial Node

1 0 8
2 5 7
4 6 3

Final Node

0 8 7
1 2 3
4 5 6

① Manhattan Distance

⓪
1 0 8
2 5 7
4 6 3

①
1 5 8
2 0 7
4 6 3

$f(n) = g(n) + h(n)$
$= 1 + 8 = 9$

②
0 1 8
2 5 7
4 6 3

$f(n) = g(n) + h(n)$
$= 1 + 8 = 9$

③
1 8 0
2 5 7
4 6 3

$f(n) = g(n) + h(n)$
$= 1 + 6 = 7$

④
1 8 7
2 5 0
4 6 3

$f(n) = g(n) + h(n)$
$= 2 + 5 = 7$

⑥
1 8 7
2 0 5
4 6 3

$f(n) = g(n) + h(n)$
$= 3 + 6 = 9$

⑤
1 8 7
2 5 3
4 6 0

$f(n) = g(n) + h(n)$
$= 3 + 4 = 7$

⑦
1 8 7
2 5 3
4 0 6

$f(n) = g(n) + h(n)$
$= 4 + 3 = 7.$

⑨
1 8 7
2 5 3
0 4 6

$f(n) = g(n) + h(n)$
$= 5 + 4$
$= 9.$

⑧
1 8 7
2 0 3
4 5 6

$f(n) = g(n) + h(n)$
$= 5 + 2 = 7.$

⑩
1 0 7
2 8 3
4 5 6

$f(n) = g(n) + h(n)$
$= 6 + 3 = 9$

⑪
1 8 7
0 2 3
4 5 6

$f(n) = g(n) + h(n)$
$= 6 + 1 = 7.$

⑫
1 8 7
2 3 0
4 5 6

$f(n) = g(n) + h(n)$
$= 6 + 3 = 9.$

⑬
0 8 7
1 2 3
4 5 6

$f(n) = g(n) + h(n)$
$= 7 + 0$

⑭
1 8 7
4 2 3
0 5 6

$f(n) = g(n) + h(n)$
$= 7 + 2 = 9$

Solution Path:

$0 \to 3 \to 4 \to 5 \to 7 \to 8$
$13 \leftarrow 11 \leftarrow$

Number of nodes generated: 14

Number of nodes expanded: 7

Example - 1

Initial State

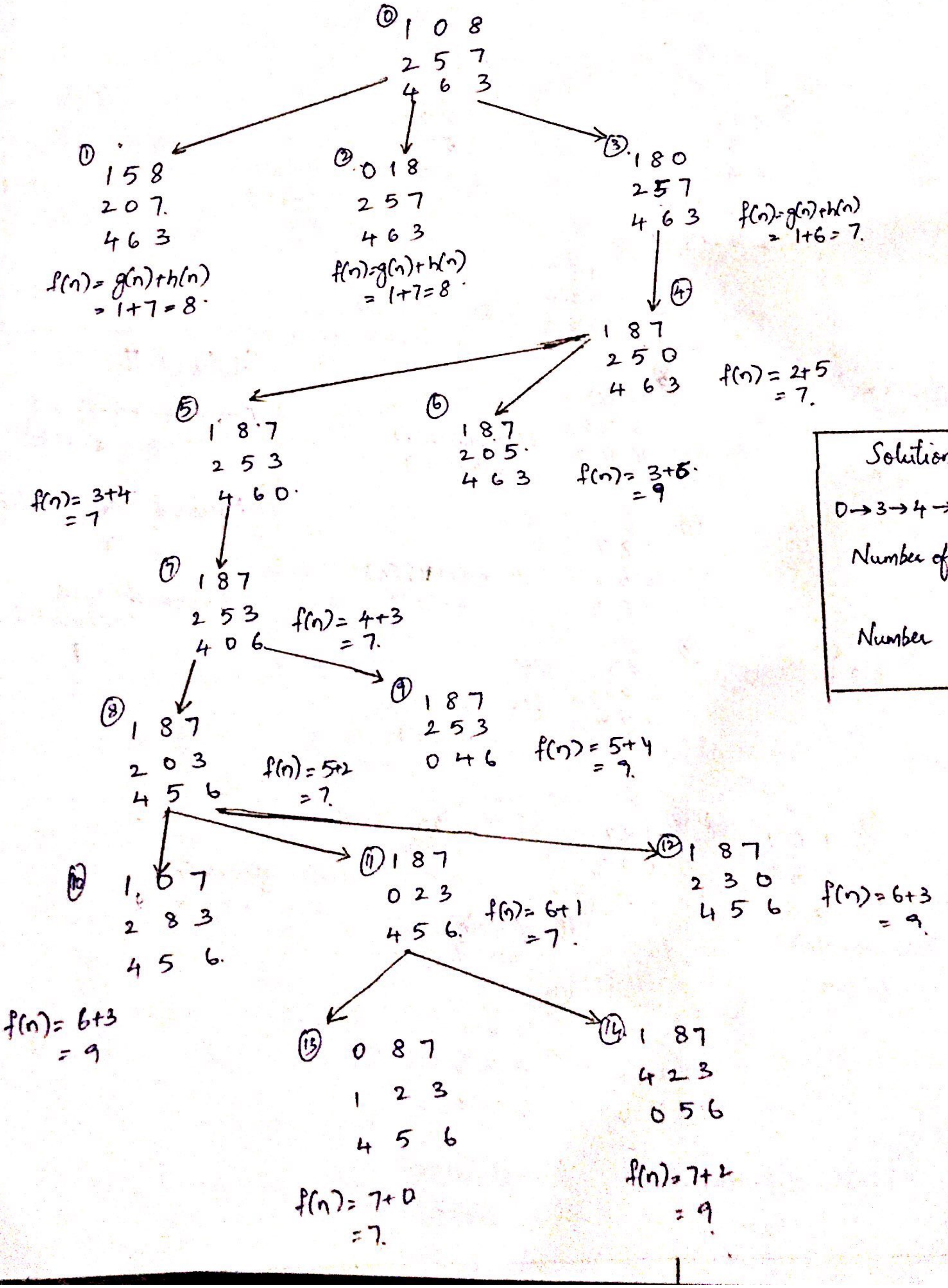
1 0 8
2 5 7
4 6 3

Final State:

0 8 7
1 2 3
4 5 6

2. Misplaced Tiles:

State Space Representation



Solution Path:

