Karan Gaddam
PB ho
B3
AI
Lab- assignment - 1

- Aim : Solve 8 puzzle problem using A* algorithm

- Objective: To study and implement A* algorithm for 8-puzzle problem.

- Theory

→ Best-first search methods and OR graphs

BFS is a traversal technique that decides which node is to be visited next by checking which node is to most promising one and then check it.
A solution graph of OR graph is a subgraph of OR graph which represent on deviation fora solution of the problem.

→ 8- puzzle problem

8- puzzle is played on a 3 by 3 grid with 8 square blocks labelled as 1 to 8 and a blank space. The goal of the problem is to reach the goal state by sliding the blocks one at a time.

Page No.:

Date:

→ A* algorithm

A* algorithm is used to approximate the shortest path in real life situations like in maps.
We use a set data structure for best optimization of A* algorithm with boolean hash table for closed list

- Input : Initial State
- Output : Goal state with optimal path
- Algorithm

① Initialize open list.
② Initialize closed list and put starting node in open list
③ While open list is not empty.
  ⓐ find node with least f, call it "q".
  ⓑ pop q off the open list;
  ⓒ generate q's successor and set their parent to q
  ⓓ for each successor
    i) if successor is goal, stop.
    ii) else compute both g and h for successor.
    iii) if node with same position as successor is in open list, skip the successor.
    iv) If a node with same position as successor is in closed list with a lower f then successor skip, or add node to open list.
    end for loop.
  ⓔ push q on closed list.
  end (while loop)

- FAQ's

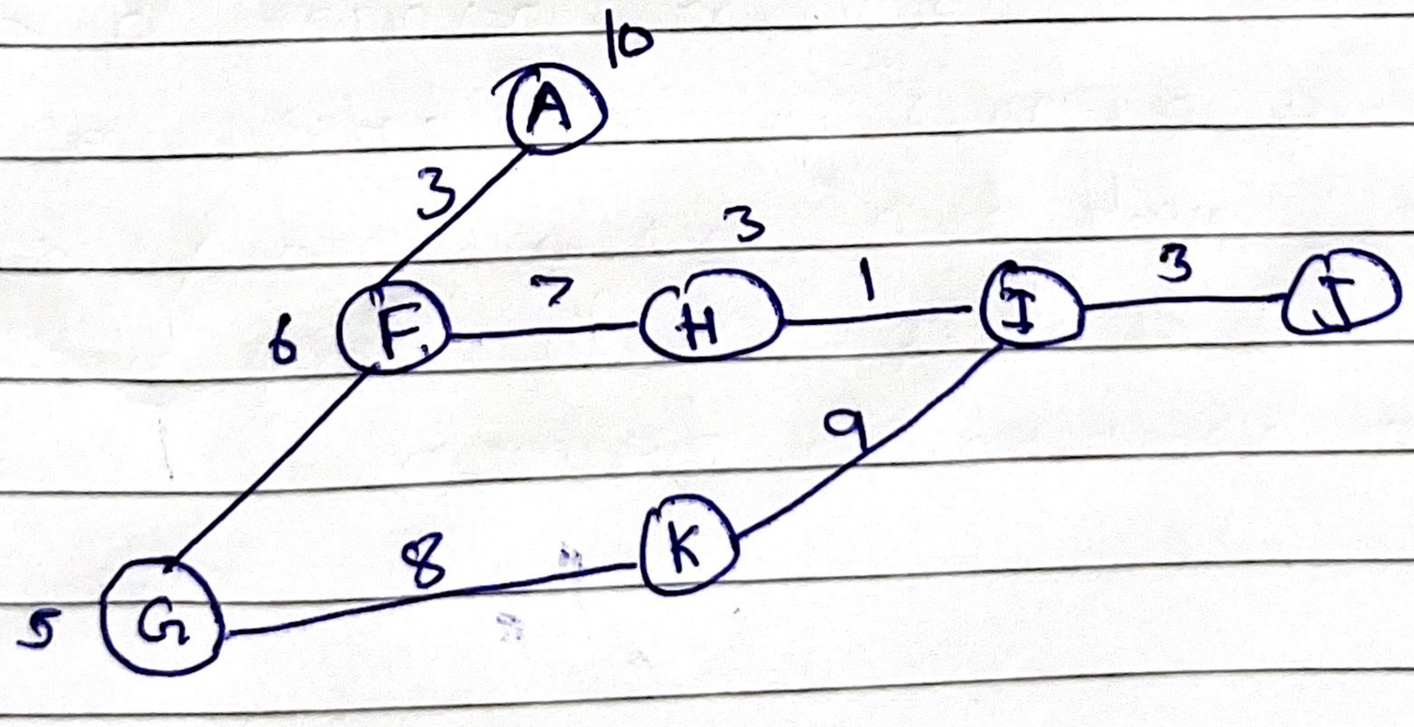
Q.1 What is a heuristic function? What is the advantage of using a heuristic function?

