Advance Algorithm from Q6

The average color that is required from 1000 runs with n = 2000 and P = 0.02 is 15.635

The average color that is required from 1000 runs with n = 2000 and P = 0.01 is 11.45

The average color that is required from 1000 runs with n = 2000 and P = 0.006 is 8.92

The average color that is required from 1000 runs with n = 2000 and P = 0.002 is 5.551

For 1000 runs average result with n = 2000 with different random graph each time

P value	Greedy algorithms from Q5	Advance Algorithm from Q6	diff
0.02	17.131	15.635	1.496
0.01	11.509	10.15	1.359
0.006	8.944	7.856	1.088
0.002	5.618	4.864	0.754

For 100 run result with the same random graph each time

P value	Greedy algorithms from Q5	Advance Algorithm from Q6	Max degree
0.02	17.2	15.7	62.8
0.01	11.48	10.15	37.06
0.006	8.94	7.85	25.44
0.002	5.6	5.5514.9	12.38

This algorithm is only slightly faster than the greedy algorithms since the Advance Algorithm from Q6 is an iterative greedy algorithm. It can save 1.5 colors on average with p = 0.02 and n = 2000. This number of saving is decreasing when the p is decreasing. Therefore, the more complicated a graph is, the Advance Algorithm from Q6 can save more color

Code:

```
import randomG
import operator
import math
import copy
import os
c = 100
n = 2000
p = 0.006
sameGraph = False
def gui(i):
    os.system("cls")
    print("                " + str(i * (100/c)) + "%")
    bar = math.floor(i / (c/10))
    for j in range(bar):
        print("■",end = "")
    for j in range(10 - bar):
        print("□",end = "")
    print()
```

```
def main():
    sum1 = 0
    sum2 = 0
    sum3 = 0
    for i in range(1, c + 1):
        gui(i)
        graph = randomG.createGraph(p, n)
        sum1 = sum1 + key(graph)
        sum2 = sum2 + keyQ5(graph)
        maxDegree = 0
        for i, j in graph.items():
            maxDegree = max(maxDegree, len(j))
        sum3 = sum3 + maxDegree
    print("The average color that is required from " + str(c) + " runs "
            "with n = " + str(n) + " and P = " + str(p))
    print("average advance coloring number is " + str(sum1*1.0/c))
    print("average greedy coloring number from q5 is " + str(sum2*1.0/c))
    print("average max degree is " + str(sum3*1.0/c))
def getSortedV(graph):
    temp = {}
    for i, j in graph.items():
        temp[i] = len(j)
    temp2 = sorted(temp.items(), key=operator.itemgetter(1))
    result = []
    for i in temp2:
        result.append(i[0])
    return result
def key(graph):
    allV = list(range(1,n+1))
    rank = getSortedV(graph)
    colored = {}
    index = 0
    while len(colored) != n:
        v = rank.pop(-1)
        if v not in colored:
            colored[v] = index
            allPossibleV = set(allV) - set(graph[v]) - colored.keys()
            for i in range(len(rank) - 1, -1, -1):
                j = rank[i]
                if (j in allPossibleV) and check(graph, index, j, colored):
                    colored[j] = index
                    rank.remove(j)
        index += 1
    return index
```

```
def check(graph, curColor, curV, colored):
    for i in graph[curV]:
        if (i in colored) and (colored[i] == curColor):
            return False
    return True
def sameGraph():
    print("Same Graph")
    print("advance algorithm: " + str(key()))
    print("greedy algorithm from Q5: " + str(color.key))

def keyQ5(graph):
    colorlist = [0]
    colored = {}
    for U, neighbors in graph.items():
        tempColor = copy.deepcopy(colorlist)
        for V in neighbors:
            if V in colored and colored[V] in tempColor:
                tempColor.remove(colored[V])
        if len(tempColor) == 0:
            newInt = len(colorlist)
            colorlist.append(len(colorlist))
            colored[U] = newInt
        else:
            colored[U] = tempColor[0]

    return len(colorlist)
main()
```