Solving 8 – Puzzle using A* Search Algorithm

PROJECT DOCUMENTATION REPORT

PROGRAMMING PROJECT 1

ITCS 6150 – Intelligent Systems

Submitted to

Prof. Dewan T. Ahmed

Submitted by

Shimpli Kalse
801202853

Devyani Barde
801208619

Table of Contents

1. PROBLEM FORMULATION

1.1 Introduction

8 Puzzle Problem

The 8 puzzle problem is a puzzle invented by Noyes Palmer Chapman. It is played on a 3-by-3 grid with 8 square blocks labeled from 1 to 8 and a blank square. The goal is to rearrange the blocks so that they are in order. We are permitted to slide the blocks horizontally or vertically into the blank space. The following shows the sequence of legal moves from initial to goal position.

A* Search Algorithm

A* search is an algorithm that searches for the shortest path from the initial to goal node. A* search combines two evaluation functions from Greedy search algorithm and uniform cost search algorithm. h(n) is the minimized estimated cost to the goal used by greedy search and g(n) is the minimized cost of path used in uniform cost search. A* search combines the two evaluation functions simply by summing them:

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

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have

$$f(n) = \text{ estimated cost of the cheapest solution through n}$$

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of f. [1] The explanation of three functions used in A* search:

- h(n): also known as the heuristic value, it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost. [2]
- g(n): the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell. [2]
- f(n): it is the sum of g(n) and h(n). So, f(n) = g(n) + h(n) [2]

Different heuristics can be used to find solution of one particular problem. In this project we will be solving the 8 puzzle problem with A* search algorithm. We will be using two heuristics namely Manhattan Distance and Misplaced Tiles.

2. PROGRAM STRUCTURE

2.1 Global Variables

Variable name	Variable type
Initial_state	Integer list (user input for initial state from [0-8])
goal	Integer list (user input for goal state from [0-8])
bestpath	Integer list of state space after every optimal move
expanded	Integer value containing total number of expanded nodes.
tot_steps	Integer value that holds the total number of moves made to reach the solution

2.2 Functions and Procedures

Function/procedure	Description
inputo()	This function takes input for initial_state ranging from 0-8 where 0 is treated as blank tile. If the user put any number out of range (0-8), it gives out the error and asks user to put the input again
goalo()	This function takes input for desired goal state ranging from 0-8 where 0 is treated as blank tile. If the user put any number out of range (0-8), it gives out the error and asks user to put the input again
Best_path(state)	This function takes all the states as a and traverses through each of the states to find the best solution and returns the best optimal solution.
mh_dis(initial_state, goal)	This function computes the mahanttan distance of each element in the initial state by calculating the optimal moves each element should travel to reach the desired goal state
all ()	This function checks whether the current state is a duplicate of some other state that has been traversed in the past and returns 1 if it is and 0 if it is not using Python's set form.
misplaced()	This function takes the current and target states as inputs and returns the number of tiles that have been lost. If no tiles are misplaced, the result is nil.

Lable(parameter)	This function returns the single integer value of the coordinates after assigning integer values as coordinates to any input passed as a parameter.
eval(Initial_state, goal)	The manhattan heuristic is used to solve the 8-puzzle problem in this function. G(n), h(n), and f(n) are determined in this function, and a priority queue is used to store the node fn and place value as a key-value pair. If a node hasn't been explored yet, we use the merge sort technique to find the lowest cost node and explore it first. To search for duplicates, we use the all() function. To compile, we use Mh_dis(). After identifying the blank tile, we proceed to the next available steps. Then, create new nodes and decide whether or not this new node is the target state. h(n) calculates the amount of the number of steps each tile in the current state would take to achieve the target state using the formula f(n)= g(n)+h(n). The step cost of each step is g(n).
eval_misplaced	Using the Misplaced Tiles heuristic, this function solves the 8-puzzle problem. G(n), h(n), and f(n) are determined in this function, and a priority queue is used to store the node fn and place value as a key-value pair. If a node hasn't been explored yet, we use the merge sort technique to find the lowest cost node and explore it first. To search for duplicates, we use the all() function. To calculate the cost of misplaced tiles, we use the misplaced tiles() function. We find the blank tile, then look for the next available steps, create new nodes, and decide whether or not this new node is the target state. In this case, f(n)= g(n)+h(n) h(n) calculates the number of misplaced tiles in the current state versus the target state. The path cost for each move is g(n).
Deepcopy()	A copy of object is copied in another object. It means that any changes made to a copy of object do not reflect in the original object.

2.3 Source Code

```
"""
Created on Wed Mar 10 21:53:45 2021
@author: Shimpli Kalse
@author: Devyani Barde
"""
import numpy as np
import time
from copy import deepcopy
# Taking input from user as an initial state and handling error for invalid input
def inputo():
    global initial_state
    initial_state = []
    print(" Input values from range 0-8 for desired goal state(0 is the blank tile)")
    for i in range(0,9):
        x = int(input("Enter values for tiles :"))
        initial_state.append(x)
    for ele in initial_state:
        if ele < 0 or ele > 8:
            print("Invalid Input")
            inputo()
# Taking input from user as desired goal state and handling error for invalid input
def goalo():
    global goal
    goal = []
    print(" Input values from range 0-8 for desired goal state(0 is the blank tile)")
    for i in range(0,9):
        x = int(input("Enter values for tiles :"))
        goal.append(x)
```

```
    for ele in goal:
        if ele < 0 or ele > 8:
            print("Invalid Input")
            goalo()
# function to take the input of initial states and analyzing the best path to goal state
def best_path(state):
    bestsolution = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsolution = np.insert(bestsolution, 0, state[count]['initial_state'], 0)
        count = (state[count]['parent_node'])
    return bestsolution.reshape(-1, 3, 3)
# function to calculate Manhattan distance cost between inital tile(input) and final goal
def mh_dis(initial_state, goal):
    u = abs(initial_state // 3 - goal // 3)
    v = abs(initial_state % 3 - goal % 3)
    mhcost = u + v
    return sum(mhcost[1:])
# funtion to check the distinctiveness of the iteration(i) state, to check if it was previously traversed
def all(checkarray):
    set=[]
    for i in set:
        for checkarray in i:
            return 1
        else:
            return 0
# function to compute the number of misplaced tiles between the given state and goal state
def misplaced(initial_state,goal):
```

```
    mscost = np.sum(initial_state != goal) - 1
    return mscost if mscost > 0 else 0
# function to determine the coordinates of every state(initial or goal)
def coordinates(initial_state):
    pos = np.array(range(9))
    for w, s in enumerate(initial_state):
        pos[s] = w
    return pos
# code for 8 puzzle evaluation using Manhattan heuristics
def eval(initial_state, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8],  3),('left', [0, 3, 6], -1),('right', [2,
5, 8],  1)],
            dtype =  [('move',  str, 1),('position', list),('head', int)])
    state1 = [('initial_state',  list),('parent_node', int),('gn',  int),('hn',  int)]
 # initializing the variables :parent_node, gn and hn where hn is manhattan distance
function call
    costg = coordinates(goal)
    parent_node = -1
    gn = 0
    hn = mh_dis(coordinates(initial_state), costg)
    state = np.array([(initial_state, parent_node, gn, hn)], state1)
# using priority queues where position treated as keys and fn as value.
    priorityq = [('position', int),('fn', int)]
    priority = np.array( [(0, hn)], priorityq)
    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
        position, fn = priority[0]
        priority = np.delete(priority, 0, 0)
        # sorting priority queue using merge sort
        initial_state, parent_node, gn, hn = state[position]
```

```
        initial_state = np.array(initial_state)
        # code to spot the blank square in input
        blank = int(np.where(initial_state == 0)[0])
        gn = gn + 1
        d = 1
        start_time = time.time()
        for s in steps:
           d = d + 1
           if blank not in s['position']:
              # generate new state as copy of current
              openstates = deepcopy(initial_state)
              openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']],
openstates[blank]
              # The distinct function is called to check previous traversal
              if ~(np.all(list(state['initial_state']) == openstates, 1)).any():
                 end_time = time.time()
                 if (( end_time - start_time ) > 1):
                    print(" 8 puzzle can not be solved for given set of input  ! \n")
                    exit
                 # calls the mh_dis function to calcuate the manhattan distance
                 hn = mh_dis(coordinates(openstates), costg)
                  # generate and add new state in the list
                 q = np.array([(openstates, position, gn, hn)], state1)
                 state = np.append(state, q, 0)
              # f(n): additon of the cost to reach the node plus cost to reach goal from
current node
                 fn = gn + hn
                 q = np.array([(len(state) - 1, fn)], priorityq)
                 priority = np.append(priority, q, 0)
```

```
                # Checking if the node in openstates are matching the goal state given by user.
                if np.array_equal(openstates, goal):
                    print('  8 puzzle can be solved for given set of input ! \n')
                    return state, len(priority)
    return state, len(priority)
# code for 8 puzzle evaluation using misplaced tiles in initial_state
def eval_misplaced(initial_state, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8],  3),('left', [0, 3, 6], -1),('right', [2, 5, 8],  1)],
            dtype =  [('move',  str, 1),('position', list),('head', int)])
    state1 = [('initial_state',  list),('parent_node', int),('gn',  int),('hn',  int)]
    costg = coordinates(goal)
    # initializing the parent node, gn and hn, where hn is misplaced function call
    parent_node = -1
    gn = 0
    hn = misplaced(coordinates(initial_state), costg)
    state = np.array([(initial_state, parent_node, gn, hn)], state1)
   #using priority queue where position treated as keys and fn as value.
    priorityq = [('position', int),('fn', int)]
    priority = np.array([(0, hn)], priorityq)

    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
        position, fn = priority[0]
        # sort priority queue using merge sort,the first element is picked for exploring.
        priority = np.delete(priority, 0, 0)
        initial_state, parent_node, gn, hn = state[position]
        initial_state = np.array(initial_state)
         # Identify the blank square in input
```

```
        blank = int(np.where(initial_state == 0)[0])
        # Increase cost g(n) by 1
        gn = gn + 1
        c = 1
        start_time = time.time()
        for s in steps:
          c = c + 1
          if blank not in s['position']:
             # generate new state as copy of current
            openstates = deepcopy(initial_state)
            openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']],
openstates[blank]
            # The check function is called, if the node has been previously explored or not.
            if ~(np.all(list(state['initial_state']) == openstates, 1)).any():
              end_time = time.time()
              if (( end_time - start_time ) > 1):
                print(" 8 puzzle can NOT be solved for given set of input ! \n")
                break
              # calls the Misplaced function to compute the cost
              hn = misplaced(coordinates(openstates), costg)
              # generate and add new state in the list
              q = np.array([(openstates, position, gn, hn)], state1)
              state = np.append(state, q, 0)
              # f(n) is the sum of cost to reach node and the cost to reach fromt he node
to the goal state
              fn = gn + hn
              q = np.array([(len(state) - 1, fn)], priorityq)
              priority = np.append(priority, q, 0)
              # Checking if the node in openstates are matching the goal state.
              if np.array_equal(openstates, goal):
```

```
                print('8 puzzle can be solved for given set of input ! \n')
                return state, len(priority)
    return state, len(priority)
# ----------Taking inputs and calling defined functions as per choice provided ----------
# function call to ask for input
inputo()
goalo()
n = int(input("1. Manhattan distance \n2. Misplaced tiles \n"))
if(n!=1 and n!=2):
# handling wrong input from user
  print('please choose correct input, either 1 or 2')
  n = int(input("1. Manhattan distance \n2. Misplaced tiles \n"))
if(n ==1 ):
    state, visited = eval(initial_state, goal)
    bestpath = best_path(state)
    print(str(bestpath).replace('[', ' ').replace(']', ''))
    tot_steps = len(bestpath) - 1
    print('Steps to reach goal:',tot_steps)
    expanded = len(state) - visited
    print('Total nodes expanded: ',expanded, "\n")
    print('Total generated:', len(state))

if(n == 2):
    state, visited = eval_misplaced(initial_state, goal)
    bestpath = best_path(state)
    print(str(bestpath).replace('[', ' ').replace(']', ''))
    tot_steps = len(bestpath) - 1
    print('Steps to reach goal:',tot_steps)
    expanded = len(state) - visited
```