0 → 3 → 4 → 5 → 7 → 8 → 11 → 13

Number of nodes generated:
14

Number of nodes expanded:
7

Example - 2

Initial State

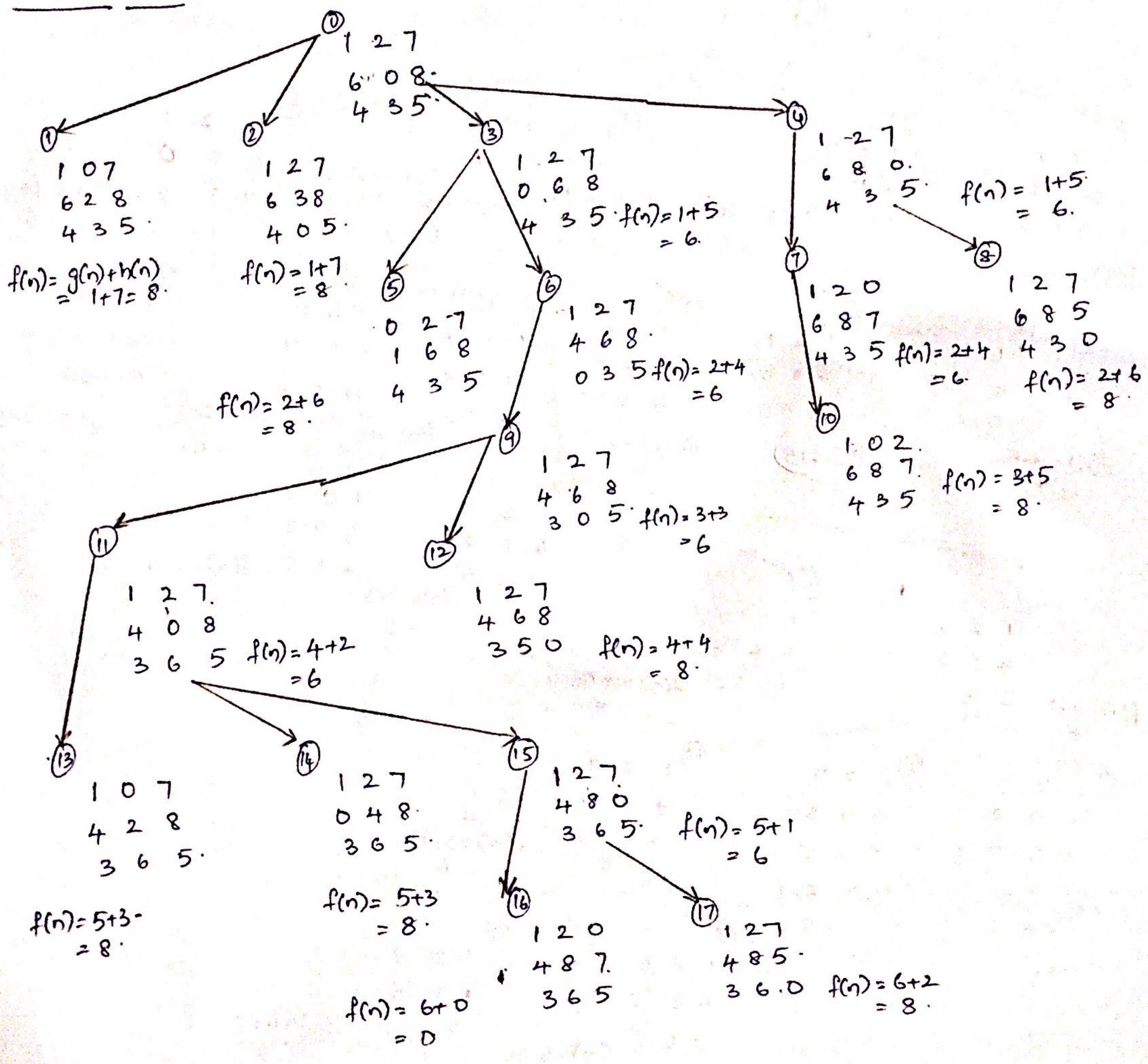
1 2 7
6 0 8
4 3 5

Final State

1 2 0
4 8 7
3 6 5

State Space Representation

1 Manhattan Distance:



Solution Path: 0 → 3 → 6 → 9 → 11 → 15 → 16

Number of nodes generated: 17.

Number of nodes expanded: 8

Example-2

Initial State:

1 2 7
6 0 8
4 3 5

Final State:

1 2 0
4 8 7
3 6 5

2. Misplaced Tiles:

State Space Representation

(0)
1 2 7
6 0 8
4 3 5

(1)
1 0 7
6 2 8
4 3 5
f(n) = 1+6 = 7

(2)
1 2 7
6 3 8
4 0 5
f(n) = 1+5 = 6

(3)
1 2 7
0 6 8
4 3 5
f(n) = 1+5 = 6

(4)
1 2 7
6 8 0
4 3 5
f(n) = 1+4 = 5

(5)
1 2 0
6 8 7
4 3 5
f(n) = 2+3 = 5

(6)
1 2 7
6 8 5
4 3 0
f(n) = 2+5 = 7

(7)
1 0 2
6 8 7
4 3 5
f(n) = 3+4 = 7

(8)
1 2 7
6 3 8
0 4 5
f(n) = 2+5 = 7

(9)
1 2 7
6 3 8
4 5 0
f(n) = 2+5 = 7

(10)
0 2 7
1 6 8
4 3 5
f(n) = 2+6 = 8

(11)
1 2 7
4 6 8
0 3 5
f(n) = 2+4 = 6

(12)
1 2 7
4 6 8
3 0 5
f(n) = 3+3 = 6

(13)
1 2 7
4 0 8
3 6 5
f(n) = 4+2 = 6

(14)
1 2 7
4 6 8
3 5 0
f(n) = 4+4 = 8

(15)
1 0 7
4 2 8
3 6 5
f(n) = 5+3 = 8

(16)
1 2 7
0 4 8
3 6 5
f(n) = 5+3 = 8

(17)
1 2 7
4 8 0
3 6 5
f(n) = 5+1 = 6

(18)
1 2 0
4 8 7
3 6 5
f(n) = 6+0 = 6

(19)
1 2 7
4 8 5
3 6 0
f(n) = 6+2 = 8

Solution Path: 0 → 3 → 11 → 12 → 13 → 17 → 18.

