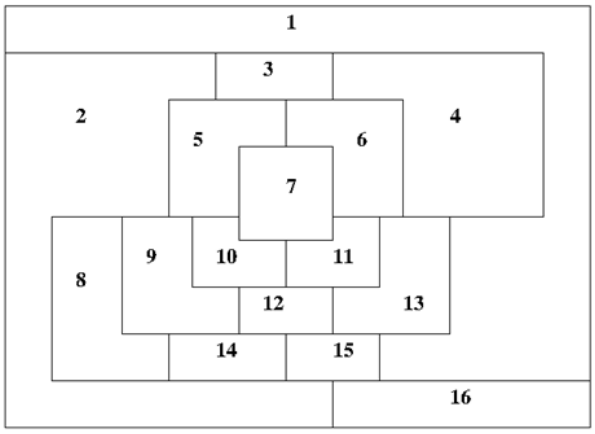
«Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα»

Δημοπούλου Νικολίτσα(Π20057)

Εκφώνηση:

Αναπτύξτε πρόγραμμα χρωματισμού του παρακάτω γράφου με χρήση γενετικών αλγορίθμων και γλώσσα προγραμματισμού της επιλογής σας. Τα διαθέσιμα χρώματα είναι 4: μπλε, κόκκινο, πράσινο, κίτρινο.



Αρχικά κάνω μία αναπαράσταση του παραπάνω γράφου σε μορφή λεξικού, όπου κάθε κλειδί αντιστοιχεί σε έναν κόμβο και κάθε μία από τις τιμές του κλειδιού δηλώνουν τους γείτονές του(πρώτος θεωρείται ο κόμβος 0 και όχι ο 1 όπως στο σχήμα). Μετά δηλώνεται η λίστα ***rand_l*** , η οποία αποτελείται από 200 υπολίστες με 16 μηδενικά η καθεμία(όσοι και οι κόμβοι). Εδώ θα αποθηκευτούν οι υποψήφιες λύσεις του προγράμματος.

```
import random
import bisect


graph2 = {0: [1, 2, 3, 12, 14, 15],
          1: [0, 2, 4, 7, 8, 13, 14, 15],
          2: [0, 1, 3, 4, 5],
          3: [0, 2, 5, 12],
          4: [1, 2, 5, 6, 8, 9],
          5: [2, 3, 4, 6, 10, 12],
          6: [4, 5, 9, 10],
          7: [1, 8, 13],
          8: [1, 4, 7, 9, 11, 13],
          9: [4, 6, 8, 10, 11],
          10: [5, 6, 9, 11, 12],
          11: [8, 9, 10, 12, 13, 14],
          12: [0, 3, 5,  10, 11, 14, 15],
          13: [1, 7, 8, 11, 14],
          14: [0, 1, 11, 12, 13, 15],
          15: [0, 1, 14],
          }
```

```
# random first list
rand_l = []
for i in range(200):
    row = [0] * 16
    rand_l.append(row)
```

Το πρόγραμμα, αποτελείται κατά κύριο λόγο από συναρτήσεις, η επεξήγηση των οποίων ακολουθεί παρακάτω:

Συναρτήσεις

1) ***mutation(rand_list)***
   Η συνάρτηση αυτή, είναι συνάρτηση μετάλλαξης. Δέχεται ως όρισμα μία λίστα(που περιέχει υπολίστες) και με ποσοστό 10% εκτελεί την λειτουργία μετάλλαξης ως εξής: Επαναληπτικά για κάθε στοιχείο(υπολίστα) της λίστας rand_list συγκρίνει έναν τυχαίο αριθμό με το παραπάνω ποσοστό και αν τον βρει μικρότερο αλλάζει με τυχαίο τρόπο τις τιμές της αντίστοιχης θέσης.

```
def mutation(rand_list):
    m_rate = 0.1
    for mut2 in range(len(rand_list)):
        if random.random() < m_rate:
            rand_list[mut2] = random.randint(1, 4)
    return rand_list
```

2) ***count_score(rand_list)***
   Η συνάρτηση αυτή είναι υπεύθυνη για τον υπολογισμό του σκορ κάθε υπολίστας βάσει του λεξικού γειτνίασης. (Το *score[s]//2* χρησιμοποιείται για να αφαιρεθούν από το τελικό σκορ τα διπλότυπα).

```
def count_score(rand_list):
    # list with the score of every row of rand_list
    score = [0] * 200

# rand_list contains 20 rows
    for r2 in range(len(rand_list)):
        for km, vm in graph2.items():
            for v5 in vm:
                if rand_list[r2][km] != rand_list[r2][v5]:
                    score[r2] += 1
    for s in range(len(score)):
        score[s] //= 2

    # print("score=", score)
    return score
```

