AI Assignment 5

---------------------------

Name: Yash Oswal

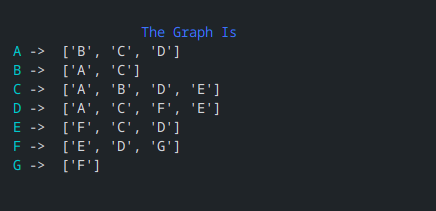
Div: B Roll no: 38

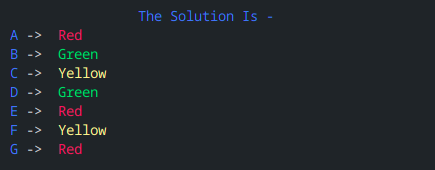
SRN: 201901226

---------------------------

1. Map Colouring

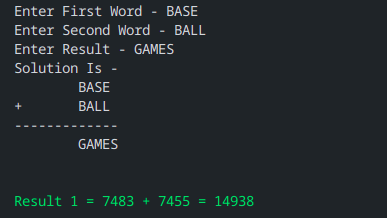
Output:





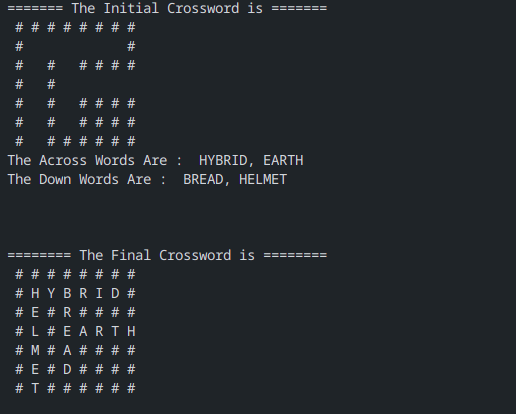
1. Cryptharithmatic

Output:



1. Crossword Puzzle:

Output



Code:

1. Map Colouring

from colorama import Fore, Back, Style, init

init(strip=False)

init(autoreset=True)

class map\_coloring():

# Colors Used

colors = [Fore.RED+'Red', Fore.GREEN+'Green', Fore.YELLOW+'Yellow',

Fore.MAGENTA+'Violet']

# Map

states = ['A', 'B', 'C', 'D', 'E', 'F', 'G']

neighbors = {}

neighbors['A'] = ['B', 'C', 'D']

neighbors['B'] = ['A', 'C']

neighbors['C'] = ['A', 'B', 'D', 'E']

neighbors['D'] = ['A', 'C', 'F', 'E']

neighbors['E'] = ['F', 'C', 'D']

neighbors['F'] = ['E', 'D', 'G']

neighbors['G'] = ['F']

# Output

colors\_of\_states = {}

def print\_graph(self):

for key in self.neighbors:

print(Fore.CYAN+ key + Fore.WHITE + ' -> ', self.neighbors[key])

def promising(self, state, color):

for neighbor in self.neighbors.get(state):

color\_of\_neighbor = self.colors\_of\_states.get(neighbor)

if color\_of\_neighbor == color:

return False

return True

def get\_color\_for\_state(self, state):

for color in self.colors:

if self.promising(state, color):

return color

def start(self):

print(Fore.BLUE+"\n\n\t\tThe Graph Is ")

self.print\_graph()

print("\n\n")

for state in self.states:

self.colors\_of\_states[state] = self.get\_color\_for\_state(state)

print(f"Color Used For State {state} is {self.colors\_of\_states[state]}")

print(Fore.BLUE+"\n\n\t\tThe Solution Is - ")

for key in self.colors\_of\_states:

print(Fore.BLUE+key + Fore.WHITE+' -> ', self.colors\_of\_states[key])

temp = map\_coloring()

temp.start()

1. Cryptharithmetic

from colorama import Fore, Back, Style, init

init(strip=False)

init(autoreset=True)

class cryptarithmetic():

solved = False

count = 0

def start(self):

word1 = input("Enter First Word - ").upper()

word2 = input("Enter Second Word - ").upper()

result = input("Enter Result - ").upper()

values = []

visited = [False for x in range(10)]

equation = [word1, word2, result]