Number of nodes generated: 19.

Number of nodes explored: 9.

Example-3

Initial State:

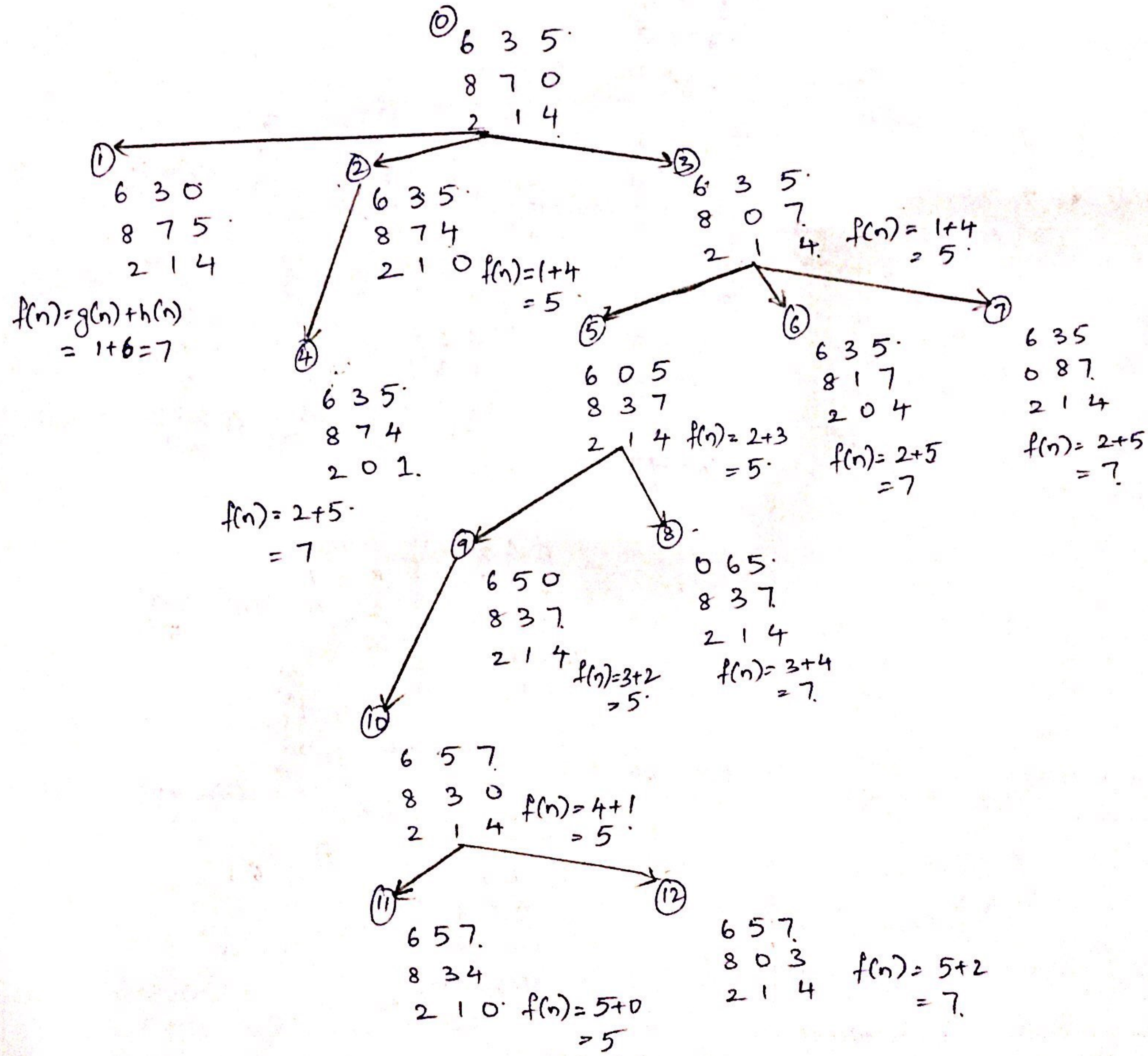
$$\begin{matrix} 6 & 3 & 5 \\ 8 & 7 & 0 \\ 2 & 1 & 4 \end{matrix}$$