A A heuristic function is a shortcut to solving a problem when there are no exact solutions for it or the time to obtain a solution is too large.
It can provide quick and relatively inexpensive feedback to the developer.

Q.2 Explain A^* algorithm with example

A: A^* algorithm maintains a tree of paths and extends the path that minimizes the following function.

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



Find cost effective path from A to J.
We start with node A and move to F.

Path $\Rightarrow$ A $\longrightarrow$ F

G and H nodes can be reached

$F(G) = 3 + 1 + 5 = 9$

$F(H) = 3 + 7 + 3 = 13$

Path ⇒ A → F → G

no choice till now

∴ Path ⇒ A → F → G → K → I → J

Q.3 Explain different heuristic functions that can be used for 8 puzzle problems.

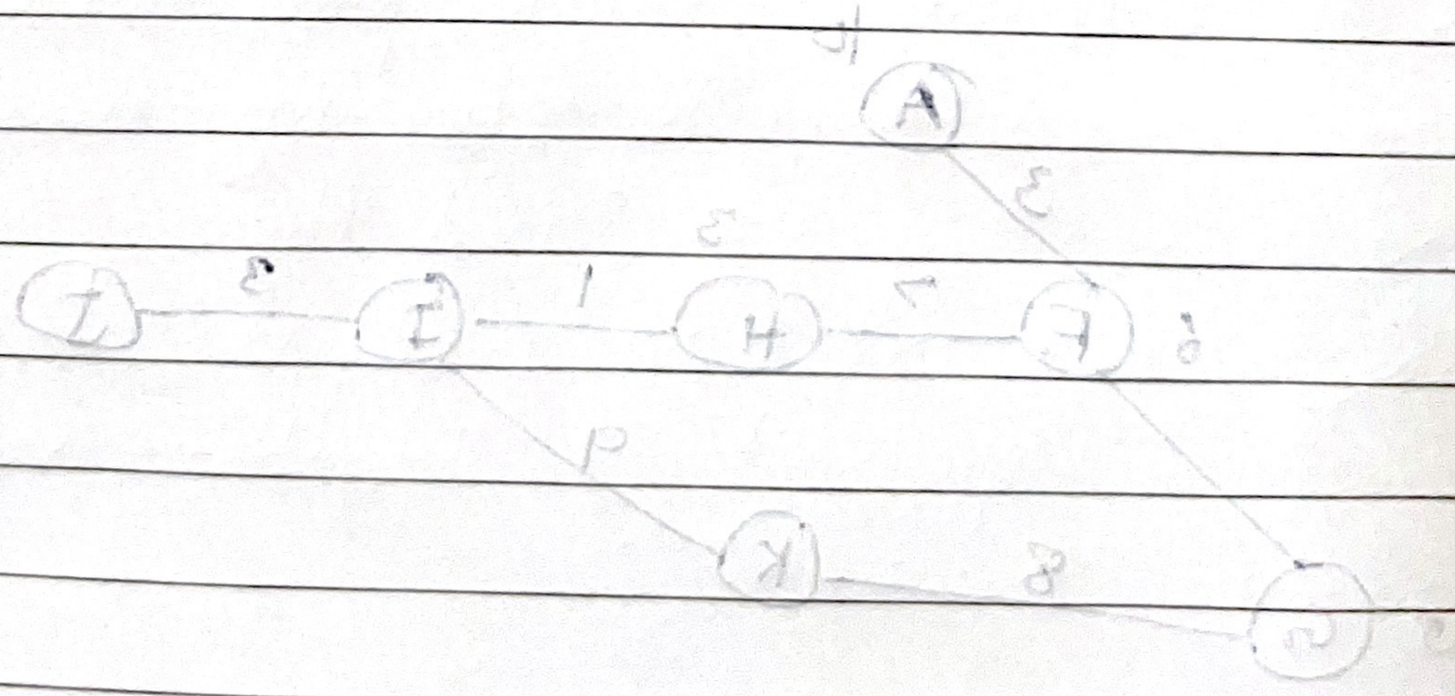
A Exact Heuristic : We can find exact values of h by pre computing distance between each nodes.

