ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΉΣ

ΠΡΟΑΙΡΕΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΑΘΗΜΑ: «ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ»

ΔΙΔΑΣΚΩΝ: «ΑΠΟΣΤΟΛΟΥ ΔΗΜΗΤΡΗΣ»

ΦΟΙΤΗΤΡΙΑ: ΠΟΛΥΧΡΌΝΗ ΕΛΕΝΗ – ΧΡΙΣΤΙΝΑ

Α.Μ : Π20161

ΕΤΟΣ ΣΠΟΥΔΩΝ: Γ’

ΛΥΣΗ ΕΡΓΑΣΙΑΣ

ΔΗΜΙΟΥΡΓΙΑ ΠΛΗΘΥΣΜΟΥ

```
'''
Creating the population
'''
p = 20 # initial population
print("\nPopulation is: ", p)
colors = ['blue','red','green','yellow'] # 4 available colors
print("Available colors are: ", colors)

n = 16
dict = {}
for i in range(0,p):
    colors_list = []
    for j in range(0,n):
        colors_list.append(random.choice(colors))
    dict[i] = colors_list

print("\nRandom population in Dictionary: ")
for k, v in dict.items():
    print(k, v)
```

Θωρούμε ως **αρχικό πληθυσμό** τον αριθμό **20**. Σε ένα **λεξικό**, αποθηκεύουμε για την **κάθε οντότητα** του πληθυσμού μια λίστα με **16 χρώματα**. Τα διαθέσιμα χρώματα είναι **4**, οπότε είναι λογικό να παρατηρείται **επανάληψη** χρωμάτων στις λίστες.

ΣΥΝΑΡΤΗΣΗ ΚΑΤΑΛΛΗΛΟΤΗΤΑΣ

```
'''
Adjacency Matrix
'''
N = [[0,1,1,1,0,0,0,0,0,0,0,0,1,0,1,1],
     [1,0,1,0,1,0,0,1,1,0,0,0,0,1,1,1],
     [1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0],
     [1,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0],
     [0,1,1,0,0,1,1,0,1,1,0,0,0,0,0,0],
     [0,0,1,1,1,0,1,0,0,0,1,0,1,0,0,0],
     [0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0],
     [0,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0],
     [0,1,0,0,1,0,0,1,0,1,0,1,0,1,0,0],
     [0,0,0,0,1,0,1,0,1,0,1,1,0,0,0,0],
     [0,0,0,0,0,1,1,0,0,1,0,1,1,0,0,0],
     [0,0,0,0,0,0,0,0,1,1,1,0,1,1,1,0],
     [1,0,0,1,0,1,0,0,0,0,1,1,0,0,1,0],
     [0,1,0,0,0,0,0,1,1,0,0,1,0,0,1,0],
     [1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1],
     [1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]]
```

Για τις ανάγκες της εργασίας, κατασκευάζουμε την **μήτρα γειτνίασης** που φαίνεται στην διπλανή εικόνα, που θα μας βοηθήσει να υπολογίσουμε την **συνάρτηση καταλληλόλητας**.

```
def find_scores(dict):
    scores = {}
    for i in range(0,p): # N adjacency matrix row
        score = 0
        for j in range(0,16): # N adjacency matrix column
            for k in range(0,16): # dictionary
                if N[j][k] == 1:  # neighbor
                    if dict[i][j] != dict[i][k]: # different color neighbor
                        score+=1
                else:
                    continue
        scores[i] = score
    return(scores)
```

Μετράμε με την μορφή **score**, το πλήθος των **καλών συνδέσεων** για κάθε μία από τις 20 ακολουθίες.

Αυτό που κάνουμε στην πραγματικότητα, είναι να τσεκάρουμε για **κάθε ένα** από τα στοιχεία της λίστας της ακολουθίας, αν ο **γείτονας** του στοιχείου αυτού έχει **διαφορετικό** χρώμα από εκείνο, και αν ναι τότε **αυξάνουμε** την μεταβλητή **score** κατά **1**. Η **μήτρα γειτνίασης** φέρει τις πληροφορίες για τους **γείτονες** των στοιχείων. Σε ένα **λεξικό** επιστρέφονται οι **αριθμοί** των ακολουθιών μαζί με το αντίστοιχο **score** τους.

