Print of the code:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  4 14:18:24 2018

@author: Deepak
"""

edges=[]
with open('case1.txt') as f:
    array = [[int(x) for x in line.split()] for line in f]

line1=array[0]
Actual_N=line1[0]
Actual_edges=[1]
C=line1[2]
for i in range(len(array)-1):
    edges.append(array[i+1])

global count
global fringe

N=max(sum(edges, []))
E=len(edges)
color=range(C)
initial_color=[]

for i in range(C):
        initial_color.append(i)
count=0
answer=[]
total_nodes=[]
removed_nodes=[]
check=0

# SEPARATING EDGES FOR EACH NODE
def extract_edges(edges,N):
    arr=[]
    arr1=[]
    ind_edges=[]
    for k in range(N):
        ind_edges.append([])
        for i in range(len(edges)):
            for j in [0,1]:
                if (edges[i][j]==k):
                    ind_edges[k].append(edges[i][:])

    for i in range(len(ind_edges)):
        arr.append([])
        for j in ind_edges[i]:
            for k in j:
                arr[i].append(k)
    for ele in arr:
        arr1.append(list(set(ele)))

    arr1=arr1[1:len(arr1)]

    return(arr1)

array=extract_edges(edges,N+1)

N1=len(array)


class Node:
    def __init__( self, state, parent, action, depth, cost,color ):
        self.state = state
        self.parent = parent
        self.action = action   # action that created this node
        self.depth = depth
        self.cost = cost
        self.color = color


# CREATES A NODE
def create_node(state, parent, action, depth, cost, color):
    return Node(state, parent, action, depth, cost, color)

# CREATE N NODES WITH STATE NUMBERS BUT EMPTY COLORS
for i in range(N):
    total_nodes.append(create_node(i+1,None,None,0,0,[]))


# BREADTH FIRST SEARCH
def bfs( start ):
    check=0
    print('Start state:',start)
    nodes = []
```

```
   dep=0
   nodes.append( create_node( start, None, None, 0, 0, 0 ) )
   total_nodes[start-1].color=0
   last_node=[start]
   last_edge=[array[start-1]]
   last_color=[0]
   while True:
      if len( nodes ) == 0: return None
      node = nodes.pop(0)
      new_node,check,last_node,last_edge,last_color,dep=expand_node( node,
nodes,array[node.state-1],check,last_node,last_edge,last_color,dep)
      nodes.extend( new_node )

      global count, fringe
      count=count+1
      fringe=len(nodes)
      print('number of states in the fringe = ',fringe)

      if check>10:            # if not able to assign color for last 10 times
         print('FAILED !!!')
         break

# DEPTH FIRST SEARCH
def dfs( start,depth_limit=N):
   print('DEPTH_LIMIT IN DFS IS ',depth_limit)
   depth_limit=depth_limit+1     # 0 to N ==> 1 to N+1
   kk=0
   check=0
   print('Start state:',start)
   nodes = []
   dep=0
   nodes.append( create_node( start, None, None, 0, 0, 0 ) )
   total_nodes[start-1].color=0
   last_node=[start]
   last_edge=[array[start-1]]
   last_color=[0]
   while True:
      if len( nodes ) == 0:
         print('\n\nFRINGE WENT EMPTY !!')
         return None
      node = nodes.pop(0)

      if dep < depth_limit:
         expanded_nodes,check,last_node,last_edge,last_color,dep=expand_node( node,
nodes,array[node.state-1],check,last_node,last_edge,last_color,dep+1)
         expanded_nodes.extend( nodes )
         nodes = expanded_nodes
         global count, fringe
         count=count+1
         fringe=len(nodes)
         kk=0
         for i in total_nodes:
             if i.color!=[]:
                 kk=kk+1

         if check>10:                # if not able to assign color for last 10 times
             print('FAILED !!!')
             break

         if dep==depth_limit:
             print('DEPTH REACHED!!')
             break


# ITERATIVE DEEPENING SEARCH
def ids( start, depth ):
   for ii in range(depth):      # 0 to N ==> 1 to N+1
      print('\n\nCALLING DFS WITH DEPTH ',ii)
      dfs( start, ii )