3) ***roulette(lista)***
   Η συνάρτηση αυτή, δέχεται ως όρισμα μία λίστα με ακεραίους(εδώ την λίστα των σκορ) και επιστρέφει μία άλλη λίστα η οποία περιλαμβάνει το άθροισμα κάθε αριθμού της αρχικής λίστας με τον προηγούμενο του.
   (Αν πχ lista = [1, 2, 3, 4, 5] -> roulette_list = [1, 3, 6, 10, 15])

```
def roulette(lista):
    roulette_list = [lista[0]]
    for rlt in range(1, len(lista)):
        roulette_list.append(lista[rlt]+roulette_list[rlt-1])
    return roulette_list
```

4) ***select_parent(list, summ, rlt)***

Η συνάρτηση αυτή, κάνει την επιλογή γονέων. Δέχεται ως ορίσματα μία λίστα με αριθμούς(την λίστα που επέστρεψε η παραπάνω συνάρτηση),το άθροισμα των αριθμών αυτών και μια λίστα που περιέχει τα κατάλληλα όρια για την κλήρωση στην κατανομή της ρουλέτας. Η συνάρτηση δημιουργεί μια κενή λίστα (*par_l*) και επαναλαμβάνει το εξής: για κάθε αριθμό της lista, επιλέγεται τυχαία ένας αριθμός από το 0 μέχρι το summ και στη συνέχεια επιλέγεται ο αριθμός της rlt που βρίσκεται στη θέση που θα είχε αυτός ο τυχαίος αριθμός στη λίστα rlt. Αυτός ο αριθμός προστίθεται στη λίστα *par_l*.

```
def select_parent(lista, summ, rlt):
    par_l = []
    for pr in range(len(lista)):
        rn = random.randint(0, summ)
        # print("random numbers", rn)
        r2 = bisect.bisect_left(rlt, rn)
        par_l.append(r2)
    return par_l
```

5) ***couple_nodes(parent_list, random_list)***

Η παρακάτω συνάρτηση, παίρνει ως είσοδο δύο λίστες: τη λίστα *parent_list*, που περιέχει τους δείκτες των γονέων που έχουν επιλεχθεί, και τη λίστα *random_list,* που περιέχει μια λίστα αριθμών. Η συνάρτηση δημιουργεί μια κενή λίστα (*node_list*) και για κάθε δείκτη του *parent_list*, προσθέτει το στοιχείο του *random_list* που βρίσκεται στη θέση που αντιστοιχεί σε αυτόν το δείκτη στην *node_list*. Τελικά, η συνάρτηση επιστρέφει τη λίστα node_list, η οποία περιέχει τους αριθμούς που αντιστοιχούν στους γονείς που έχουν επιλεχθεί.

Πχ αν parent_list = [2, 4, 0] και random_list = [1, 2, 3, 4, 5], τότε node_list = [3, 5, 1]. Δηλαδή ο πρώτος γονέας που έχει επιλεχθεί έχει τιμή 3 στη θέση του δείκτη 2 στη λίστα random_list κοκ.

```
def couple_nodes(parent_list, random_list):
    node_list = []
    for prn in parent_list:
        node_list.append(random_list[prn])
    return node_list
```

6) ***child_meth(node_lista)***

Η συνάρτηση παίρνει ως είσοδο μια λίστα από κόμβους (*node_lista*) που περιέχει τους κόμβους που αντιστοιχούν στα δύο γονείς που επιλέχθηκαν για να δημιουργηθούν τα παιδιά.

Η συνάρτηση δημιουργεί μια κενή λίστα (*ch_list*) και για κάθε ζευγάρι γονιών στην node_lista, δημιουργεί δύο παιδιά. Κάθε παιδί δημιουργείται από τον συνδυασμό της πρώτης μισής του κόμβου του ενός γονέα με τη δεύτερη μισή του κόμβου του άλλου γονέα. Ο συνδυασμός αυτός γίνεται για τα πρώτα 8 γονίδια του πρώτου γονέα και τα υπόλοιπα 8 γονίδια του δεύτερου γονέα για το πρώτο παιδί, και αντίστροφα για το δεύτερο παιδί.

Τελικά, η συνάρτηση επιστρέφει τη λίστα ch_list, η οποία περιέχει τα παιδιά που δημιουργήθηκαν.

```
def child_meth(node_lista):
    ch_list = []
    # range 4
    for ch in range(0, len(node_lista), 2):
        q1 = node_lista[ch][:8] + node_lista[ch+1][-8:]
        q2 = node_lista[ch+1][:8] + node_lista[ch][-8:]
        ch_list.append(q1)
        ch_list.append(q2)
    return ch_list
```

7) ***partial_update(random_list)***

Η συνάρτηση αυτή αφορά την μερική ανανέωση του πληθυσμού. Δέχεται μια λίστα από λίστες (*random_list*) ως είσοδο και ενημερώνει τυχαία μερικά από τα στοιχεία στις υπολίστες του *random_list*. Συγκεκριμένα, για κάθε υπολίστα στο *random_list*, η συνάρτηση δημιουργεί ένα τυχαίο αριθμό μεταξύ 0 και 1 και εάν αυτός ο αριθμός είναι μικρότερος από 0.3(ποσοστό ανανέωσης πληθυσμού), τότε ενημερώνει τυχαία μερικά από τα στοιχεία της υπολίστας με τιμές από 1 έως 4. Στη συνέχεια, επιστρέφει το ενημερωμένο random_list.

```
def partial_update(random_list):
    for u1 in range(len(random_list)):
        if random.random() < 0.3:
            for u2 in range(16):
                r_int = random.randint(1, 4)
                random_list[u1][u2] = r_int
    return random_list
```