ΔΙΑΔΙΚΑΣΙΑ ΕΠΙΛΟΓΗΣ ΓΟΝΕΩΝ

ΤΕΧΝΙΚΗ ΤΗΣ ΡΟΥΛΕΤΑΣ (ROULETTE WHEEL SELECTION)

```python
def roulette_wheel_selection(scores, r): # with dictionary
    '''
    Selection
    Roulette-Wheel Selection for Partial Renewal
    '''

    # find the sum of dictionary's values
    print("Sum of all dictionary values is: ",sum(scores.values()))

    l = []
    for i in range(0,r):
        l.append(random.randint(0, sum(scores.values())))


    print("\nl is: ",l)
    print("\n")

    keys = list(scores.keys())
    values = list(scores.values())

    l1 = []
    for i in range(0,r): # l list
        j = 0
        sum1 = 0
        if l[i] <= values[j]:
            l1.append(keys[j])
        elif (l[i] > values[j]):
            sum1 = values[j] + values[j+1]

            j = 1
            while sum1 < l[i] and j < r-1:
                sum1 += values[j+1]
                j+=1
            l1.append(keys[j])

    # remove the duplicates elements
    res = [*set(l1)]

    return(res)
```

Εφαρμόζεται η **τεχνική της ρουλέτας** που αναφέρεται και στις διαφάνειες του μαθήματος.

Η συνάρτηση επιστρέφει μια **λίστα** με τους **αριθμούς** των υποψήφιων λύσεων που **προέκυψαν** (μέσω της ρουλέτας). Ουσιαστικά, η λίστα αυτή αποτελεί την λίστα των **υποψήφιων γονέων**.

```python
def parents_pairs(list1):
    '''
    All possible pairs in list
    Using combinations()
    '''
    l = list(combinations(list1, 2))
    return(l)
```

Η συνάρτηση αυτή, επιστρέφει όλους τους **δυνατούς συνδυασμούς** ανά **2** των στοιχείων μιας λίστας. Χρησιμοποιείται για να σχηματίσει τα **ζευγάρια των γονέων**.

ΑΝΑΠΑΡΑΓΩΓΗ ΜΕ ΔΙΑΣΤΑΥΡΩΣΗ ΕΝΟΣ ΣΗΜΕΙΟΥ (SINGLE POINT CROSSOVER)

```python
def single_point_crossover(pairs, dict):
    '''
    Crossover
    Single Point Crossover

    16 colors - > split the sequence in 8 and 8 colors
    '''

    num_of_children = len(pairs) * 2
    c = 16
    children_table = [[0] * c for _ in range(num_of_children)]

    l = 0
    for k in range(len(pairs)):
        t1 = dict[pairs[k][0]]
        t2 = dict[pairs[k][1]]
        children_table[l] = t1[:8] + t2[8:]
        children_table[l+1] = t2[:8] + t1[8:]
        l+=2

    return(children_table)
```

Κάθε **ζευγάρι γονέων** κάνει **2 παιδιά**.

1ο παιδί: Για κάθε ζευγαράκι γονιών παίρνουμε τα **πρώτα 8 στοιχεία** (χρώματα) από τον πρώτο γονιό και τα συνενώνουμε με τα **8 τελευταία** από τον άλλο γονιό.

2ο παιδί: Για κάθε ζευγαράκι γονιών παίρνουμε τα **πρώτα 8 στοιχεία** (χρώματα) από τον δεύτερο γονιό και τα συνενώνουμε με τα **8 τελευταία** από τον άλλο γονιό.

Η συνάρτηση επιστρέφει τον ανανεωμένο πίνακα με τα **παιδιά** (την νέα γενιά).

ΜΕΡΙΚΗ ΑΝΑΝΕΩΣΗ ΠΛΗΘΥΣΜΟΥ

Πραγματοποιούμε **μερική ανανέωση πληθυσμού** και συγκεκριμένα στο **30%** του πληθυσμού.

```python
    '''
    Merikh Ananewsh
    Merikh ananewsh plhthismoy -> 30% diastavrwsh
    '''
    r = 0.3 * p  # partial population e.g 30% population
    print("\nPartial popluation r is ",int(r))

    d = {}
    for i in range(int(r)):
        flag = False
        while flag == False:
            # choose them randomly
            k = random.randint(0, p-1) # generate a number between 0 and p-1 (both included)
            if k not in d.keys():
                v = scores_d[k]
                d[k] = v
                flag = True

    print("\nPartial scores in dict are: ",d)
    # aytoi tha ginoun goneis, tha kanoun paidia kai tha enwthoun me toys allous -> thn previous generation


    keysList = list(d.keys())
    print("Keys are: ",keysList)

    flag = False
    while (flag == False):
        parents = roulette_wheel_selection(d, int(r))
        if (len(parents) >= 1):
            flag = True

    print("Parents are: ",parents)

    pairs = parents_pairs(parents) # pair of parents
    print("Pair of parents are: ",pairs)
    print("\nNumber of parents pairs is: ",len(pairs))
```