# Get Unique Words

set = []

for c in word1:

if c not in set:

set.append(c)

for c in word2:

if c not in set:

set.append(c)

for c in result:

if c not in set:

set.append(c)

if len(set) > 10:

print("\nNo Solution (as values will repeat)\n")

exit()

print("Solution Is - ")

print(f" \t{word1}\n+\t{word2}\n-------------\n\t{result}\n\n")

self.solve(set, values, visited, equation)

def solve(self, letters, values, visited, equation):

if len(letters) == len(values):

map = {}

for letter, val in zip(letters, values):

map[letter] = val

if map[equation[0][0]] == 0 or map[equation[1][0]] == 0 or map[equation[2][0]] == 0:

return

word1, word2, res = "", "", ""

for c in equation[0]:

word1 += str(map[c])

for c in equation[1]:

word2 += str(map[c])

for c in equation[2]:

res += str(map[c])

if int(word1) + int(word2) == int(res):

self.count += 1

print(Fore.GREEN+f"Result {self.count} = {word1} + {word2} = {res}\n")

solved = True

return

for i in range(10):

if not visited[i]:

visited[i] = True

values.append(i)

self.solve(letters, values, visited, equation)

values.pop()

visited[i] = False

temp = cryptarithmetic()

temp.start()

1. Crossword

from typing import List

def check\_right(i, j, grid) -> tuple[int, int, int]:

counter = 0

while (counter + j) < len(grid[i]):

if grid[i][j + counter] == ' ':

counter += 1

else:

break

if counter < 2:

return None

else:

return (i, j, counter)

def check\_down(i, j, grid) -> tuple[int, int, int]:

counter = 0

while (counter + i) < len(grid):

if grid[i + counter][j] == ' ':

counter += 1

else:

break

if counter < 2:

return None

else:

return (i, j, counter)

def get\_across\_slots(grid: list[str]):

accross\_slots = []

i = 0

while i < len(grid):

j = 0

while j < len(grid[i]):

if grid[i][j] == ' ':

if slot := check\_right(i, j, grid):

accross\_slots.append(slot)

j += slot[2]

j += 1

i += 1

return accross\_slots

def get\_down\_slots(grid: list[str]):

t\_grid = []

# Get transpose of grid

for i in range(len(grid)):

string = ''.join([row[i] for row in grid])

t\_grid.append(string)

down\_slots = get\_across\_slots(t\_grid)

# The down slots are for the transposed grid,

# so we need to convert them to our original grid's coordinates

down\_slots = [(slot[1], slot[0], slot[2]) for slot in down\_slots]

return down\_slots

def start(across\_words: list[str], down\_words: list[str], grid: list[str]) -> list[str]:

across\_slots = get\_across\_slots(grid)

down\_slots = get\_down\_slots(grid)

# We need a mutable grid, so we use list[list[str]]

mut\_grid = []

for i in range(len(grid)):

arr = []

for j in range(len(grid[i])):

arr.append([grid[i][j]])

mut\_grid.append(arr)

# Start filling the across words

i = 0

while len(across\_words):

used = False

if used:

across\_slots.pop(i)

else:

i = (i + 1) % len(down\_slots)

if len(across\_words[0]) == across\_slots[i][2]:

x, y, \_ = across\_slots[i]

for counter, letter in enumerate(across\_words[0]):

mut\_grid[x][y + counter] = [letter]

else:

used = True

across\_words.pop(0)

# Start filling the down words

i = 0

while len(down\_words):

used = False

if used:

down\_slots.pop(i)

else:

i = (i + 1) % len(down\_slots)

if len(down\_words[0]) == down\_slots[i][2]:

x, y, \_ = down\_slots[i]

for counter, letter in enumerate(down\_words[0]):

mut\_grid[x + counter][y] = [letter]

else:

used = True

down\_words.pop(0)

# Convert list[list[str]] to list[str]

grid = []

for i in range(len(mut\_grid)):

string = ""

for j in range(len(mut\_grid[i])):

for k in range(len(mut\_grid[i][j])):

string += mut\_grid[i][j][k][0]

grid.append(string)

return grid

def display\_grid(grid: list[str]) -> None:

for row in grid:

for col in row:

print(f"{col:>2}", end='')

print()

def main():

ACROSS = ['HYBRID', 'EARTH']

DOWN = ['BREAD', 'HELMET']

grid = [

"########",

"# #",

"# # ####",

"# # ",

"# # ####",

"# # ####",

"# ######",

]

print(" The Initial Crossword is ".center(40, '='))

display\_grid(grid)

print("The Across Words Are : ", ', '.join(ACROSS))

print("The Down Words Are : ", ', '.join(DOWN))

result = start(ACROSS, DOWN, grid)

print('\n\n')

print(" The Final Crossword is ".center(40, '='))

display\_grid(result)

main()