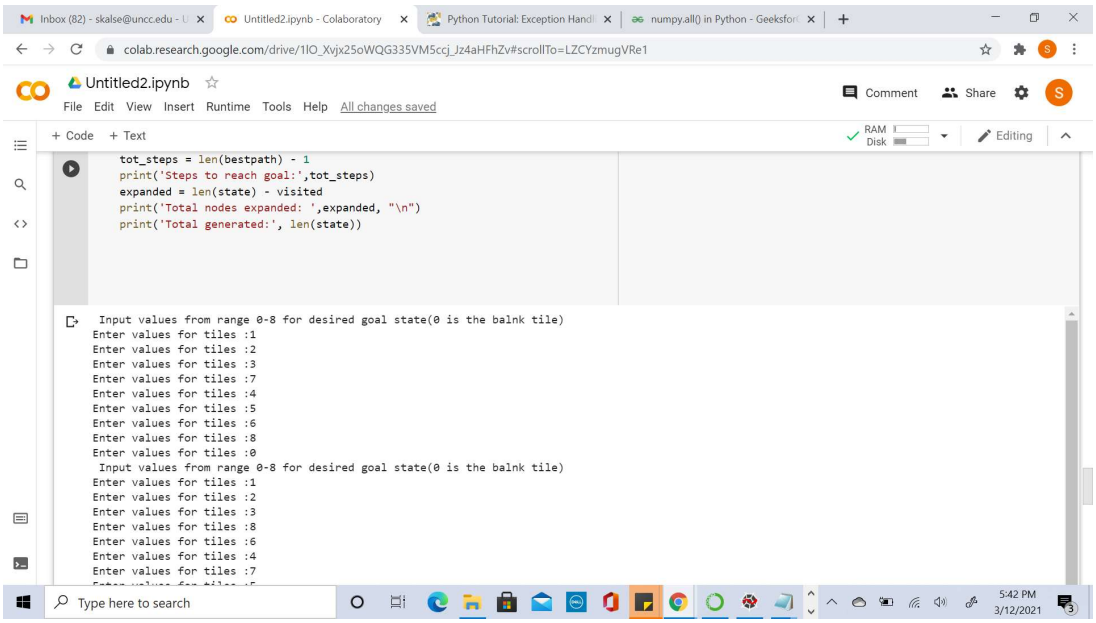
```
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))
```

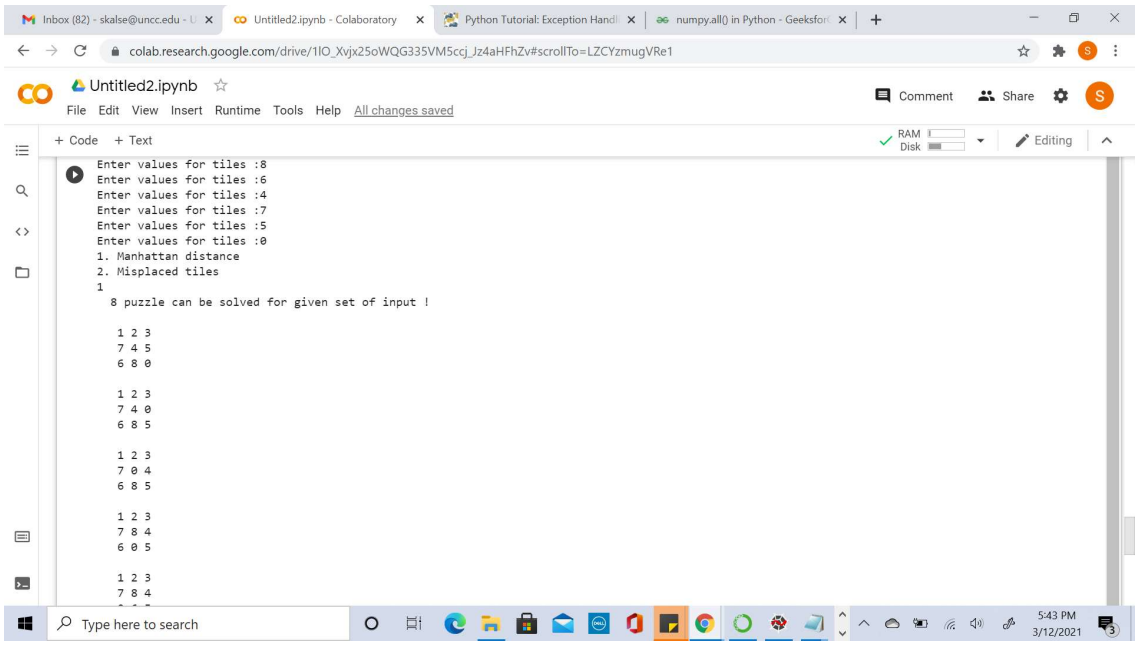
2.4 Result Summary

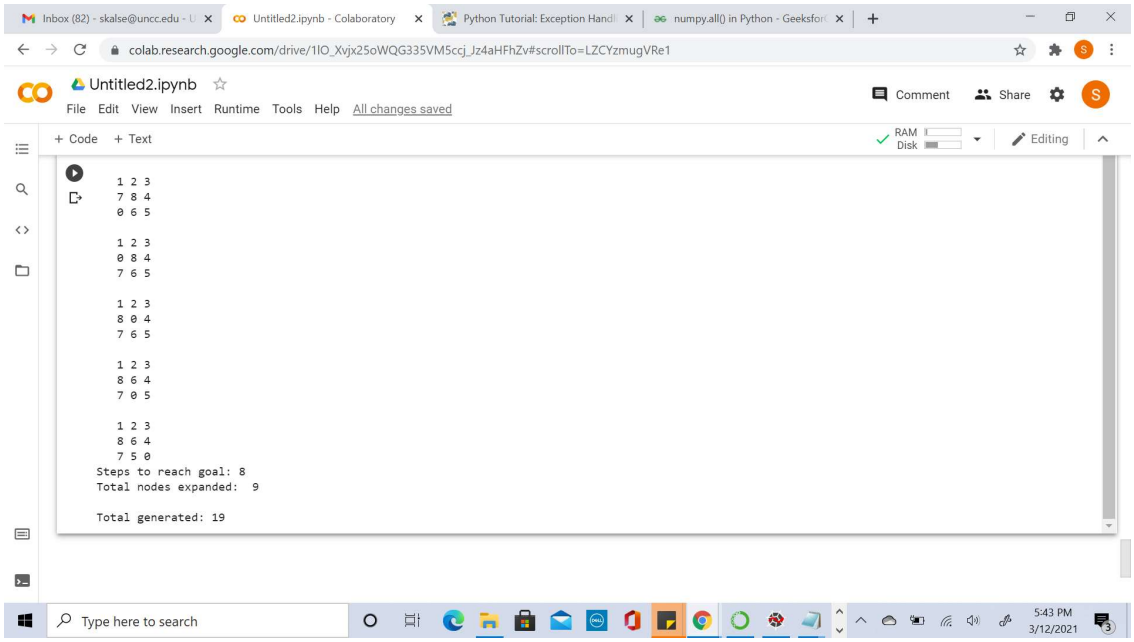
Input 1:

```
Initial state:      Goal State:
1 2 3               1 2 3
7 4 5               8 6 4
6 8 0               7 5 0
```