Επιλέγουμε **τυχαία** ακολουθίες που αντιστοιχούν στο **30% του πληθυσμού** και αποθηκεύουμε τόσο τον **αύξοντα αριθμό** των ακολουθιών αυτών, όσο και το **σκορ** τους σε ένα λεξικό. Στην συνέχεια με την **τεχνική της ρουλέτας**, κατασκευάζουμε τους **υποψήφιους γονείς** και έπειτα με την κλήση της αντίστοιχης συνάρτησης (**parents_pairs()**) μας επιστρέφονται τα **υποψήφια ζευγαράκια γονέων**.

```
213     flag = False
214     while flag == False:
215         if (len(pairs)//2 + 1) < r-1:
216             num_of_p_pairs = random.randint(len(pairs)//2 + 1 , r-1)
217         else:
218             num_of_p_pairs = random.randint(r-1, len(pairs)//2)
219 
220         if (num_of_p_pairs <= len(pairs)):
221             flag = True
222     print("Final number of pairs is: ",num_of_p_pairs)
223 
224     # prepei na dialexw etsi ta zevgaria wste na simperilambanontai oloi oi komvoi
225     pairs_f = []
226 
227     for i in range(num_of_p_pairs):
228         pair = random.choice(pairs)
229         pairs_f.append(pair)
230         pairs.remove(pair)
231 
232     if not all([x in parents for x in pairs_f[i] for i in range(len(pairs_f))]): # ckeck if all numbers are in pairs list
233         print("Not all parents case!")
234 
235     print("Final pairs",pairs_f)
236 
237     # make children from these pairs of parents
238     children_table = single_point_crossover(pairs_f, dict)
239 
240     print("\nChildren's table  is:")
241     for child in children_table:
242         print(child)
243 
244     print("\nNumber of children is: ",len(children_table))
```

Στην συνέχεια επιλέγουμε **ποιος** θα είναι ο **τελικός αριθμός των ζευγαριών των γονέων**, καθώς και το **ποια** θα είναι αυτά τα ζευγάρια. Η επιλογή των ζευγαριών γίνεται **τυχαία**. Έχοντας πλέον καταλήξει στα ζευγάρια των γονιών, με την κλήση της κατάλληλης συνάρτησης (**single_point_crossover()**) μας επιστέφεται ο **πίνακας των παιδιών**, δηλαδή η **νέα γενιά**.

```
247     # NOW deal with the rest population
248     # choose randomly some of them to pass to the new generation
249     untouching_population = [[0] * (n) for _ in range(p - len(children_table))]
250     for i in range(p - len(children_table)):
251         flag = False
252         while flag == False:
253             k = random.randint(0, p-1)
254             if k not in untouching_population:
255                 untouching_population[i] = dict[k]
256                 flag = True
257
258     print("\nRest population is: ")
259     for r in untouching_population:
260         print(r)
261
262
263     # merge the 2 populations
264     dict = {}
265     i = 0
266
267     for child in children_table:
268         v = child
269         dict[i] = v
270         i+=1
271     for j in untouching_population:
272         v = j
273         dict[i] = v
274         i+=1
275
276     print("\nMerged dictionary is: ")
277     for k, v in dict.items():
278         print(k, v)
```

Η **επιλογή των μελών** της **υπάρχουσας** γενιάς, που θα συνεχίσουν και στην **επόμενη** γίνεται **τυχαία**. Έπειτα **συγχωνεύουμε** (σε ένα λεξικό) τα μέλη αυτά, με τα **παιδιά** που προέκυψαν νωρίτερα και έτσι προκύπτει ένας **νέος πληθυσμός**.