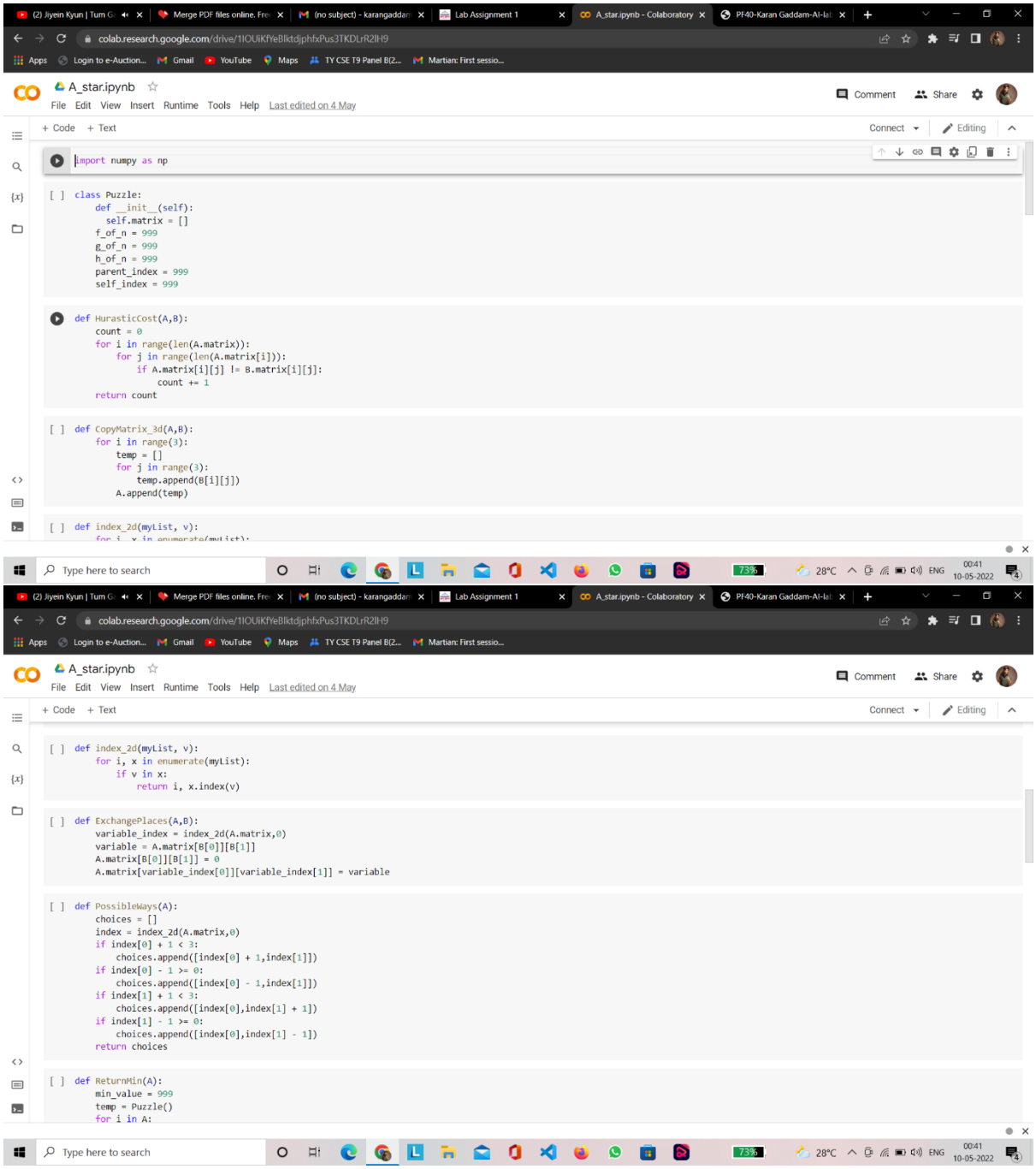
Approximate heuristic : Finds approximate values of h using one of the following

① Manhattan Distance

② Diagonal Distance

③ Euclidean Distance





```
import numpy as np
```

```
class Puzzle:
    def __init__(self):
      self.matrix = []
    f_of_n = 999
    g_of_n = 999
    h_of_n = 999
    parent_index = 999
    self_index = 999
```

```
def HuristicCost(A,B):
    count = 0
    for i in range(len(A.matrix)):
        for j in range(len(A.matrix[i])):
            if A.matrix[i][j] != B.matrix[i][j]:
                count += 1
    return count
```

```
def CopyMatrix_3d(A,B):
    for i in range(3):
        temp = []
        for j in range(3):
            temp.append(B[i][j])
        A.append(temp)
```

```
def index_2d(myList, v):
```

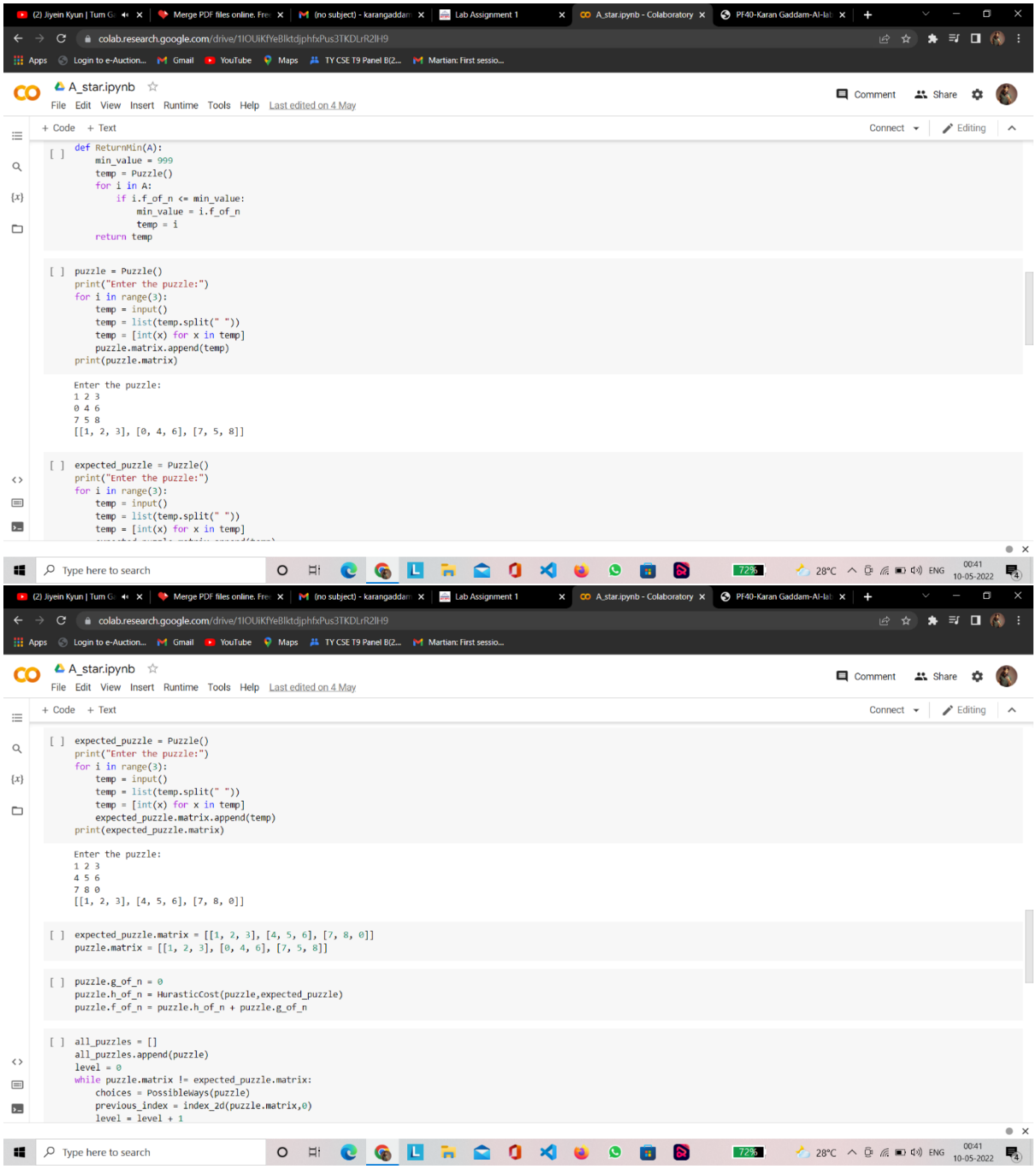


```
def index_2d(myList, v):
    for i, x in enumerate(myList):
        if v in x:
            return i, x.index(v)
```

```
def ExchangePlaces(A,B):
    variable_index = index_2d(A.matrix,0)
    variable = A.matrix[B[0]][B[1]]
    A.matrix[B[0]][B[1]] = 0
    A.matrix[variable_index[0]][variable_index[1]] = variable
```

```
def PossibleWays(A):
    choices = []
    index = index_2d(A.matrix,0)
    if index[0] + 1 < 3:
        choices.append([index[0] + 1,index[1]])
    if index[0] - 1 >= 0:
        choices.append([index[0] - 1,index[1]])
    if index[1] + 1 < 3:
        choices.append([index[0],index[1] + 1])
    if index[1] - 1 >= 0:
        choices.append([index[0],index[1] - 1])
    return choices
```

```
def ReturnMin(A):
    min_value = 999
    temp = Puzzle()
    for i in A:
```



```
def ReturnMin(A):
    min_value = 999
    temp = Puzzle()
    for i in A:
        if i.f_of_n <= min_value:
            min_value = i.f_of_n
            temp = i
    return temp
```

```
puzzle = Puzzle()
print("Enter the puzzle:")
for i in range(3):
    temp = input()
    temp = list(temp.split(" "))
    temp = [int(x) for x in temp]
    puzzle.matrix.append(temp)
print(puzzle.matrix)
```

```
Enter the puzzle:
1 2 3
0 4 6
7 5 8
[[1, 2, 3], [0, 4, 6], [7, 5, 8]]
```

```
expected_puzzle = Puzzle()
print("Enter the puzzle:")
for i in range(3):
    temp = input()
    temp = list(temp.split(" "))
    temp = [int(x) for x in temp]
```