# A-STAR SEARCH
def a_star( start ):
   nodes = []
   mincost=0
   index=0
   y=[]
   dep=0
   check=0
   last_node=[start]
   last_edge=[array[start-1]]
   last_color=[0]

   nodes.append( create_node( start, None, None, 0, 0, 0 ) )
   while True:
      if len( nodes ) == 0: return None
      for i in range (len(nodes)):
         y=nodes[i]
         cost = Huer(y)
         if i==0:
             mincost=cost
             index=i
         elif mincost>cost:
             mincost=cost
             index=i
      node = nodes.pop(index)
      new_node,check,last_node,last_edge,last_color,dep=expand_node( node,
nodes,array[node.state-1],check,last_node,last_edge,last_color,dep)
      nodes.extend( new_node )
      global count, fringe
      count=count+1
      fringe=len(nodes)

      if check>10:                # if not able to assign color for last 10 times
         print('FAILED !!!')
         break

global red, green, blue
red=0
green=0
blue=0
# MINIMUM NUMBER OF EDGES AS HEURISTIC
def Huer(x):
   score1 =0
   score=0
   global red, green, blue
   if x.color==0:
      red=red+1
      score1=red
   elif x.color==1:
      green=green+1
      score1=green
   elif x.color==2:
      blue=blue+1
      score1=blue

   score=-len(array[x.state-1])-score1  # negative value as minimum number of edges
   return score


def recolor(node,edges,color):
   print('inside recolor')
   temp_col=initial_color

   for i in edges:
      temp_col=initial_color
      edges_i=array[i-1]
      for j in edges_i:
         iedge_col=total_nodes[j-1].color
         if iedge_col in temp_col:

             temp_col.remove(iedge_col)
      if len(temp_col)==1:
         total_nodes[i-1].color=temp_col.pop(0)
         print(total_nodes[i-1].state,' node is now getting ',total_nodes[i-1].color,' color')
         print()

# EXPANDS THE NODES
def expand_node( node, nodes,edges,check,last_node,last_edge,last_color,dep):

   print('State selected for expansion ',node.state)
   expanded_nodes = []
   temp=[]
   for i in range(C):
      temp.append(i)

   if node not in removed_nodes:          # to avoid looping
      temp=check_color(node,temp,edges)
      if temp==[]:       # REASSIGNING THE COLOR
         check=check+1
         print('\nGoing to recolor node ',last_node.state,' with color ', last_node.color)
         recolor(last_node,last_edge,last_color)
      if temp !=[]:
         total_nodes[node.state-1].color=temp.pop(0)
      for i in edges:
         expanded_nodes.append(total_nodes[i-1])
      expanded_nodes = [node for node in expanded_nodes if node.color != None]

      last_node=total_nodes[node.state-1]
      last_edge=edges
      last_color=node.color

      removed_nodes.append(node)
      node.depth=dep
      print('node ',node.state,'added with color ',node.color)
   return expanded_nodes, check, last_node, last_edge,last_color,node.depth


def check_color(node,temp,edges):
   for a in edges:
      col1=total_nodes[a-1].color
      if (col1 in temp) and node.state!=a:
         temp.remove(col1)
   return temp



def main():
   start_state=1

   #bfs( start_state )          # BREADTH FIRST SEARCH
   #dfs(start_state)           # DEPTH FIRST SEARCH
   #ids(start_state,10)        # IDS SEARCH
   a_star(start_state)        # A_STAR


   # PRINTING ANSWERS
   print('\n**************************************************')
   print('FINAL ANSWER')
   print('**************************************************')
   if Actual_N>N:
      print('\n0=RED, 1=GREEN, 2=BLUE')
      print('\nNOTE: If any state is not in FINAL ANSWER, then it is an island and can take any
color')
   for i in range(len(total_nodes)):
      print('state = ',total_nodes[i].state,' color = ',total_nodes[i].color)
   print('\nnumber of states expanded:',count)
   print('number of states in the fringe: ',fringe)
   print('**************************************************')

if __name__ == "__main__":
main()
```