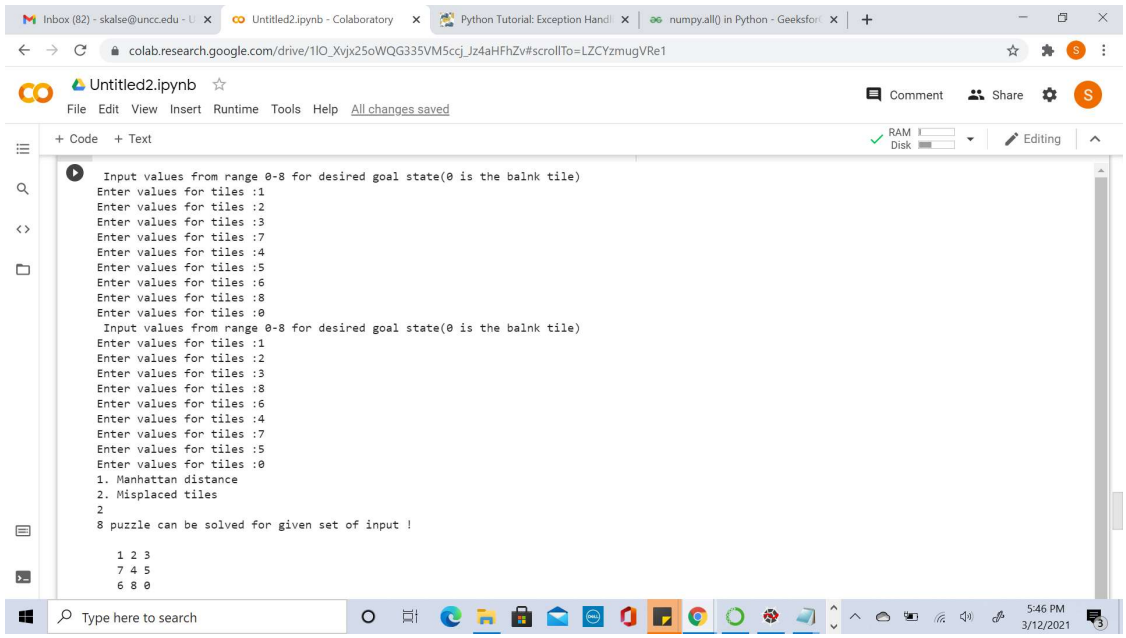
1. Manhattan Distance

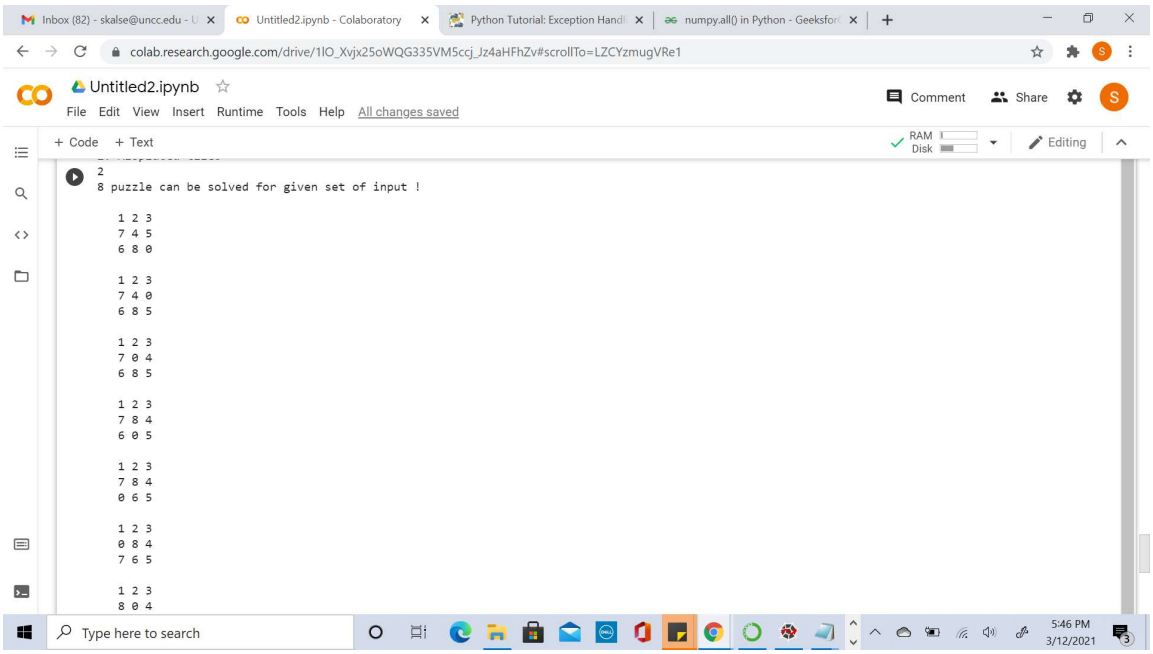


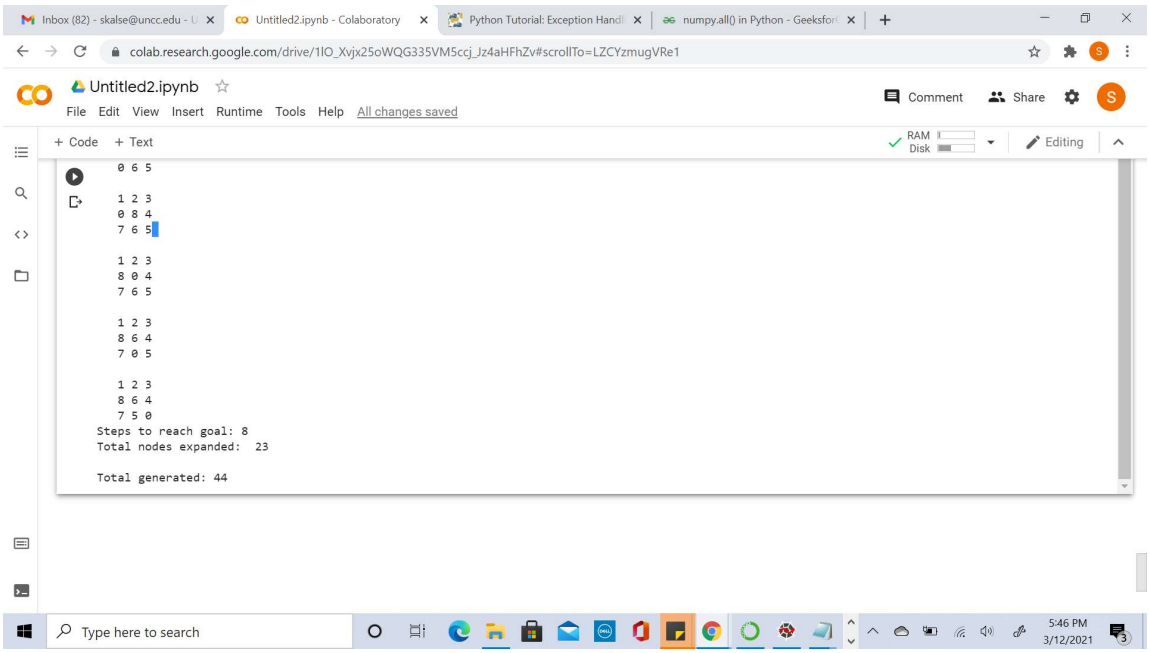




2.Misplaced_tiles


Untitled2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
Input values from range 0-8 for desired goal state(0 is the balnk tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :7
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :8
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the balnk tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :8
Enter values for tiles :6
Enter values for tiles :4
Enter values for tiles :7
Enter values for tiles :5
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
2
8 puzzle can be solved for given set of input !
1 2 3
7 4 5
6 8 0
Type here to search
5:46 PM
3/12/2021

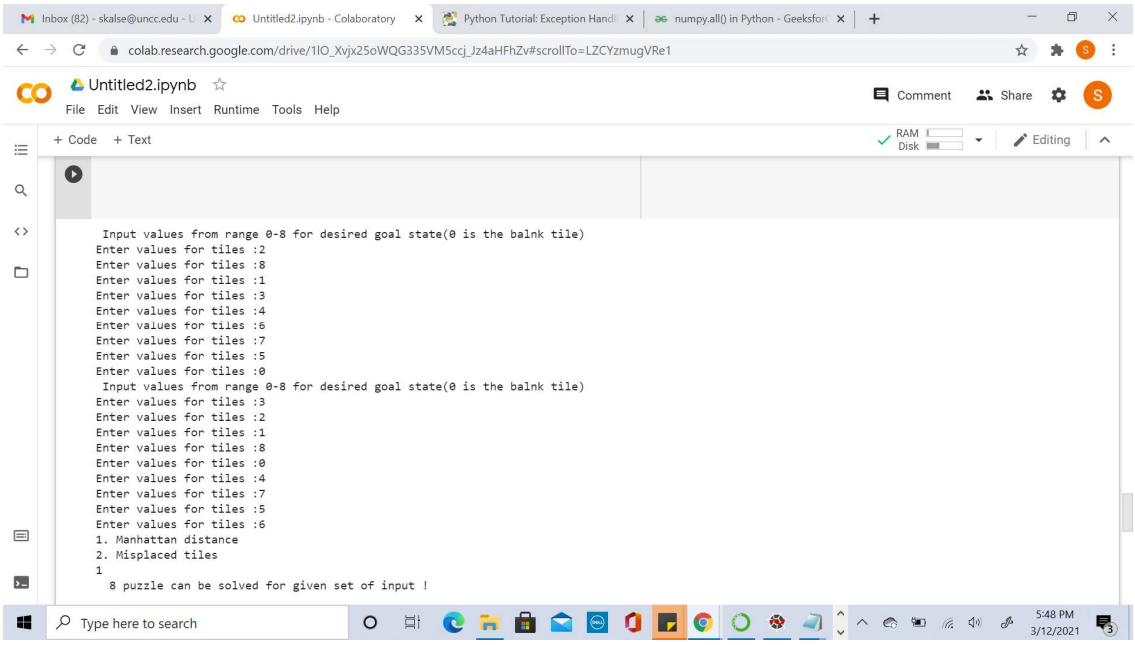

Untitled2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
2
8 puzzle can be solved for given set of input !
1 2 3
7 4 5
6 8 0
1 2 3
7 4 0
6 8 5
1 2 3
7 0 4
6 8 5
1 2 3
7 8 4
6 0 5
1 2 3
7 8 4
0 6 5
1 2 3
0 8 4
7 6 5
1 2 3
8 0 4
Type here to search
5:46 PM
3/12/2021

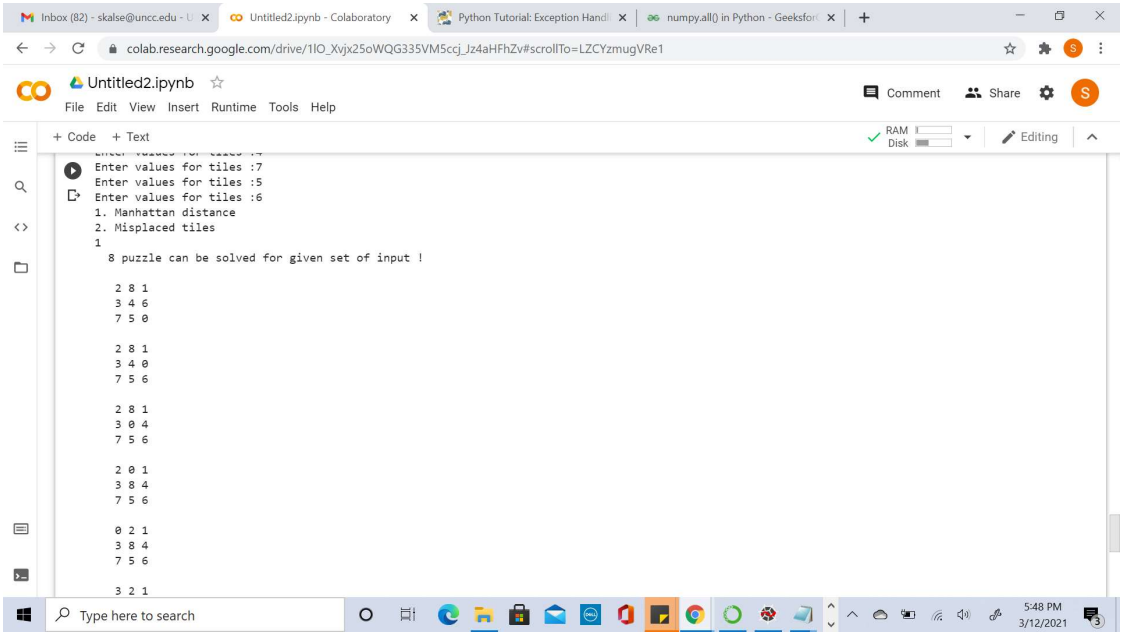


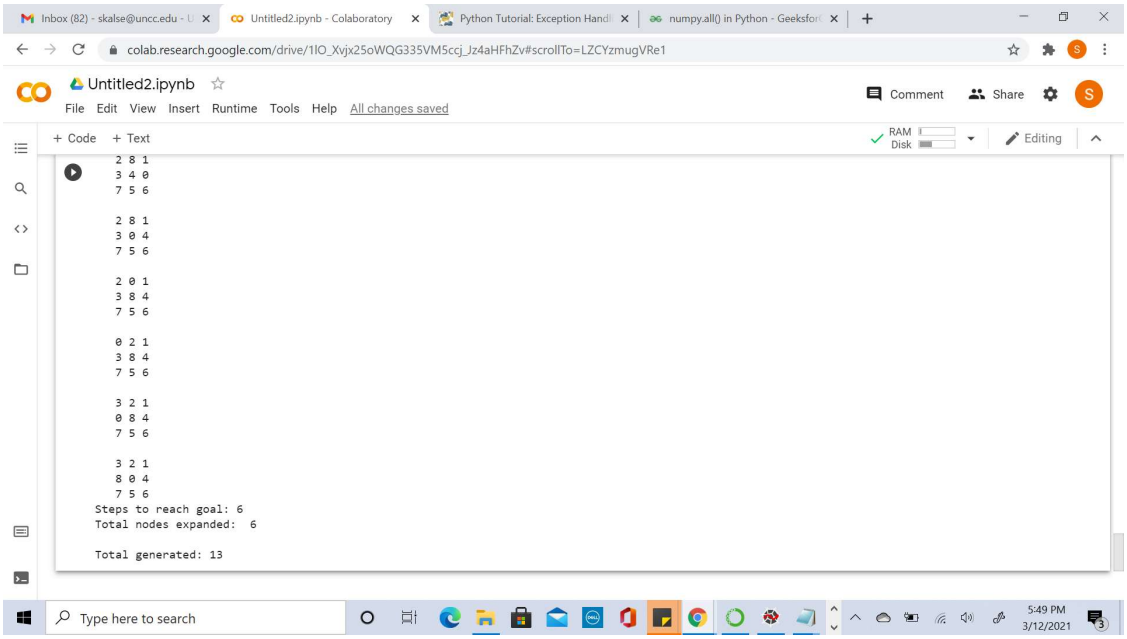
Input2:

Initial state:	Goal State:
2 8 1	3 2 1
3 4 6	8 0 4
7 5 0	7 5 6

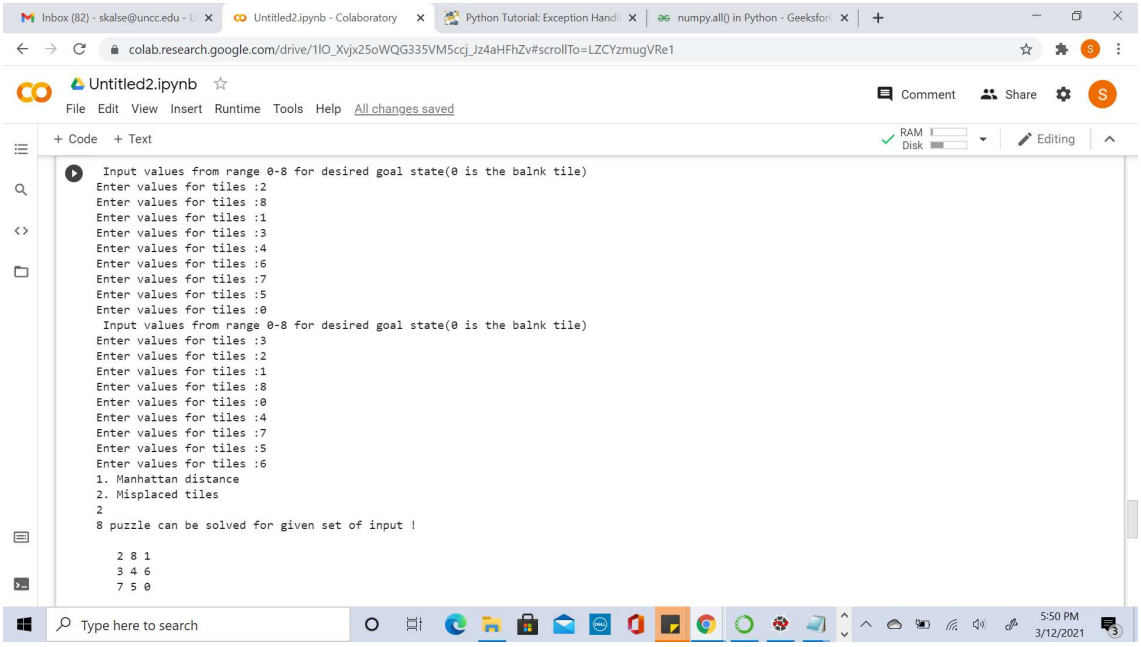
1. Manhattan Distance:

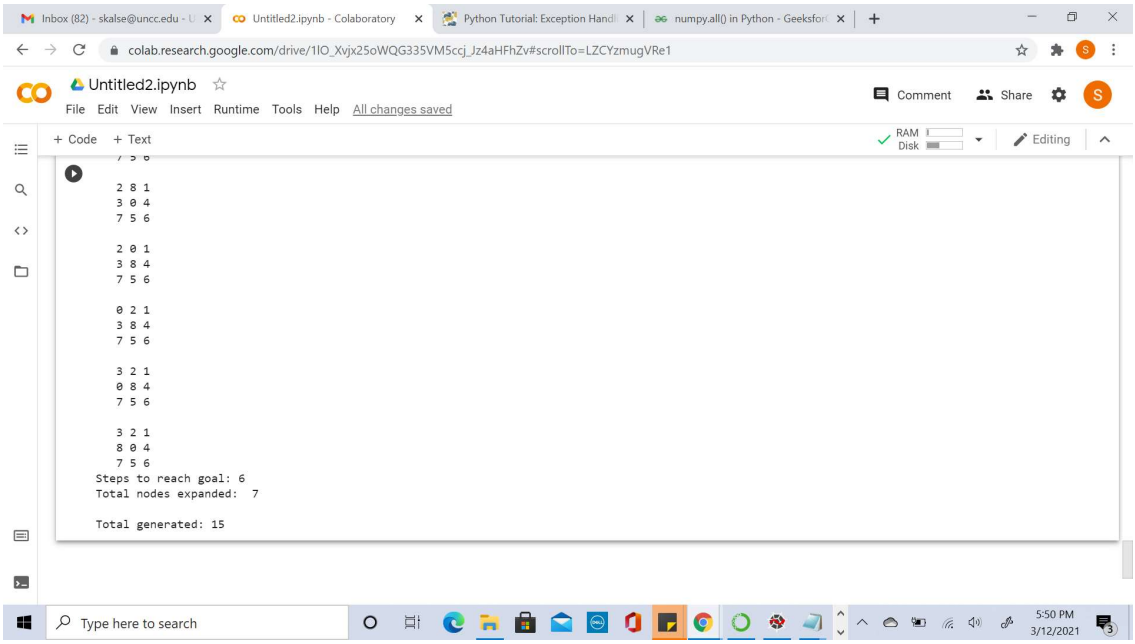






2. Misplaced_tile



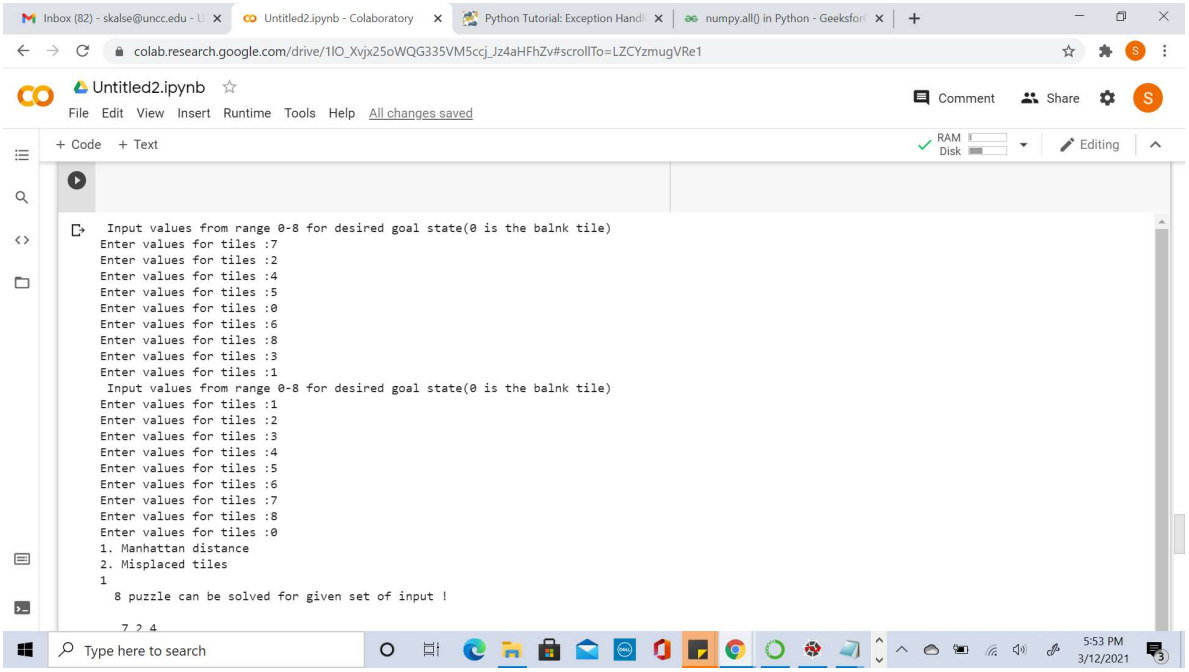


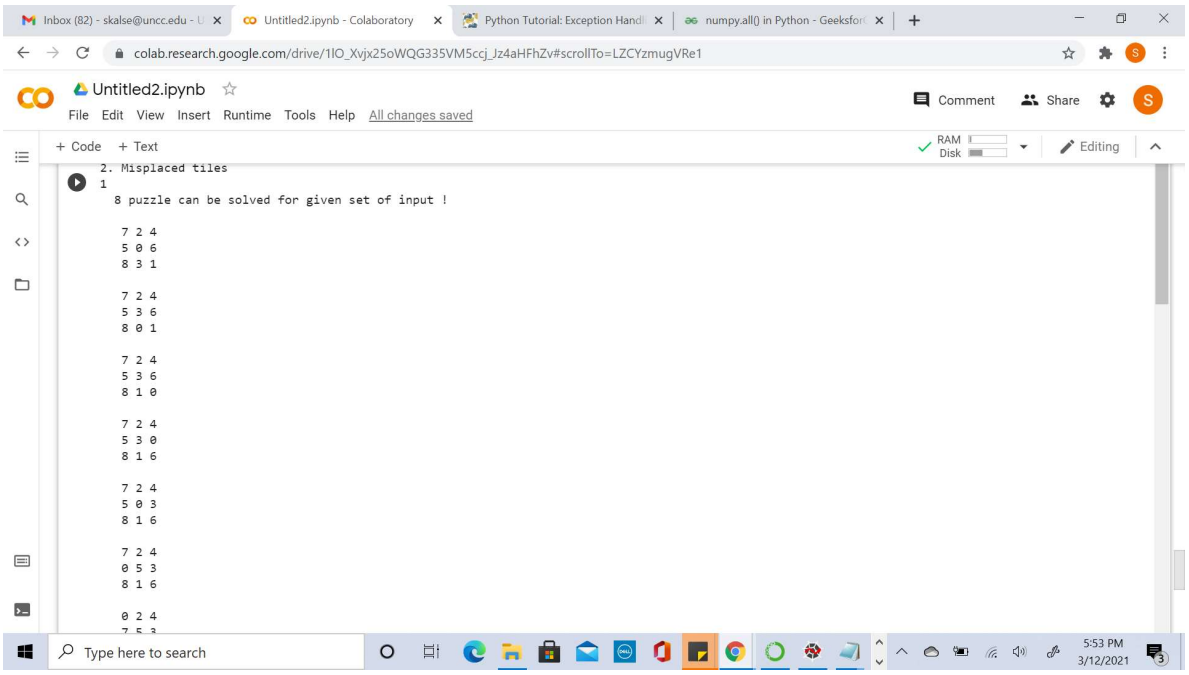
Input 3

Initial state:	Goal State:
7 2 4	1 2 3
5 6	4 5 6
8 3 1	7 8

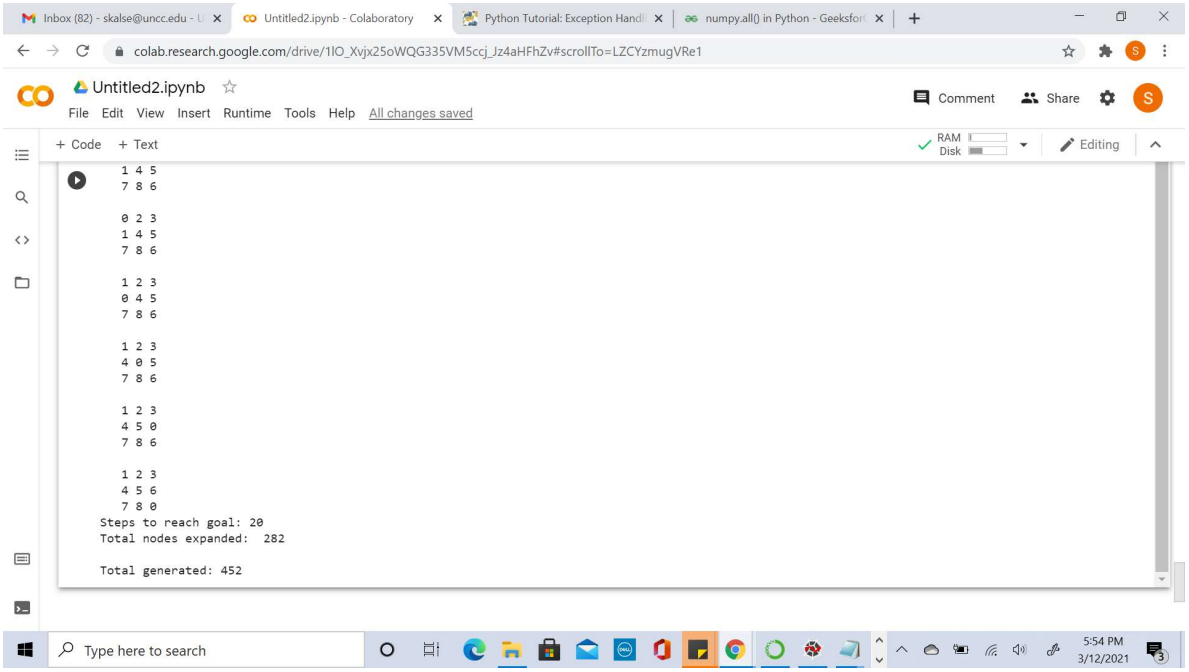