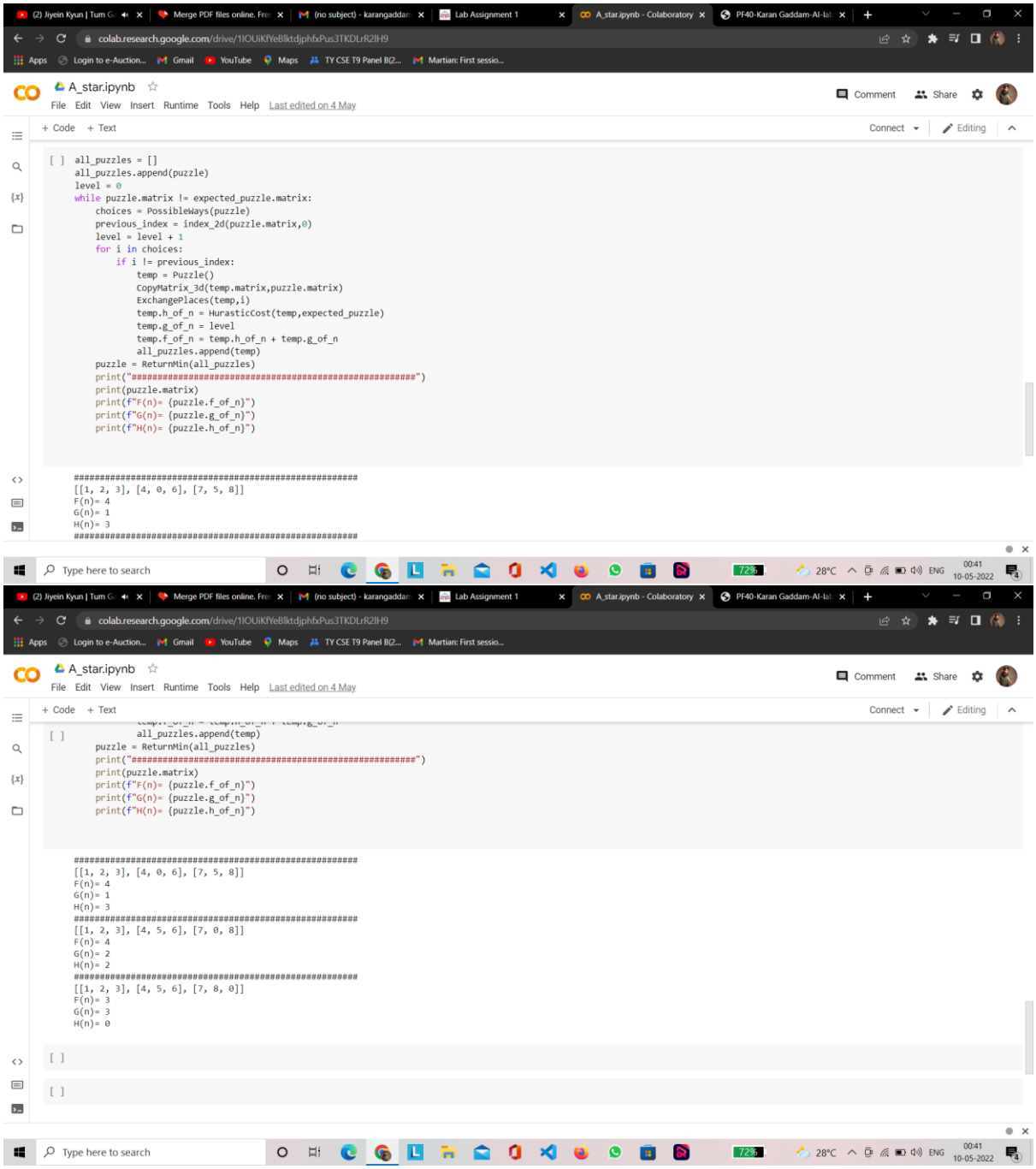
```
expected_puzzle = Puzzle()
print("Enter the puzzle:")
for i in range(3):
    temp = input()
    temp = list(temp.split(" "))
    temp = [int(x) for x in temp]
    expected_puzzle.matrix.append(temp)
print(expected_puzzle.matrix)
```

```
Enter the puzzle:
1 2 3
4 5 6
7 8 0
[[1, 2, 3], [4, 5, 6], [7, 8, 0]]
```

```
expected_puzzle.matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
puzzle.matrix = [[1, 2, 3], [0, 4, 6], [7, 5, 8]]
```

```
puzzle.g_of_n = 0
puzzle.h_of_n = HurasticCost(puzzle,expected_puzzle)
puzzle.f_of_n = puzzle.h_of_n + puzzle.g_of_n
```

```
all_puzzles = []
all_puzzles.append(puzzle)
level = 0
while puzzle.matrix != expected_puzzle.matrix:
    choices = PossibleWays(puzzle)
    previous_index = index_2d(puzzle.matrix,0)
    level = level + 1
```



```
all_puzzles = []
all_puzzles.append(puzzle)
level = 0
while puzzle.matrix != expected_puzzle.matrix:
    choices = PossibleWays(puzzle)
    previous_index = index_2d(puzzle.matrix,0)
    level = level + 1
    for i in choices:
        if i != previous_index:
            temp = Puzzle()
            CopyMatrix_3d(temp.matrix,puzzle.matrix)
            ExchangePlaces(temp,i)