ΜΕΤΑΛΛΑΞΗ ΕΝΟΣ ΨΗΦΙΟΥ

Πραγματοποιούμε **μετάλλαξη ενός ψηφίου** και συγκεκριμένα στο **10%** του πληθυσμού.

```
281     '''
282     Mutation
283     - change list item color
284     - affects 10% of the population
285     '''
286
287     # metallaxh enos psifioy e.g sto 10% tou population
288     n1 = 0.1 * p
289
290     for i in range(int(n1)):  # afhnoume to endehoemno na pathei metallaxh xana to idio stoiheio
291         k = random.randint(0, p - 1) # random key
292         v = random.randint(0, n - 1) # random value index
293
294         flag = False
295         while flag == False:
296             new_color = random.choice(colors)
297             if new_color is not dict[k][v]:
298                 dict[k][v] = new_color
299                 flag = True
300
301     print("\nAfter mutation dictionary is: ")
302     for k, v in dict.items():
303         print(k, v)
```

Τυχαία επιλέγεται ο **αριθμός μιας ακολουθίας** και ο **αριθμός της θέσης** που θα γίνει η **αλλαγή** του χρώματος.

ΣΥΝΘΗΚΕΣ ΤΕΡΜΑΤΙΣΜΟΥ ΓΕΝΕΤΙΚΟΥ ΑΛΓΟΡΙΘΜΟΥ

Τρεις σε πλήθος είναι οι πιο συνηθισμένες **συνθήκες τερματισμού** ενός **γενετικού** αλγορίθμου.

1. Η εύρεση μιας **τέλειας** λύσης με βάση τη συνάρτηση **καταλληλόλητας**.

2. Η **σύγκλιση** όλων των λύσεων σε **μία**.

3. Να έχουμε **υπερβεί** ένα μεγάλο **όριο επαναλήψεων**.

ΥΠΟΛΟΓΙΣΜΟΣ ΤΗΣ ΤΕΛΕΙΑΣ ΛΥΣΗΣ

```
140   # counting the best score
141   best_score = 0
142   for i in range(n):
143       for j in range(n):
144           if N[i][j] == 1:
145               best_score += 1
146
147   #print("\nBest score is: ", best_score)
```

Για τον υπολογισμό της καλύτερης λύσης, αρκεί να μετρήσουμε το **πλήθος των άσσων** του **πίνακα γειτνίασης**. Οι άσσοι ουσιαστικά αντιστοιχούν στους **γείτονες** του κάθε κόμβου του γράφου, οι οποίοι οφείλουν να έχουν και **διαφορετικό** χρώμα.

```
153   repetitions = 0
154   flag1 = False
155
156   while (repetitions < 30 or flag1 == False):
157
158       scores_d = find_scores(dict)
159       print("\nScores of random population in dictionary:")
160       print(scores_d)
161
162       b = {}
163       for i in range(p):
164           if (best_score - scores_d[i]) <= 7:
165               v = best_score - scores_d[i]
166               b[i] = v
167               flag1 = True
168               break
169
170       if flag1:
171           # print("\nTermination because of best score.")
172           break
173
```

Οι **συνθήκες τερματισμού** του αλγορίθμου στην γραμμή **156**. Στις γραμμές **162-172** αποθηκεύουμε σε ένα **λεξικό** τις **λύσεις** εκείνες που έχουν **σκορ** αρκετά καλό, **συγκριτικά** με το σκορ της **καλύτερης λύσης** (απόκλιση <= 7). Αν **βρεθούν** τέτοιες λύσεις ο αλγόριθμος **τερματίζει**.

```
306         repetitions+=1
307
308     # end of loop
309     print("\nBest score of all is: ", best_score)
310
311     if flag1:
312         print("Termination because of best score.")
313         if (len(b.keys()) > 1): # more than one best scores
314             sorted_dict = sorted(b.items(), key=lambda x:x[1])
315             best = dict[sorted_dict[0][0]]
316             b_score = scores_d[sorted_dict[0][0]]
317         else: # one best score
318             best = dict[list(b.keys())[0]]
319             b_score = scores_d[list(b.keys())[0]]
320         print("\nBest graph colouring solution(s) is/are: ", best)
321         print("With score(s): ", b_score)
322
323     else:
324         print("Termination because of repetitions")
325         print("\nFinal population: ")
326         for k, v in dict.items():
327             print(k, v)
328
329         scores_d = find_scores(dict)
330         print("\nFinal population's scores are:")
331         print(scores_d)
```

Πριν το τέλος του βρόγχου επανάληψης **αυξάνουμε κατά 1** τον αριθμό των επαναλήψεων. Όταν ο αλγόριθμος **τερματίσει**, εμφανίζουμε τα αντίστοιχα αποτελέσματα.

ΣΤΙΓΜΥΟΤΥΠΑ ΕΚΤΕΛΕΣΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

```
0 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
1 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
2 ['green', 'red', 'green', 'green', 'green', 'red', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
3 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
4 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
5 ['green', 'red', 'green', 'green', 'green', 'red', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
6 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
7 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
8 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
9 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
10 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
11 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
12 ['blue', 'yellow', 'green', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'green', 'blue', 'blue', 'blue']
13 ['green', 'red', 'green', 'green', 'blue', 'yellow', 'red', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
14 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
15 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
16 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'green', 'blue', 'blue', 'blue']
17 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
18 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
19 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']

Scores of random population in dictionary:
{0: 67, 1: 65, 2: 63, 3: 71, 4: 67, 5: 63, 6: 71, 7: 71, 8: 65, 9: 71, 10: 65, 11: 71, 12: 65, 13: 64, 14: 71, 15: 71, 16: 69, 17: 67, 18: 65, 19: 71}

Partial popluation r is  6

Partial scores in dict are:  {19: 71, 10: 65, 15: 71, 17: 67, 1: 65, 3: 71}
Sum of all dictionary values is:  410


Parents are:  [17, 10, 3]
Pair of parents are:  [(17, 10), (17, 3), (10, 3)]

Number of parents pairs is:  3
Final number of pairs is:  3
Final pairs [(17, 10), (17, 3), (10, 3)]

Children's table  is:
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']

Number of children is:  6
```

Ο πληθυσμός που έρχεται ως είσοδος, **πριν** τον τερματισμό του αλγορίθμου. Βλέπουμε **ποιος** είναι

ο πληθυσμός αυτός και τα **σκορ** αυτού, **ποιος** θα αποτελέσει τον **μερικό πληθυσμό**, **ποιοι** οι **γονείς** για τον μερικό αυτό πληθυσμό, καθώς και **ποια** τα **παιδιά** αυτών.

```
Rest population is:
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['green', 'red', 'green', 'green', 'green', 'red', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']

Merged dictionary is:
0 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
1 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
2 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
3 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
4 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
5 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
6 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
7 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
8 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
9 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
10 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
11 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
12 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
13 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
14 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
15 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
16 ['green', 'red', 'green', 'green', 'green', 'red', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
17 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
18 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
19 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
```

Βλέπουμε τον **υπόλοιπο** πληθυσμό (rest population), και τον πληθυσμό **μαζί** με την μερική ανανέωση που αναφέραμε προηγουμένως.

```
After mutation dictionary is:
0 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
1 ['blue', 'red', 'blue', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
2 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
3 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
4 ['blue', 'red', 'blue', 'blue', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
5 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
6 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'red', 'yellow', 'blue', 'blue', 'blue']
7 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'green', 'blue']
8 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
9 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
10 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
11 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'green', 'blue']
12 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
13 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
14 ['blue', 'yellow', 'green', 'green', 'green', 'red', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
15 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
16 ['green', 'red', 'green', 'green', 'green', 'red', 'yellow', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
17 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'green', 'blue']
18 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'blue', 'blue']
19 ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'green', 'blue']

Scores of random population in dictionary:
{0: 67, 1: 65, 2: 67, 3: 71, 4: 59, 5: 71, 6: 67, 7: 78, 8: 71, 9: 67, 10: 67, 11: 78, 12: 71, 13: 67, 14: 67, 15: 71, 16: 63, 17: 78, 18: 71, 19: 78}

Best score of all is:  84
Termination because of best score.

Best graph colouring solution(s) is/are:  ['blue', 'yellow', 'red', 'green', 'green', 'blue', 'blue', 'green', 'red', 'yellow', 'red', 'blue', 'yellow', 'blue', 'green', 'blue']
With score(s):  78
```

Ο πληθυσμός μετά από **μετάλλαξη ενός** ψηφίου στο **10%** του πληθυσμού. Ο πληθυσμός αυτός αποτελεί και τον **τελικό πληθυσμό**, καθώς ο αλγόριθμος **τερματίζει** λόγω εύρεσης μιας **τέλειας λύσης** με βάση την **συνάρτηση καταλληλόλητας**. Βλέπουμε τα **σκορ** του τελικού πληθυσμού, το **σκορ** της **καλύτερης λύσης** και το **ποια** είναι αυτή.