1.Manhattan Distance

Solving 8 – Puzzle using A* Search Algorithm

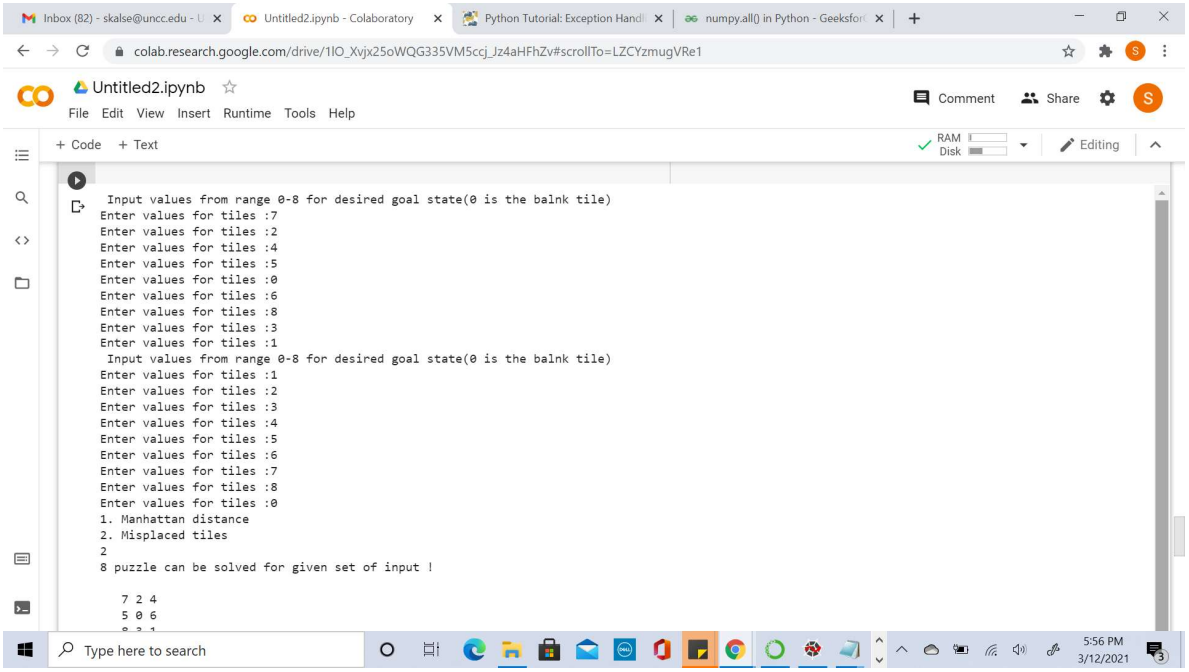


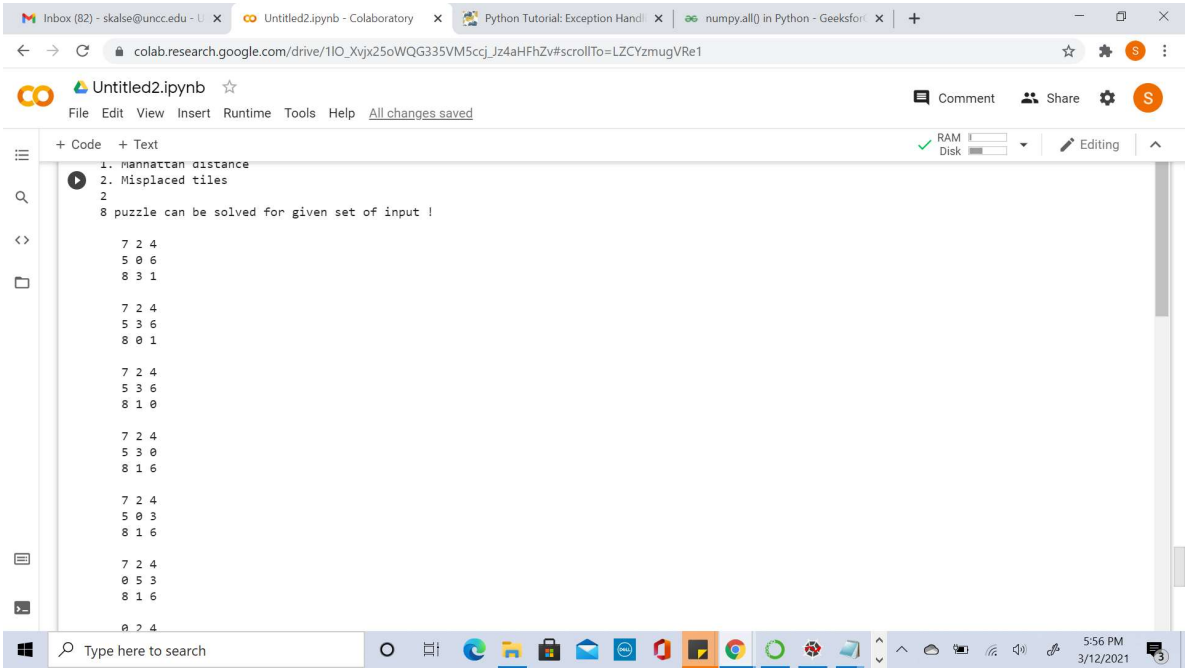


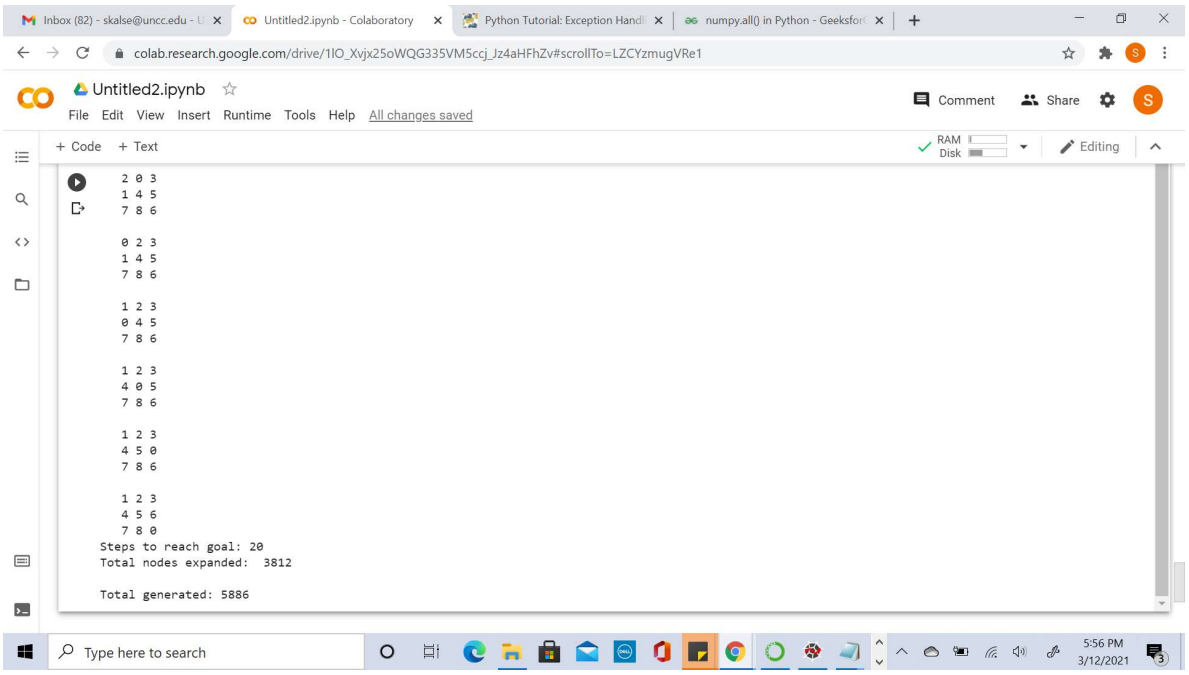
Solving 8 – Puzzle using A* Search Algorithm



