AI Lab-7

190031187
Radhakrishna

prelab

1. Solve the csp problem

TWO + TWO = FOUR

$$\begin{array}{r} TWO \\ TWO \\ \hline FOUR \end{array}$$

→ F has to be 1, which also means that $T \geq 5$

→ The value of T depends on 'O'. So, if we look at the value of O

→ If O=0 the R would be 0, which doesn't work and 'O' can't be 1 because already F = 1

→ If O=2, then R=4 and since O=2, T must be equal to 6, so that $W < 5$ because there shouldn't be carry to T which changes the value of 'O'

^{1}	^{6} T	W	O^{2}
	^{6} T	W	O^{2}
F^{1}	O^{2}	U	R^{4}

→ So the only possible value of W is 3 which implies the value of U = 6 But already T=6

So, O=2 doesn't work

→ If O=3 then R=6, which forces the value of 'T' to be 6.

So, O=3 doesn't work

→ If O=4 then R=8

	T⁷	W³	O⁴
	T⁷	W³	O⁴
F¹	O⁴	U⁶	R⁸

Since O=4, T=7, so that W<5 because there shouldn't be carry to T

→ so possible values of W are 0, 2, 3

W cannot be 0 because then U becomes 0

If W=2 U=4 but already O=4

so W≠2

If W=3 U=6 which works

T=7, W=3, O=4, F=1, U=6, R=8

734 + 734 = 1468

→ If O=5 then R=0

So T should be 7 and it should get carry which implies that W≥6

If W=6, U=3 since there is a carry from 'O' which works

765 + 765 = 1530

W can't be 7 because T is already 7

W can't be 8 because U becomes 7 but T is already 7

W can't be 9 which results U = 9

→ If O = 6 then R = 2 and T = 8 and W should be < 5 because there can't be carry to T. So W could be 0, 3 or 4

If W = 0, U = 1, which doesn't work because F is already 1

If W = 3, U = 7 which works

836 + 836 = 1672

→ If W = 4, then U = 9 which works

846 + 846 = 1692

T⁸	W³	O⁶
T⁸	W³	O⁶
F¹	O⁶	U⁷	R²

T⁸	W⁴	O⁶
T⁸	W⁴	O⁶
F¹	O⁶	U⁹	R²

→ If O = 7 then R = 4 and T = 8 and W should be ≥ 5 because there has to be carry

If W = 6 U = 3 which works

867 + 867 = 1734

If W = 9 then U = 9 which doesn't work

→ If O = 8 then R = 6 and T = 9 so that W < 5 because there should not be any carry so W could be 0, 2, 3, 4

If W = 0 then U = 1 doesn't work (U = F)

If W = 2 then U = 5 which works

928 + 928 = 1856

If W = 3 then U = 7 which works

938 + 938 = 1876

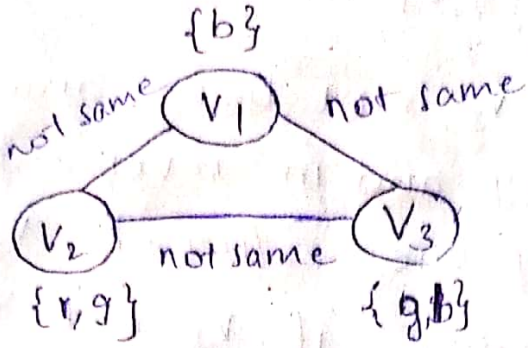
If W = 4 then U = 9 doesn't work because T = 9

→ If O = 9 then R = 8 but T should be 9 which doesn't work

so 7 possible answers are

938 + 938 = 1876
928 + 928 = 1856
867 + 867 = 1734
846 + 846 = 1692
836 + 836 = 1672
765 + 765 = 1530
734 + 734 = 1468

2. Apply ac1 for following



Initial domains:

$V_1 = \{b\}$

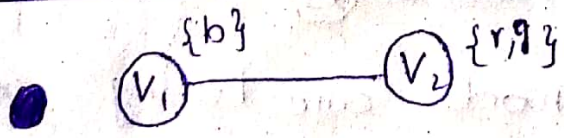
$V_2 = \{r, g\}$

$V_3 = \{g, b\}$

Each undirected constraint are is really two directed constraint arcs.

Constraint : The graph is consistent if and only if there should be atleast one color to a node and no two connected nodes have same colors

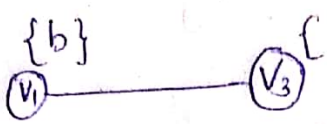
Arc Examined : $V_1 - V_2$



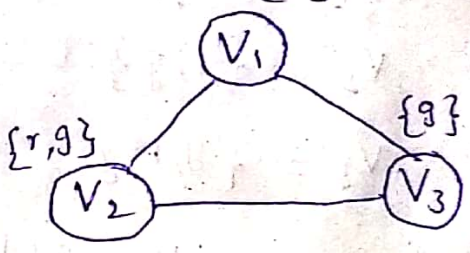
if V_1 is b then V_2 can be any color (r or g) and in reverse case since V_1 is having only b in its domain. The arc is consistent for any value of domain of V_2

Therefore no value is deleted.

Arc Examined : $V_1 - V_3$



if V_1 is b then V_3 must be g and there is no inconsistency and in reverse case, since V_1 is having only one value 'b' V_3 can be g but not b because if V_3 is 'b' then arc is inconsistent (since V_1 is also b). so the value b is deleted from domain of V_3 so that the arc is consistent



Arc Examined: $V_2 - V_3$

if V_2 is r, then arc is consistent since V_3 is having only value that is g. the reverse case since V_3 is g then V_2 must be r and the arc is inconsistent if V_2=g so V_2(g) is deleted to make arc consistent

Therefore all the nodes are having one colour and no two nodes have same colour so the graph is consistent

Arc Examined	Value Deleted
$V_1 - V_2$	None
$V_1 - V_3$	V_3(b)
$V_2 - V_3$	V_2(g)

INLAB

1. You have to color a map with different colors where no two neighboring regions can have the same color. Assume that you have 5 colors red, blue, green, yellow and pink.Write an efficient python code to color the regions in the following map.

Sample Output:

```
input
{'Madhya Pradesh': 'Red', 'Andhra Pradesh': 'Red', 'Kerala': 'Red', 'Odisha': 'Blue',
 'Telengana': 'Green', 'TamilNadu': 'Green', 'Chhattisgarh': 'Yellow', 'Maharashtra':
 'Pink', 'Karnataka': 'Blue'}


...Program finished with exit code 0
Press ENTER to exit console.
```

Code:

```
colors=["Red","Blue","Green","Yellow","Pink"]

states=['AndhraPradesh','Karnataka','Tamilnadu','Kerala','MadhyaPradesh','Odisha','Telangana','Chhattisgarh','Maharastra']

neighbours={}

neighbours['AndhraPradesh']=['Karnataka','Tamilnadu','Odisha','Telangana']
neighbours['Karnataka']=['AndhraPradesh','Tamilnadu','Kerala','Telangana','Maharastra']
neighbours['Tamilnadu']=['AndhraPradesh','Karnataka','Kerala']
neighbours['Kerala']=['Karnataka','Tamilnadu']
neighbours['Telangana']=['Karnataka','Odisha','Chhattisgarh','Maharastra','AndhraPradesh']
neighbours['Maharastra']=['MadhyaPradesh','Karnataka','Chhattisgarh','Telangana']
neighbours['MadhyaPradesh']=['Maharastra','Chhattisgarh']
neighbours['Chhattisgarh']=['Maharastra','Odisha','Telangana','MadhyaPradesh']
neighbours['Odisha']=['Chhattisgarh','AndhraPradesh','Telangana']


colors_of_states={}
```

```
def promising(state,color):
  for neighbour in neighbours.get(state):
    color_of_neighbour = colors_of_states.get(neighbour)
    if color_of_neighbour==color:
      return False

  return True
```

```
def get_color_for_state(state):
  for color in colors:
    if(promising(state,color)):
      return color
```

```
def main():
  for state in states:
    colors_of_states[state]=get_color_for_state(state)

  print(colors_of_states)
```

```
main()
```

```
{'AndhraPradesh': 'Red', 'Karnataka': 'Blue', 'Tamilnadu': 'Green', 'Kerala': 'Red', 'MadhyaPradesh': 'Red', 'Odisha': 'Blue', 'Telangana': 'Green', 'Chhattisgarh': 'Yellow', 'Maharastra': 'Pink'}
```

Code:

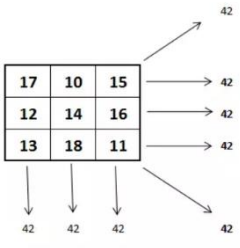
```
colors=["Red","Blue","Green","Yellow","Pink"]

states=['AndhraPradesh','Karnataka','Tamilnadu','Kerala','MadhyaPradesh
','Odisha','Telangana','Chhattisgarh','Maharastra']

neighbours={}

neighbours['AndhraPradesh']=['Karnataka','Tamilnadu','Odisha','Telangan
a']
neighbours['Karnataka']=['AndhraPradesh','Tamilnadu','Kerala','Telangan
a','Maharastra']
neighbours['Tamilnadu']=['AndhraPradesh','Karnataka','Kerala']
neighbours['Kerala']=['Karnataka','Tamilnadu']
neighbours['Telangana']=['Karnataka','Odisha','Chhattisgarh','Maharastr
a','AndhraPradesh']
neighbours['Maharastra']=['MadhyaPradesh','Karnataka','Chhattisgarh','T
elangana']
neighbours['MadhyaPradesh']=['Maharastra','Chhattisgarh']
neighbours['Chhattisgarh']=['Maharastra','Odisha','Telangana','MadhyaPr
adesh']
neighbours['Odisha']=['Chhattisgarh','AndhraPradesh','Telangana']


colors_of_states={}
def promising(state,color):
  for neighbour in neighbours.get(state):
    color_of_neighbour = colors_of_states.get(neighbour)
    if color_of_neighbour==color:
      return False
  return True

def get_color_for_state(state):
  for color in colors:
    if(promising(state,color)):
      return color

def main():
  for state in states:
    colors_of_states[state]=get_color_for_state(state)

  print(colors_of_states)

main()
```

POSTLAB

1. Solve the following problem using Constraint Satisfaction Problems (CSP):

Test Case 1: Magic Square ([[10,11,12], [13, 14, 15], [16, 17, 18]])
False, this is not a magic square. The numbers in the rows/columns/diagonals do not add up to the same value. Let's try another list of lists.

Test Case 2: Magic Square ([[17,10,15],[12,14,16],[13,18,11]])
True



• Develop a python program that satisfies below operations.

Code:

PostLab7.ipynb

File Edit View Insert Runtime Tools Help Last edited on October 1

Comment Share

+ Code + Text

Connect Editing

```
def isMagicSquare(mat) :
    s = 0
    for i in range(0,R):
        s = s + mat[i][i]

    s2 = 0
    for i in range(0,R):
        s2 = s2 + mat[i][R-i-1]
    if(s!=s2):
        return False

    for i in range(0,R):
        rowSum = 0;
        for j in range(0,R):
            rowSum += mat[i][j]
        if(rowSum != s):
            return False

    for i in range(0,R):
        colSum = 0
        for j in range(0,R):
            colSum += mat[j][i]
        if(s != colSum):
            return False
    return True
```

```
R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))

# Initialize matrix
mat = []
print("Enter the entries rowwise:")

# For user input
for i in range(R):      # A for loop for row entries
  a =[]
  for j in range(C):   # A for loop for column entries
    a.append(int(input()))
  mat.append(a)

if(isMagicSquare(mat)) :
    print("Magic Square")
else :
    print("Not a magic Square")
```

```
Enter the number of rows:3
Enter the number of columns:3
Enter the entries rowwise:
1
3
6
7
2
5
8
9
7
Not a magic Square
```

Code:

```
def isMagicSquare(mat) :
    s = 0
    for i in range(0,R):
        s = s + mat[i][i]

    s2 = 0
    for i in range(0,R):
        s2 = s2 + mat[i][R-i-1]
    if(s!=s2):
        return False

    for i in range(0,R):
        rowSum = 0;
        for j in range(0,R):
            rowSum += mat[i][j]
        if(rowSum != s):
            return False

    for i in range(0,R):
        colSum = 0
        for j in range(0,R):
            colSum += mat[j][i]
        if(s != colSum):
            return False
    return True


R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))

# Initialize matrix
mat = []
print("Enter the entries rowwise:")

# For user input
for i in range(R):      # A for loop for row entries
  a =[]
  for j in range(C):    # A for loop for column entries
    a.append(int(input()))
  mat.append(a)

if(isMagicSquare(mat)) :
    print("Magic Square")
else :
    print("Not a magic Square")
```