ΦΡΑΝΤΖΗΣ ΚΟΡΗΛΑΚΗΣ Π20095

Γενετικός αλγόριθμος χρωματισμός γράφων

Για την ανάπτυξη τυχαίου πληθυσμού έφτιαξα την συνάρτηση

```
def color_graph(graph, colors):
```

η οποία δέχεται τις συνδέσεις του γραφήματος και τα χρώματα που είναι διαθέσιμα και επιστρέφει ένα τυχαίο χρωματισμό ,την συνάρτηση την καλώ 40 φορες αρα ξεκινάω με αρχικό πλύσιμο 40

για συνάρτηση καταληλοτηας εκανα την συνάρτηση score(parents)που ελέγχει κάθε σύνδεση στο γραφο και για κάθε γραφο που συνδέεται με άλλον εχει άλλο χρώμα αυξάνεται κατά ένα το σκορ καταληλοτηας για αυτό το γράφημα

για την επιλογή βάση σκορ χρησιμοποίησα την τεχνική ρουλέτας

για την διασταύρωση σημείου χώρισα τους γονείς σε ζευγάρια και από το κάθε ζευγάρι παίρνω δυο απογόνους

για την μετάλλαξη έβαλα να επηρεάζει το 40% του πληθυσμού όμως έβαλα να γίνεται ανα 20 γενιές ώστε να προλάβουν να γινουν αρκετές διασταυρωσεις

για τερματική συνθήκη έβαλα όταν βρεθεί χρωματισμός με σκορ 79 ή όταν υπερβεί τις 10000 επαναλήψεις παρακάτω παραθέτω αυτούσιο τον κώδικα και ένα παράδειγμα επιτυχούς εκτέλεσης

```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool
windows, actions, and settings.

import random

import random
import json
from copy import deepcopy
import numpy as np
def foo(some_var):
    some_var_ = deepcopy(some_var)

    return some_var_


def mutation(parents):

    mhkos=len(parents)
    colors =   ['red', 'blue', 'green', 'yellow']


    posst=mhkos*(40/100)
    xromsom= 4

    posst=int(posst)
```

```
    print(posst)
    print("mhkos",mhkos)


    if posst==0:
        posst=1

    for i in range(0,posst):
        person= random.randint(0,mhkos-1)
        xromsom=random.randint(0,3)
        pos=random_integer = np.random.randint(0,15)




        while parents[person][pos]==colors[xromsom]:
            xromsom = random.randint(0, 3)

        parents[person][pos]=colors[xromsom]



    return  parents
def onpointcut(parents):
    pairs=[]
    apogonoi=[]
    p2=[]


    for i in range(0, len(parents), 2):
        if i + 1 < len(parents):
         pairs.append([parents[i], parents[i + 1]])



    cp2=foo(pairs)


    for i in  pairs:

        first_dict = i[0]
        second_dict = i[1]
        midpoint = len(first_dict) // 2

        for i in range(midpoint, len(first_dict)):

            first_dict[i] = second_dict[i]

            if i==len(first_dict)-1:

                apogonoi.append(first_dict)



    for x in cp2:

        dict1=x[0]
        dict2=x[1]

        # Get the keys and values of the dictionaries as lists
        keys1 = list(dict1.keys())
```

```
        values1 = list(dict1.values())
        keys2 = list(dict2.keys())
        values2 = list(dict2.values())


        # Calculate the midpoint
        midpoint = len(keys1) // 2

        # Create a new dictionary containing the second half of the
first dictionary and the first half of the second dictionary
        new_dict = {}
        new_dict.update(dict(zip(keys2[:midpoint],
values2[:midpoint])))
        new_dict.update(dict(zip(keys1[midpoint:],
values1[midpoint:])))


        apogonoi.append(new_dict)
    #print("apogonoi",apogonoi)
    return apogonoi
def roulette_wheel_selection(population, fitness_scores):
    total_fitness = sum(fitness_scores)


    rand=random.randint(1,total_fitness)
    parents2 = []
    for i in range(len(population)):
        rand=random.randint(1,total_fitness)

        suma=0
        for x in range(len(population)):
               suma=suma+fitness_scores[x]


               if rand<=suma:


                   parents2.append(population[x])
                   break

    return parents2


def score(graph,vertex_colors):
    edge_count = 0
    for vertex in graph:
        for neighbor in graph[vertex]:


            if vertex_colors[vertex] != vertex_colors[neighbor]:
                edge_count += 1

    return  edge_count

def color_graph(graph, colors):
    # Initialize a dictionary to store the color of each vertex
    vertex_colors = {}

    # Assign a random color to each vertex
    for vertex in graph:
        vertex_colors[vertex] = random.choice(colors)
```

```
    # Initialize a counter for the number of edges with different-
colored vertices
    edge_count = 0


    # Check if any neighboring vertices have a different color
    for vertex in graph:
        for neighbor in graph[vertex]:
            if vertex_colors[vertex] != vertex_colors[neighbor]:
                edge_count += 1

    # Return the dictionary of vertex colors and edge count
    return vertex_colors, edge_count
# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    vertex_colors_list = []
    edge_count_list = []
    edge_count_list2=[]
    graph = {0: [1, 2, 3,15], 1: [0,2,4,7,15], 2: [0, 1, 3,4,5], 3:
[0,
2,5,15,12],4:[1,2,5,6,8,9],5:[4,2,3,6,10,12],6:[4,5,9,10],7:[1,8,13],
8:[1,4,7,9,13,11],9:[4,6,8,10,11],10:[6,9,5,11,12],11:[9,10,8,12,13,1
4],12:[10,11,0,5,3,14],13:[7,8,11,1,14],14:[13,11,12,0,15,1],15:[0,1,
15]}
    colors = ['blue', 'red', 'green', 'yellow']
    for i in range(40):
        vertex_colors, edge_count = color_graph(graph, colors)
        vertex_colors_list.append(vertex_colors)
        edge_count_list.append(edge_count)
    # Print the results
    for i in range(len(vertex_colors_list)):
        print(f"Result {i + 1}:")
        print(f"Vertex colors: {vertex_colors_list[i]}")
        print(f"Number of edges with different-colored vertices:
{edge_count_list[i]}\n")


    parents =
roulette_wheel_selection(vertex_colors_list,edge_count_list)

    parents = onpointcut(parents)

    parents=mutation(parents)

    edge_count_list2 = []
    ader=0
    found=False
    for i in range (10000):
        ader=ader+1


        for i in range(len(parents)):
            e = score(graph, parents[i])
            edge_count_list2.append(e)


        for v in range(len(parents)):
            print(edge_count_list2[v])
            if edge_count_list2[v]>=78:
                print("found",parents[v])
```

```
                found=  True
                break


        parents = roulette_wheel_selection(parents, edge_count_list2)


        parents = onpointcut(parents)

        if ader ==20:
            parents = mutation(parents)
            ader=0



        edge_count_list2 = []
        if found: break


#   print(parents)
```

```
main
75
77
75
75
77
77
79
found {0: 'yellow', 1: 'red', 2: 'green', 3: 'blue', 4: 'blue', 5: 'yellow', 6: 'red', 7: 'blue', 8: 'yellow', 9: 'green', 10: 'blue', 11: 'red', 12: 'green', 13: 'green', 14: 'blue', 15: 'green'}

Process finished with exit code 0
```