2 Misplaced Tile





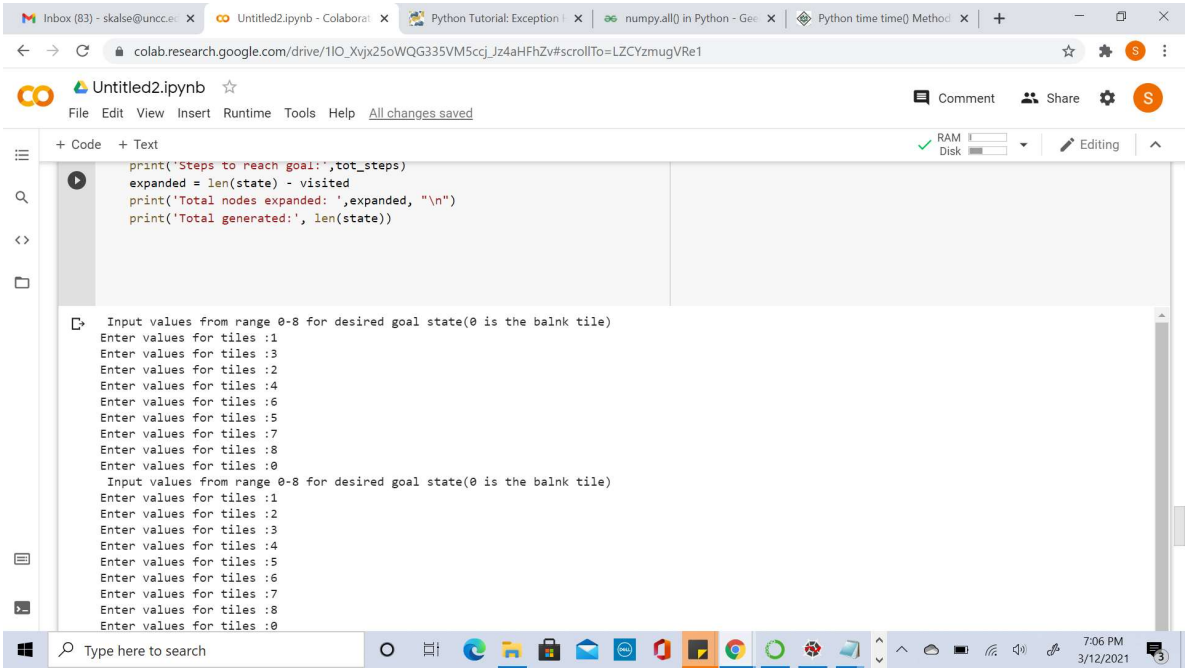


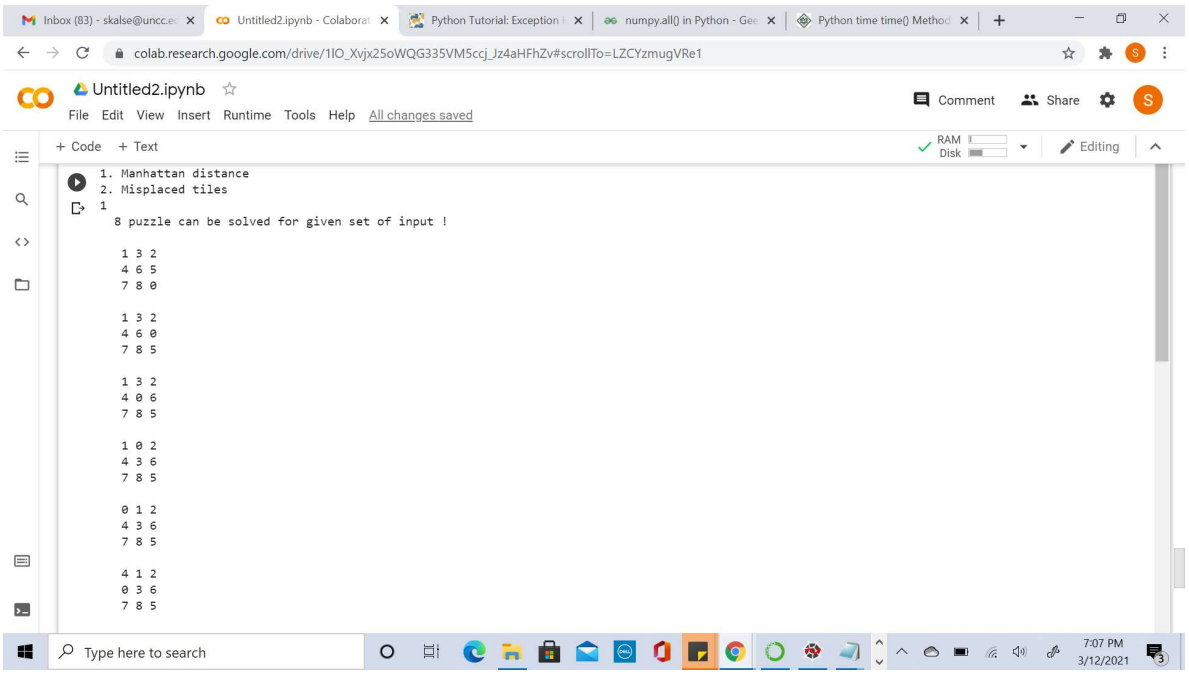
Input 4

Initial state:132465780

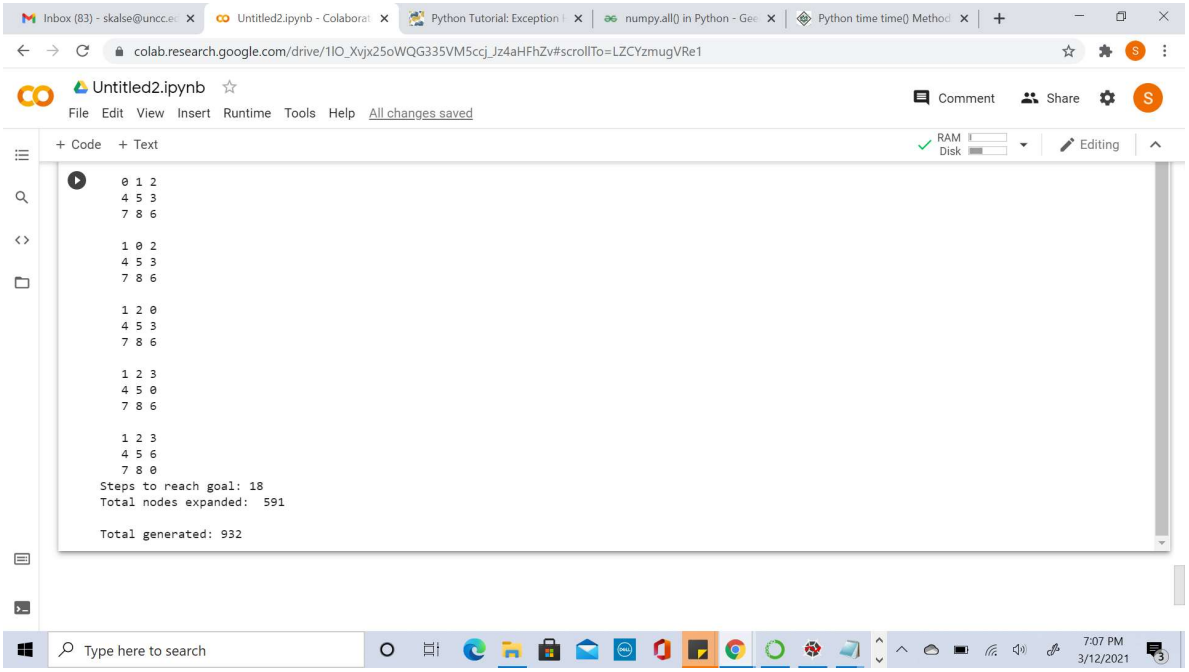
Goal state: 123456780

1.Manhattan Distance

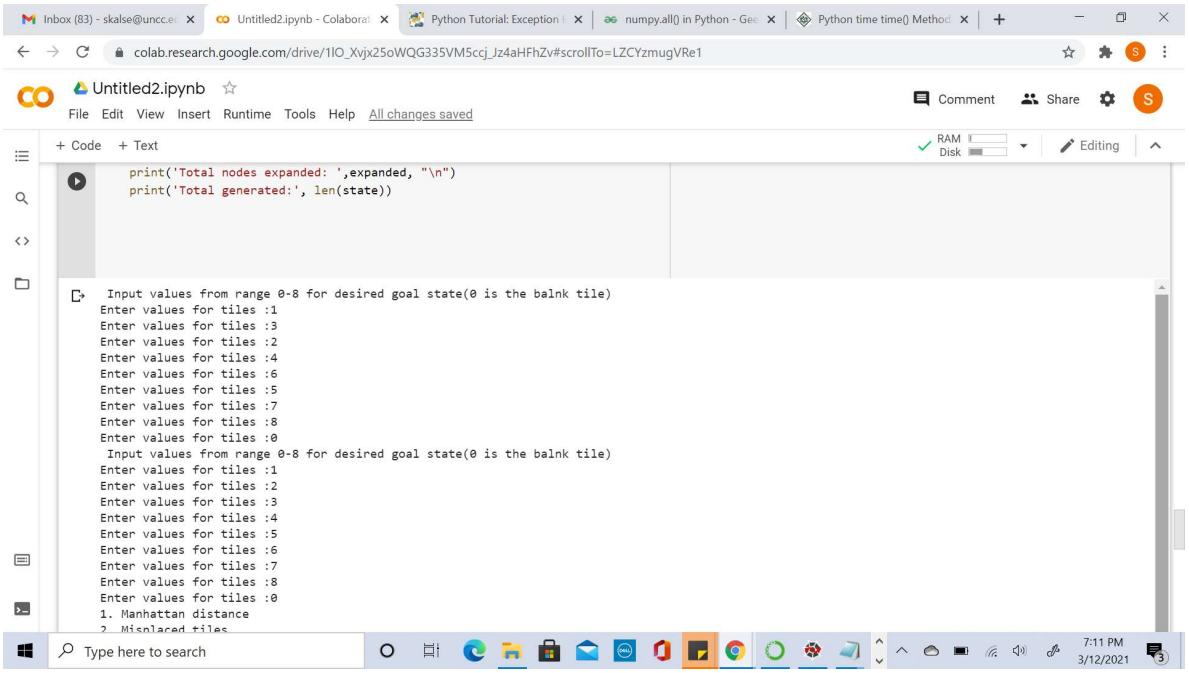

Untitled2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
print('Steps to reach goal:',tot_steps)
expanded = len(state) - visited
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))
Input values from range 0-8 for desired goal state(0 is the balnk tile)
Enter values for tiles :1
Enter values for tiles :3
Enter values for tiles :2
Enter values for tiles :4
Enter values for tiles :6
Enter values for tiles :5
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the balnk tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
Type here to search
7:06 PM
3/12/2021

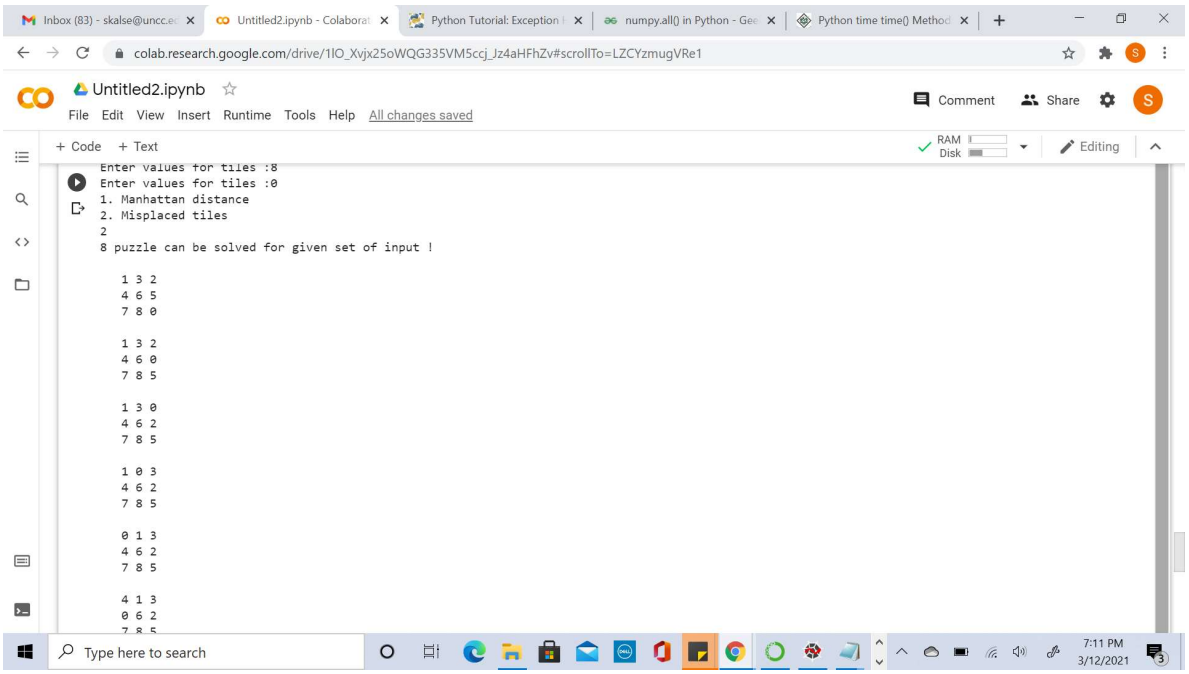

Untitled2.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Editing
1. Manhattan distance
2. Misplaced tiles
1
8 puzzle can be solved for given set of input !
1 3 2
4 6 5
7 8 0
1 3 2
4 6 0
7 8 5
1 3 2
4 0 6
7 8 5
1 0 2
4 3 6
7 8 5
0 1 2
4 3 6
7 8 5
4 1 2
0 3 6
7 8 5
Type here to search
7:07 PM
3/12/2021

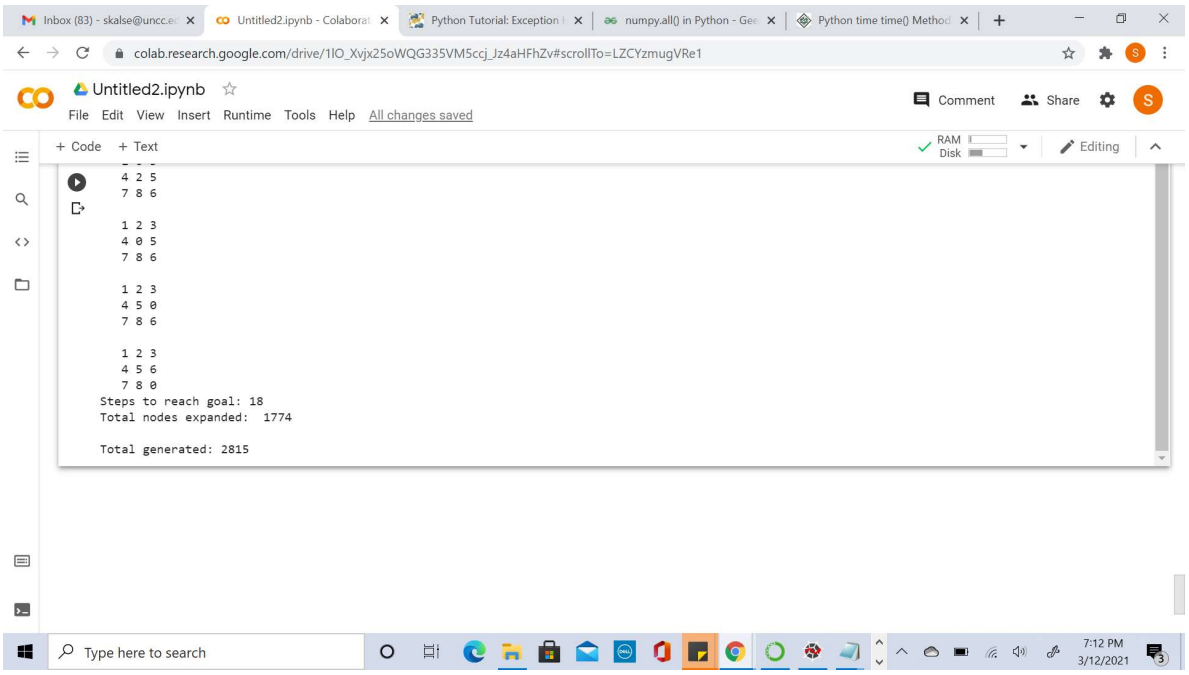
Solving 8 – Puzzle using A* Search Algorithm



2.Misplaced Tile





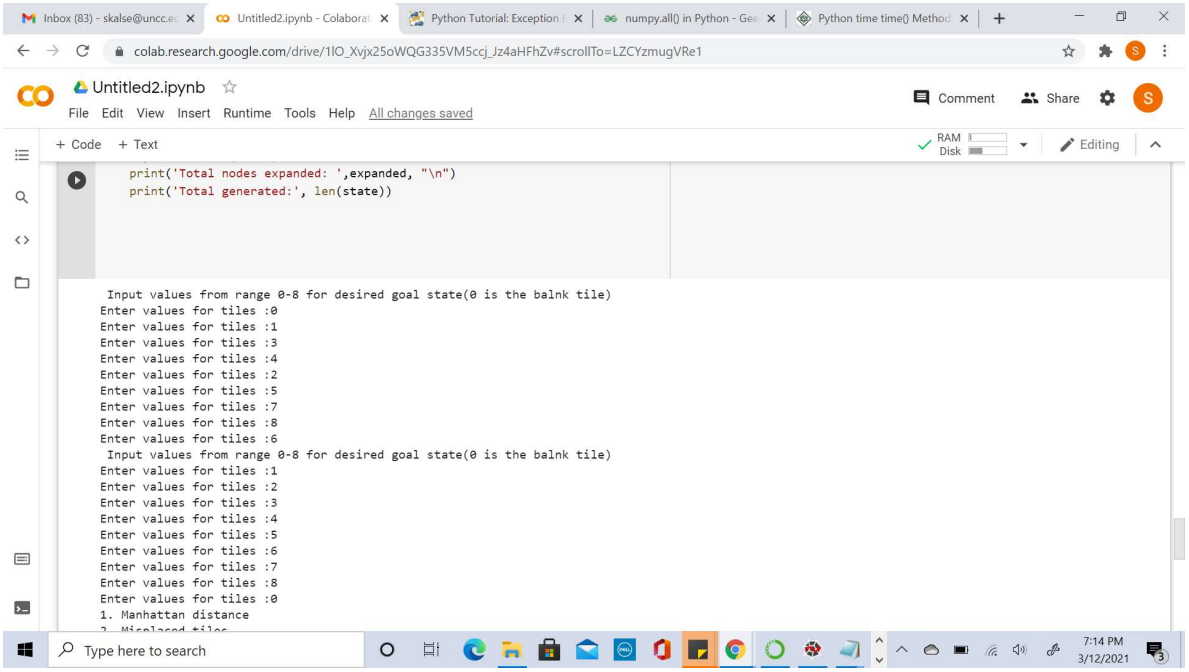


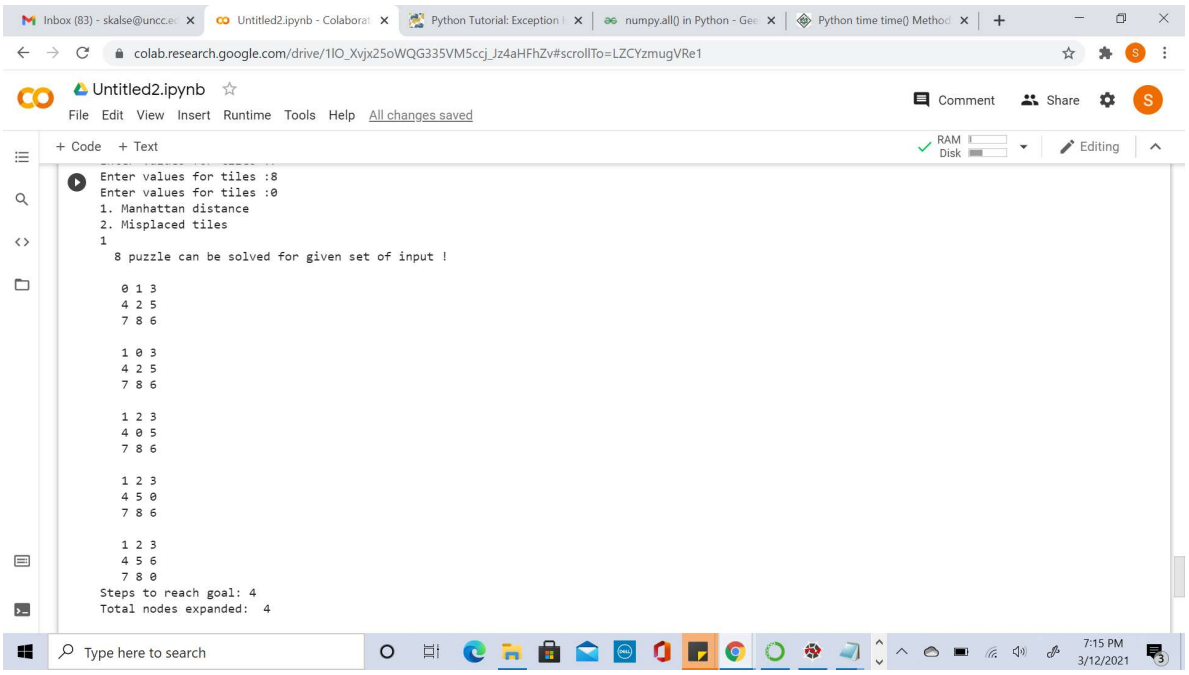
Input 5

Initial state:013425786

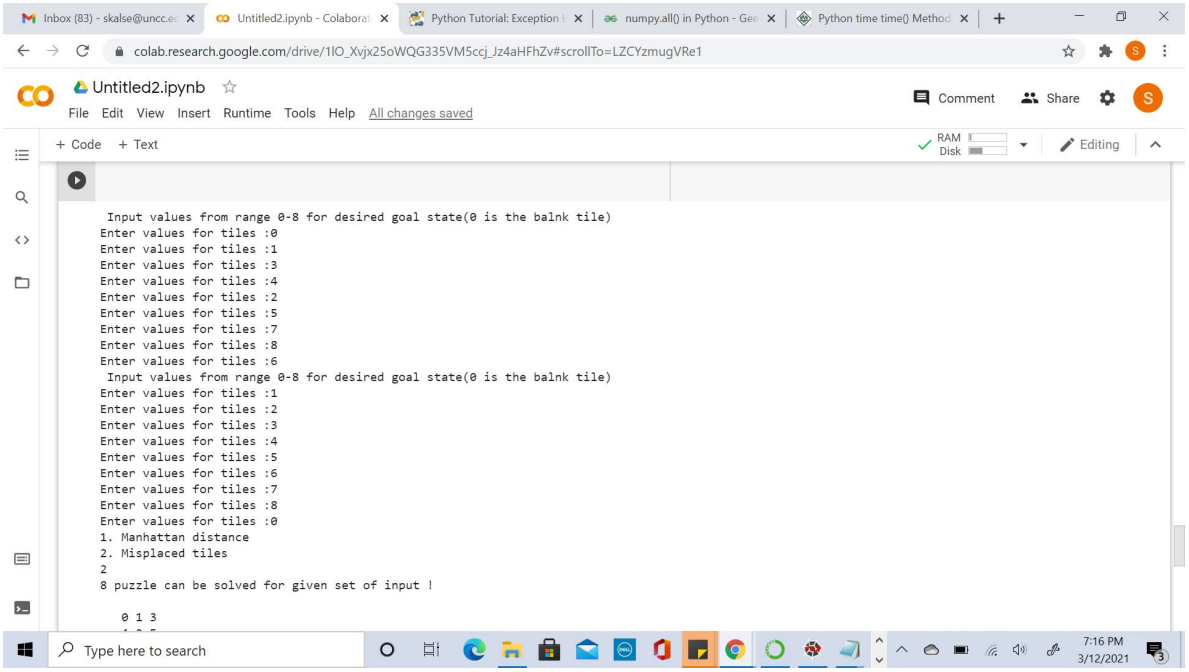
Goal state: 123456780

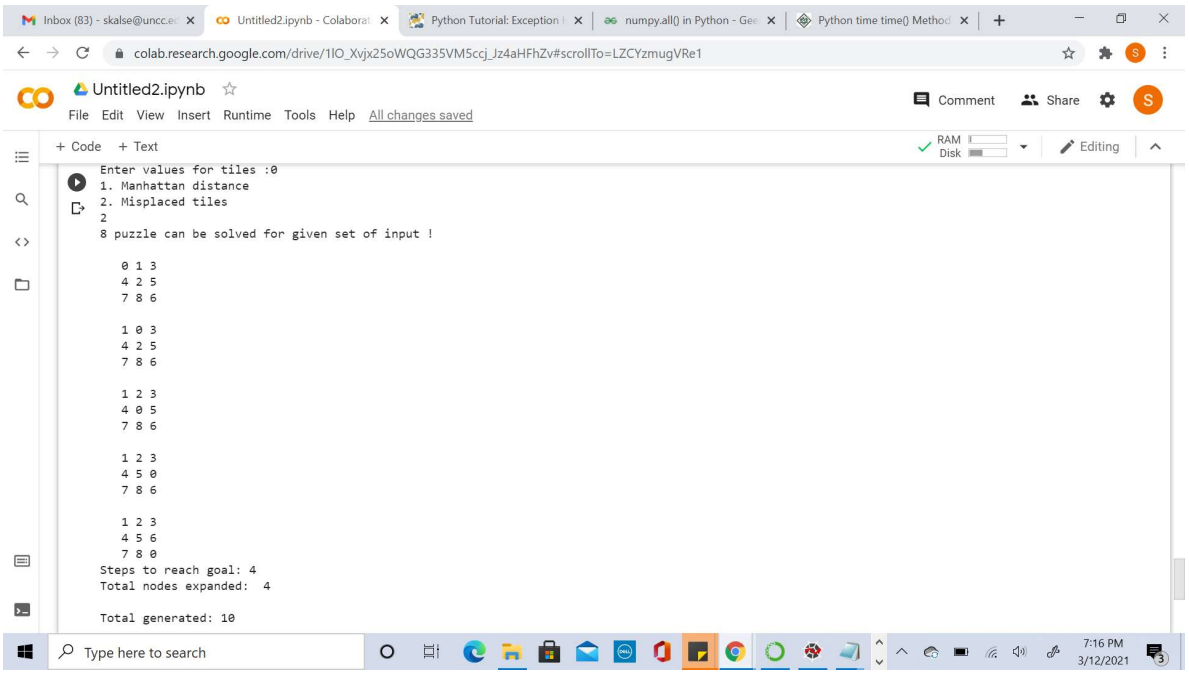
1.Manhattan Distance





2 Misplaced Tile



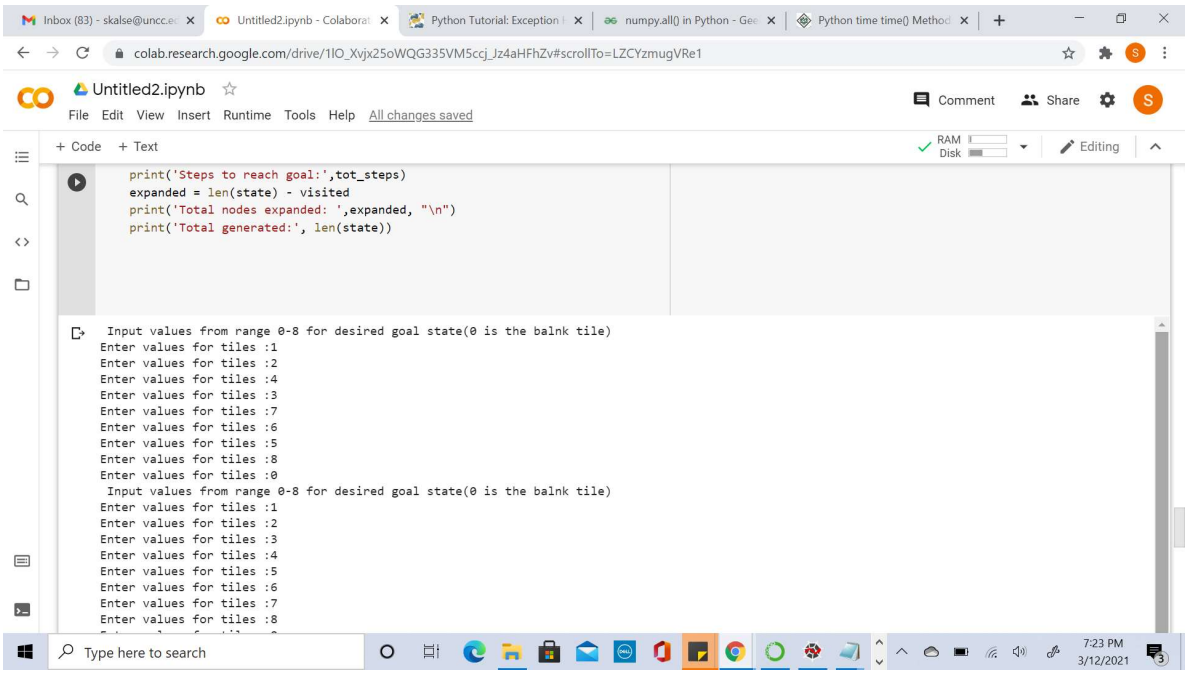


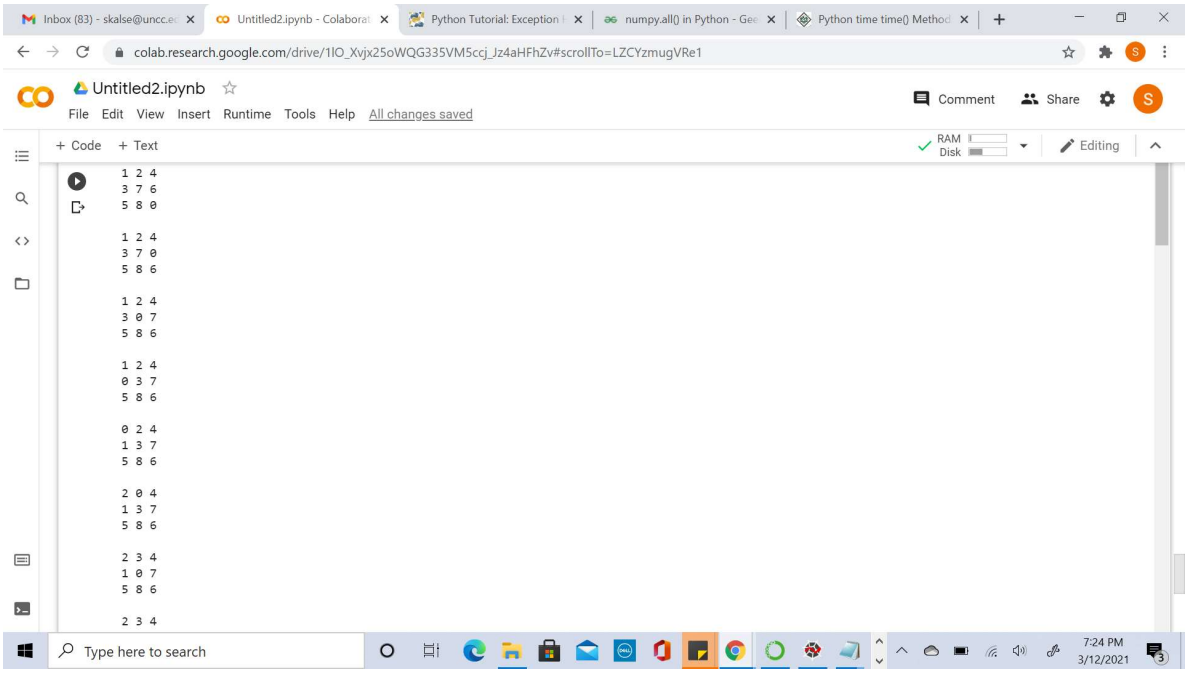
Input6:

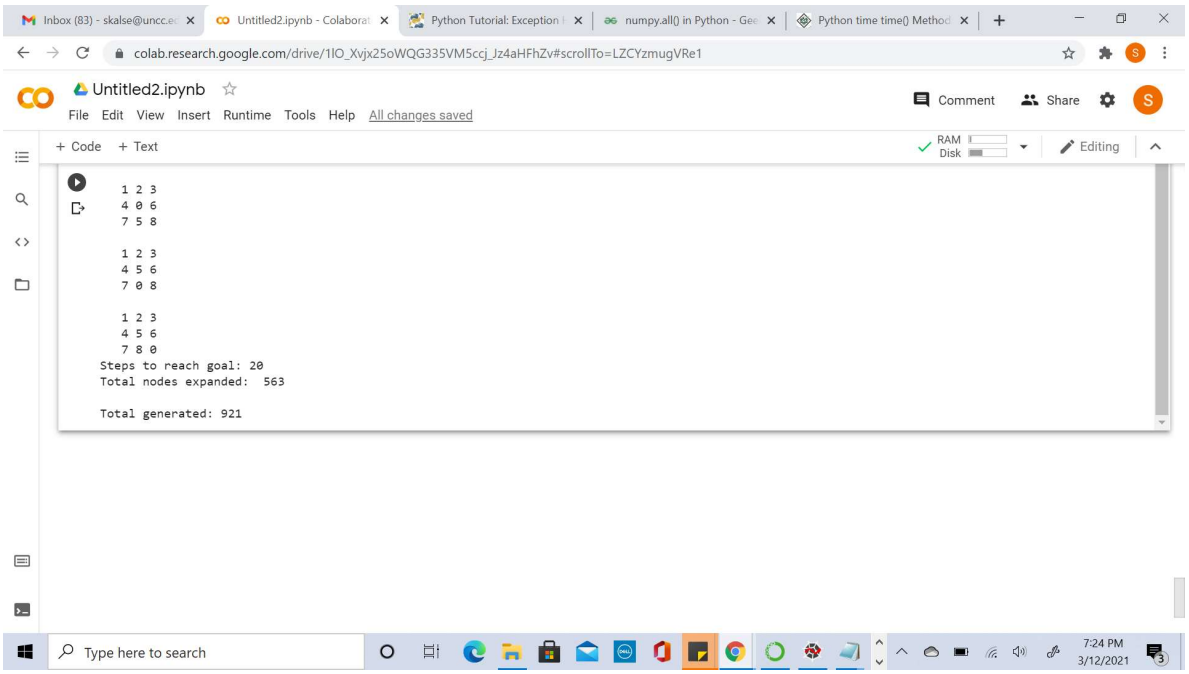
Initial state: 124376580

Goal State: 123456780

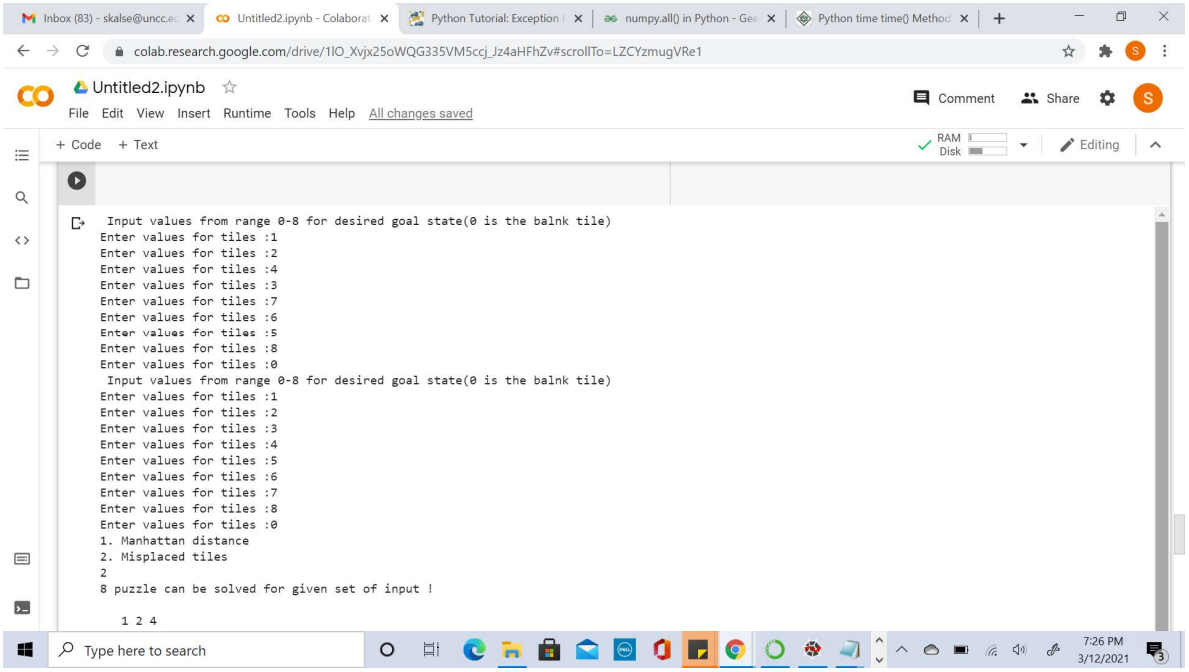
1Manhattan distance

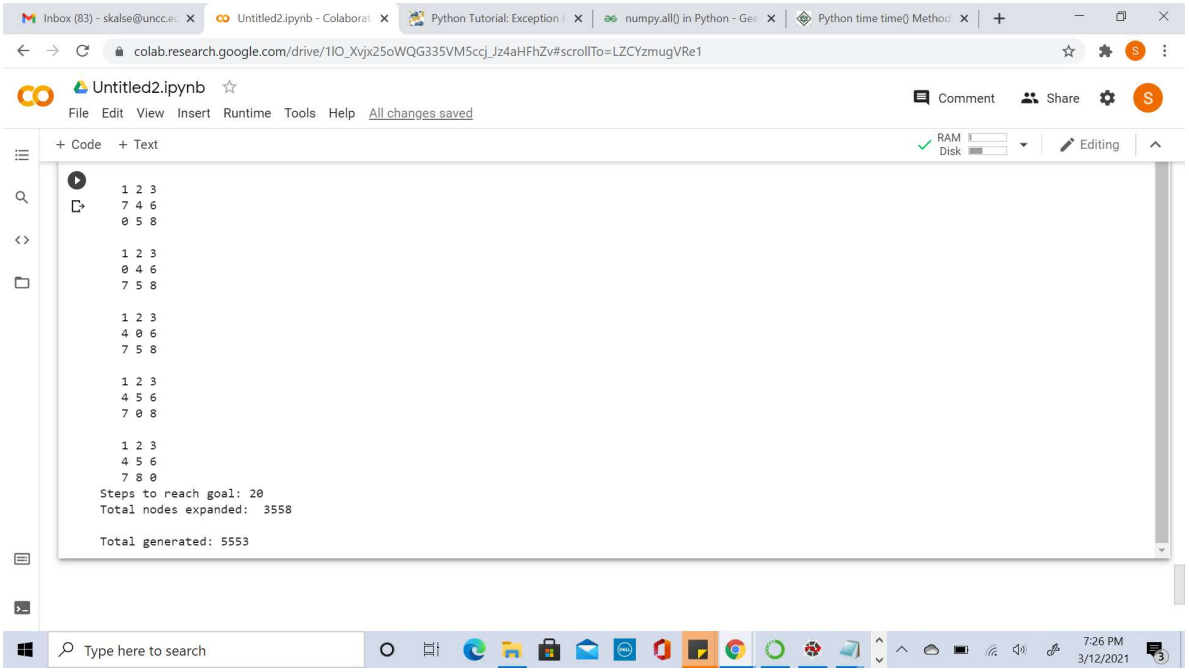






2 Misplaced tile:





Summary In table:

Input no		Steps to Reach the goal	Total nodes expanded	Total generated		Steps to Reach the goal	Total nodes expanded	Total generated
1	Initial state 123745680 Goal state 123864750	8	9	19		8	23	44
2	Initial State 281346750 Goal State 321804756	6	6	13		6	7	15
3	Initial State 724506831 Goal State 123456780	20	282	452		20	3812	5886
4	Initial State 132465780 Goal State 123456780	18	591	932		18	1774	2815
5	Initial State 13425786 Goal State 123456780	4	4	10		4	4	10
6	Initial State 124376580 Goal State 123456780	20	563	921		20	3558	5553

CONCLUSION

As per above results, the 8-puzzle problem is solved using A* algorithm which uses heuristics such as manhattan distance and misplaced tile. Also, number of nodes expanded in manhattan distance heuristics are less as compared to misplaced tiles in most of the cases.

REFERENCES

[1] S. J. R. a. P. Norvig, Artificial Intelligence: A Modern Approach, Alan Apt, 1995.

[2] "What is the A* algorithm?," [Online]. Available: https://www.educative.io/edpresso/what-is-the-a-star-algorithm. [Accessed 12 03 2021].

[3] "PY4E," [Online]. Available: www.py4e.com. [Accessed 12 03 2021].