Final State:

$$\begin{matrix} 6 & 5 & 7 \\ 8 & 3 & 4 \\ 2 & 1 & 0 \end{matrix}$$

1.) Manhattan Distance:

State Space Representation:



Solution Path: $0 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10 \rightarrow 11$

Number of nodes generated: 12

Number of nodes expanded: 6

Example -3

Initial State:

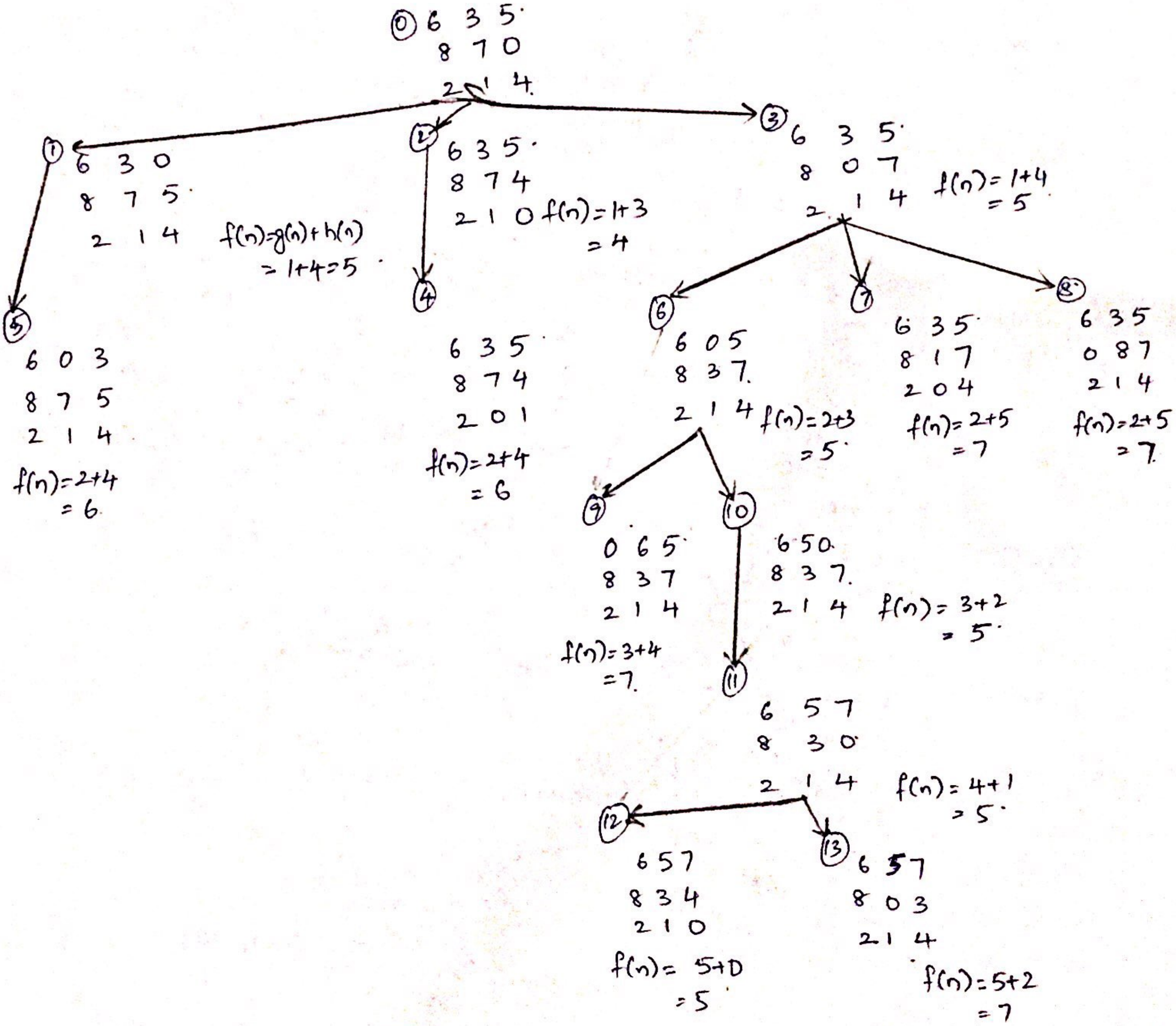
6 3 5
8 7 0
2 1 4

Final State:

6 5 7
8 3 4
2 1 0

② Misplaced Tiles:

State Space Representation:



Path of solution: 0 → 3 → 6 → 10 → 11 → 12

Number of nodes generated: 13

Number of nodes explored: 7

Example-4

Initial State:

2 1 8
3 0 4
6 7 5

Final State

2 7 1
3 0 8
6 5 4

State Space Representation

① **Manhattan Distance:**

(0)
2 1 8
3 0 4
6 7 5

(1)
2 0 8
3 1 4
6 7 5

$f(n) = g(n) + h(n)$
$= 1 + 7 = 8$

(2)
2 1 8
3 7 4
6 0 5

$f(n) = 1 + 5$
$= 6$

(3)
2 1 8
0 3 4
6 7 5

$f(n) = 1 + 7$
$= 8$

(4)
2 1 8
3 4 0
6 7 5

$f(n) = 1 + 7$
$= 8$

(5)
2 1 8
3 7 4
0 6 5

$f(n) = 2 + 6$
$= 8$

(6)
2 1 8
3 7 4
6 5 0

$f(n) = 2 + 4$
$= 6$

(7)
2 1 8
3 7 0
6 5 4

$f(n) = 3 + 3$

(8)
2 1 0
3 7 8
6 5 4

$f(n) = 4 + 2$
$= 6$

(9)
2 1 8
3 0 7
6 5 4

$f(n) = 4 + 4$
$= 8$

(10)
2 0 1
3 7 8
6 5 4

$f(n) = 5 + 1$

(11)
2 7 1
3 0 8
6 5 4

$f(n) = 6 + 0$

(12)
0 2 1
3 7 8
6 5 4

$f(n) = 6 + 2$
$= 8$

Solution Path: $0 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 10 \rightarrow 11$.

Number of nodes generated: 12.

Number of nodes explored: 6

Example-4

Initial State:

2 1 8
3 0 4
6 7 5

Final State:

2 7 1
3 0 8
6 5 4.

② Misplaced Tiles:

State Space Representation

⓪
2 1 8
3 0 4
6 7 5

①
2 0 8
3 1 4
6 7 5
f(n) = 1+5 = 6

②
2 1 8
3 7 4
6 0 5
f(n) = 1+5 = 6

③
2 1 8
0 3 4
6 7 5
f(n) = 1+6 = 7

④
2 1 8
3 4 0
6 7 5
f(n) = 1+5 = 6

⑤
0 2 8
3 1 4
6 7 5
f(n) = 2+6 = 8

⑥
2 8 0
3 1 4
6 7 5
f(n) = 2+5 = 7.

⑦
2 1 8
3 7 4
0 6 5
f(n) = 2+6 = 8

⑧
2 1 8
3 7 4
6 5 0
f(n) = 2+4 = 6

⑨
2 1 0
3 4 8
6 7 5
f(n) = 2+4 = 6

⑩
2 1 8
3 4 5
6 7 0
f(n) = 2+5 = 7

⑪
2 1 8
3 7 0
6 5 4
f(n) = 3+3 = 6

⑫
2 0 1
3 4 8
6 7 5
f(n) = 3+3 = 6

⑬
2 1 0
3 7 8
6 5 4
f(n) = 4+2 = 6

⑭
2 1 8
3 0 7
6 5 4.
f(n) = 4+3 = 7

⑮
2 4 1
3 0 8
6 7 5
f(n) = 4+3 = 7

⑯
0 2 1
3 4 8
6 7 5
f(n) = 4+4 = 8.

⑰
2 0 1
3 7 8
6 5 4.
f(n) = 5+1 = 6.

⑱
2 7 1
3 0 8
6 5 4
f(n) = 6+0 = 6

⑲
0 2 1
3 7 8
6 5 4.
f(n) = 6+2 = 8.

Solution Path:

0→2→8→11→13→17→18.

Number of nodes generated: 19

Number of nodes expanded: 10