            temp.h_of_n = HurasticCost(temp,expected_puzzle)
            temp.g_of_n = level
            temp.f_of_n = temp.h_of_n + temp.g_of_n
            all_puzzles.append(temp)
    puzzle = ReturnMin(all_puzzles)
    print("#####################################################")
    print(puzzle.matrix)
    print(f"F(n)= {puzzle.f_of_n}")
    print(f"G(n)= {puzzle.g_of_n}")
    print(f"H(n)= {puzzle.h_of_n}")
```

```
#####################################################
[[1, 2, 3], [4, 0, 6], [7, 5, 8]]
F(n)= 4
G(n)= 1
H(n)= 3
#####################################################
```



```
            all_puzzles.append(temp)
    puzzle = ReturnMin(all_puzzles)
    print("#####################################################")
    print(puzzle.matrix)
    print(f"F(n)= {puzzle.f_of_n}")
    print(f"G(n)= {puzzle.g_of_n}")
    print(f"H(n)= {puzzle.h_of_n}")
```

```
#####################################################
[[1, 2, 3], [4, 0, 6], [7, 5, 8]]
F(n)= 4
G(n)= 1
H(n)= 3
#####################################################
[[1, 2, 3], [4, 5, 6], [7, 0, 8]]
F(n)= 4
G(n)= 2
H(n)= 2
#####################################################
[[1, 2, 3], [4, 5, 6], [7, 8, 0]]
F(n)= 3
G(n)= 3
H(n)= 0
```

```
[ ]
```

```
[ ]
```