```
    ans = []
    for rr in R:
        res = checkRestriction(rr, province, color)
        if res == False:
            return False
        elif res == None:
            continue
        else:
            ans.append(res)
    return ans

# checks if the restrition rr allows the given province to have the given color
# returns false if not possible, otherwise returns the new restriction
def checkRestriction(rr, province, color):
    #finding the index of the province (saved to index)
    index = -1
    other = -1
    if rr[0] == province:
        index = 0
        other = 1
    elif rr[1] == province:
        index = 1
        other = 0
    else:
        return rr

    if isinstance(rr[other], int):
        # other component is a color
        if (color != rr[other]):
            return None
        else:
            return False
    else:
        return [rr[other], color]


# solving the CSP by variable elimination
# recursive structure: ci is the province index to be colored (0 = bc, 1 = ab, etc)
# n is the number of colors
# provinces is a list of provinces
```

```
File  Edit  Shell  Debug  Options  Window  Help
>>>
========== RESTART: C:/Users/siris/8 PUZZLE PROBLEM.py ==========
Traceback (most recent call last):
  File "C:/Users/siris/8 PUZZLE PROBLEM.py", line 78, in <module>
    if _name_ == '_main_':
NameError: name '_name_' is not defined. Did you mean: '__name__'?
>>>
========== RESTART: C:/Users/siris/CSP.py ==========
Enter number of the color? RED
Traceback (most recent call last):
  File "C:/Users/siris/CSP.py", line 126, in <module>
    num=int(input("Enter number of the color? "))
ValueError: invalid literal for int() with base 10: 'RED'
>>>
========== RESTART: C:/Users/siris/CSP.py ==========
Enter number of the color? 6
{'ab': 1, 'bc': 2, 'mb': 1, 'nb': 1, 'ns': 2, 'nl': 1, 'nt': 3, 'nu': 2
, 'on': 2, 'pe': 3, 'qc': 3, 'sk': 2, 'yt': 1}
Enter number of the color? 7
{'ab': 1, 'bc': 2, 'mb': 1, 'nb': 1, 'ns': 2, 'nl': 1, 'nt': 3, 'nu': 2
, 'on': 2, 'pe': 3, 'qc': 3, 'sk': 2, 'yt': 1}
Enter number of the color? 8
{'ab': 1, 'bc': 2, 'mb': 1, 'nb': 1, 'ns': 2, 'nl': 1, 'nt': 3, 'nu': 2
, 'on': 2, 'pe': 3, 'qc': 3, 'sk': 2, 'yt': 1}
Enter number of the color? 8
{'ab': 1, 'bc': 2, 'mb': 1, 'nb': 1, 'ns': 2, 'nl': 1, 'nt': 3, 'nu': 2
, 'on': 2, 'pe': 3, 'qc': 3, 'sk': 2, 'yt': 1}
Enter number of the color? 9
{'ab': 1, 'bc': 2, 'mb': 1, 'nb': 1, 'ns': 2, 'nl': 1, 'nt': 3, 'nu': 2
, 'on': 2, 'pe': 3, 'qc': 3, 'sk': 2, 'yt': 1}
Enter number of the color? 1
False
Enter number of the color? 2
False
Enter number of the color? 4
{'ab': 1, 'bc': 2, 'mb': 1, 'nb': 1, 'ns': 2, 'nl': 1, 'nt': 3, 'nu': 2
, 'on': 2, 'pe': 3, 'qc': 3, 'sk': 2, 'yt': 1}
Enter number of the color?
========== RESTART: C:/Users/siris/CSP.py ==========
```

```
def calculateCosts(mats, finals) -> int:

    count = 0
    for i in range(n):
        for j in range(n):
            if ((mats[i][j]) and
                (mats[i][j] != final[i][j])):
                count += 1

    return count

def newNodes(mats, empty_tile_posi, new_empty_tile_posi,
             levels, parent, final) -> nodes:

    # Copying data from the parent matrixes to the present matrixes
    new_mats = copy.deepcopy(mats)

    # Moving the tile by 1 position
    x1 = empty_tile_posi[0]
    y1 = empty_tile_posi[1]
    x2 = new_empty_tile_posi[0]
    y2 = new_empty_tile_posi[1]
    new_mats[x1][y1], new_mats[x2][y2] = new_mats[x2][y2], new_mats[x1][y1]

    # Setting the no. of misplaced tiles
    costs = calculateCosts(new_mats, final)

    new_nodes = nodes(parent, new_mats, new_empty_tile_posi,
                      costs, levels)
    return new_nodes

# func to print the N by N matrix
def printMatsrix(mats):

    for i in range(n):
        for j in range(n):
            print("%d " % (mats[i][j]), end = " ")

        print()

# func to know if (x, y) is a valid or invalid
```

```
File  Edit  Shell  Debug  Options  Window  Help
5  8  6
0  7  4

>>>

========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

1  2  3
5  8  6
0  7  4

>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

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

```
            return

        # Generating all feasible children
        for i in range(n):
            new_tile_posi = [
                minimum.empty_tile_posi[0] + rows[i],
                minimum.empty_tile_posi[1] + cols[i], ]

            if isSafe(new_tile_posi[0], new_tile_posi[1]):

                # Creating a child node
                child = newNodes(minimum.mats,
                                 minimum.empty_tile_posi,
                                 new_tile_posi,
                                 minimum.levels + 1,
                                 minimum, final,)

                # Adding the child to the list of live nodes
                pq.push(child)

# Main Code

# Initial configuration
# Value 0 is taken here as an empty space
initial = [ [ 1, 2, 3 ],
            [ 5, 6, 0 ],
            [ 7, 8, 4 ] ]

# Final configuration that can be solved
# Value 0 is taken as an empty space
final = [ [ 1, 2, 3 ],
          [ 5, 8, 6 ],
          [ 0, 7, 4 ] ]

# Blank tile coordinates in the
# initial configuration
empty_tile_posi = [ 1, 2 ]

# Method call for solving the puzzle
solve(initial, empty_tile_posi, final)
```

```
File  Edit  Shell  Debug  Options  Window  Help
5  8  6
0  7  4

>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

1  2  3
5  8  6
0  7  4

>>>
========== RESTART: C:/Users/siris/QUEEN PROBLEM.py ==========
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

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