Main:

Μόνο για την πρώτη φορά εκτέλεσης του προγράμματος επιλέγεται τυχαία ο αρχικός πληθυσμός(το πρόγραμμα έχει επιλεγεί να δημιουργεί 200 υποψήφιες λύσεις) με τιμές από το 1 έως το 4 για καθένα από τα 4 χρώματα που θα χρωματίσουν τον γράφο. Επίσης εδώ αρχικοποιείται και η μεταβλητή *max_sc* στην οποία θα αποθηκευτεί το μέγιστο σκορ για την εξαγωγή της βέλτιστης λύσης. Ύστερα ακολουθεί μια επαναληπτική δομή, η οποία εκτελείται 1000 φορές για μεγαλύτερη ακρίβεια του αποτελέσματος και περιέχει κατά κύριο λόγο την κλήση των παραπάνω συναρτήσεων.

```
# First Random solution
max_sc = 0
# fill rand_l with random values-only for first time
for i in range(200):
    for j in range(16):
        rand_int = random.randint(1, 4)
        rand_l[i][j] = rand_int
```

```
for t in range(1000):
    rand_l = mutation(rand_l)

    # count score for the elements of list
    scr = count_score(rand_l)
    sum_l = sum(scr)

    for sc in range(len(scr)):
        if scr[sc] > max_sc:
            max_sc = scr[sc]
            th = scr.index(max_sc)
            best_choice = rand_l[th]

    # roulette algorithm
    rl = roulette(scr)
    # print(rl)

    # parent is the list that contains rows-parents
    parent = select_parent(rand_l, sum_l, rl)
    # print("parent=", parent)
    nodes = couple_nodes(parent, rand_l)
    rand_l.clear()
    rand_l = []
    rand_l = child_meth(nodes)
    rand_l = partial_update(rand_l)

print("max score= ", max_sc)
print("best choice=", best_choice)
```

Μετά την δημιουργία του τυχαίου αρχικού πληθυσμού, καλείται η συνάρτηση ***mutation(rand_list)*** για την διαδικασία της μετάλλαξης(ποσοστό 10%), ώστε να διαφοροποιηθούν κάποια στοιχεία. Ύστερα καλείται η ***count_score(rand_list)*** στην οποία υπολογίζεται η συνάρτηση καταλληλόλητας του πληθυσμού. Η λίστα ***scr*** περιέχει το σκορ για κάθε κόμβο της τρέχουσας υποψήφιας λύσης .Η μεταβλητή ***sum_l*** περιέχει το συνολικό σκορ κάθε υποψήφιας λύσης.

Κατόπιν για κάθε σκορ που προκύπτει γίνεται έλεγχος για το αν είναι το μέγιστο που έχει βρεθεί έως εκείνη την στιγμή στο πρόγραμμα και αν αυτό ισχύει, θέτει το ***max_sc*** ίσο με το εν λόγω σκορ και στην λίστα ***best_choice*** αποθηκεύεται η λύση που σημείωσε το μέγιστο σκορ.

Μετά καλείται η συνάρτηση ***roulette(lista)*** η οποία στην ουσία επιστρέφει την λίστα που θα διευκολύνει την ισότιμη απονομή των γονέων βάσει καλύτερου σκορ. Έτσι καλείται ύστερα η ***select_parent(list, summ, rlt)*** για την τυχαία επιλογή γονέων(δίνεται βαρύτητα στους κόμβους με το καλύτερο σκορ λόγω της συνάρτησης ρουλέτας).

Μετά την επιλογή των γονέων καλείται η συνάρτηση ***couple_nodes(parent_list, random_list)*** για να επιστραφούν οι κόμβοι της λίστας rand_l στους οποίους αντιστοιχούν οι γονείς που επιλέχθηκαν. Έτσι οι γονείς «ζευγαρώνονται» ανά 2 και μέσω της συνάρτησης ***child_meth(node_lista)*** προκύπτουν οι λύσεις παιδιά με τον τρόπο που αναφέρθηκε στην επεξήγηση της εν λόγω συνάρτησης.

Τέλος γίνεται μερική ανανέωση του πληθυσμού σε ποσοστό 30% μέσω της συνάρτησης ***partial_update(random_list).***

Το πρόγραμμα δίνει έξοδο το ***max_sc*** και την λύση(από την λίστα rand_l) στην οποία αντιστοιχεί το εν λόγω σκορ.

<u>Παράδειγμα εκτέλεσης:</u>

Το οποίο αν υποθέσουμε ότι 1 = μπλε, 2 = κόκκινο, 3 = πράσινο, 4 = κίτρινο δίνει ως αποτέλεσμα το παρακάτω γράφημα: