

Εξώφυλλο με Στοιχεία

Μύαρης Στυλιανός Π22114

Εφαρμοσμένη Συνδυαστική Προαιρετική Άσκηση

Πανεπιστήμιο Πειραιώς

Διδάσκουσα: Ευγενία Ψαρρά

Περιγραφή Υλοποίησης

Το πρόγραμμα αποτελείται από 3 συναρτήσεις και την `main()`.

Για την περιγραφή της υλοποίησης θα ακολουθήσουμε την ροή της συνάρτησης `main()`.

Αρχικά ορίζεται ο αριθμός των πόλεων από το πρόγραμμα με την μεταβλητή `num_cities`. Ήθελα την μεταβλητή ώστε να μπορεί ο προγραμματιστής να μεταβάλλει τον αριθμό των πόλεων σε περισσότερες ή λιγότερες από 10 πόλεις που ζητάει η εκφώνηση.

Στην μεταβλητή `cities` εκχωρείται το αποτέλεσμα της συνάρτησης `get_coordinates(num_cities)`. Η παράμετρος είναι ο αριθμός των πόλεων που θα ζητηθούν οι συντεταγμένες τους από τον χρήστη.

Στην συνάρτηση `get_coordinates()` δημιουργείται αρχικά μία λίστα `cities`. Έπειτα με ένα `for loop` όπου το `i` ξεκινάει από 0 και καταλήγει στο `num_cities`, ζητείται από τον χρήστη να εκχωρήσει τις συντεταγμένες κάθε πόλης με την μορφή `x y` (χι κενό ψι). Η είσοδος αυτή χωρίζεται με το κενό (`split()`) και εκχωρείται στις μεταβλητές `x` και `y` αντίστοιχα. Τέλος, εκχωρούνται στην λίστα `cities` (με την εντολή `append`) την οποία επιστρέφει η συνάρτηση.

Αμέσως μετά στην συνάρτηση `main()`, δημιουργούνται δύο μεταβλητές `path` και `distance` στις οποίες εκχωρούνται τα αποτελέσματα της συνάρτησης `tsp_backtracking(cities)` όπου `cities` είναι η λίστα με τις συντεταγμένες των πόλεων.

Χρησιμοποιώντας την βιβλιοθήκη `itertools` εκχωρούμε σε μία μεταβλητή την `itertools.permutations` που δέχεται σαν παράμετρο μία λίστα. Η λίστα που θα του δώσουμε θα είναι η `cities [1:]` δηλαδή η `cities` εκτός από την πρώτη πόλη. Αυτό το κάνουμε διότι ξεκινάμε από μία συγκεκριμένη πόλη και ξανά καταλήγουμε σε αυτήν, οπότε δεν μας ενδιαφέρουν λίστες που περιέχουν όλες τις πόλεις (οι συνολικοί συνδυασμοί μας είναι $(10-1)! = 9!$). Ορίζουμε επίσης και δυο μεταβλητές `min_path` και `min_distance`.

Τώρα για κάθε ξεχωριστό `permutation` κάνουμε τα εξής σε ένα `for loop`:

- Ορίζουμε το μία μεταβλητή, το `current_path`. Την αρχικοποιούμε στην αρχή του loop ως την πρώτη πόλη + το συγκεκριμένο permutation των υπόλοιπων πόλεων κάθε φορά.
- Ορίζουμε με ένα άλλο for loop, μία άλλη μεταβλητή το `current_distance`. Ουσιαστικά, θα είναι το άθροισμα (sum) των αποστάσεων των πόλεων ανά δύο μέσα στο permutation. Το loop μας πηγαίνει από το 0 μέχρι το μήκος του `current_path` - 1 ώστε μετά να προσθέσουμε και την απόσταση της τελευταίας πόλης μέχρι την αρχική μας.
- Σε περίπτωση που το `current_distance` είναι μικρότερο από το `min_distance` που ορίσαμε στην αρχή της `tsp_backtracking`, αντικαθιστούμε το `min_distance` με το `current_distance` και το `min_path` με το `current_path`.

Μέχρι το τέλος της συνάρτησης, ο αλγόριθμος έχει ελέγξει όλους τους πιθανούς συνδυασμούς μεταξύ των πόλεων και έχει κρατήσει στην μεταβλητή `min_path` την συντομότερη. Επιστρέφουμε το `min_path` και το `min_distance`.

Η απόσταση μεταξύ δύο πόλεων γίνεται με την χρήση της συνάρτησης `distance()` η οποία παίρνει σαν ορίσματα τις συντεταγμένες 2 πόλεων και επιστρέφει την Ευκλείδεια απόστασή τους.

Στο τέλος της `main()` έχουμε τα αποτελέσματά μας στις μεταβλητές `path` και `distance`. Εκτυπώνουμε τις συντεταγμένες των πόλεων που υπάρχουν μέσα στο `path` και φυσικά την συνολική απόσταση της διαδρομής.

Ψευδοκώδικας

Είναι ουσιαστικά ο ίδιος κώδικας αλλά πιο «κατανοητός»

```
function distance(city1, city2)
```

```
    return sqrt((city1.x - city2.x)^2 + (city1.y - city2.y)^2)
```

```
function tsp_backtracking(cities)
```

```
    permutations = generate_permutations(cities[1:])
```

```
    min_path = None
```

```
    min_distance = infinity
```

```
    for each perm in permutations do
```

```
current_path = [cities[0]] + perm
```

```
current_distance = 0
```

```
for i from 0 to length(current_path) - 2 do
```

```
    current_distance = current_distance + distance(current_path[i],  
current_path[i+1])
```

```
    current_distance = current_distance + distance(current_path[-1],  
cities[0]) // returning to the start city
```

```
if current_distance < min_distance then
```

```
    min_distance = current_distance
```

```
    min_path = current_path
```

```
return min_path, min_distance
```

```
function get_coordinates(num_cities)
```

```
    cities = []
```

```
    for i from 1 to num_cities do
```

```
        x, y = input("Enter coordinates for city " + i + " (format: x y): ")
```

```
        cities.append((x, y))
```

```
    return cities
```

```
function main()
```

```
    num_cities = 10
```

```
    print("Please enter the coordinates of the cities:")
```

```
    cities = get_coordinates(num_cities)
```

```
path, distance = tsp_backtracking(cities)
```

```
print("The shortest path is:")
```

```
for each city in path do
```

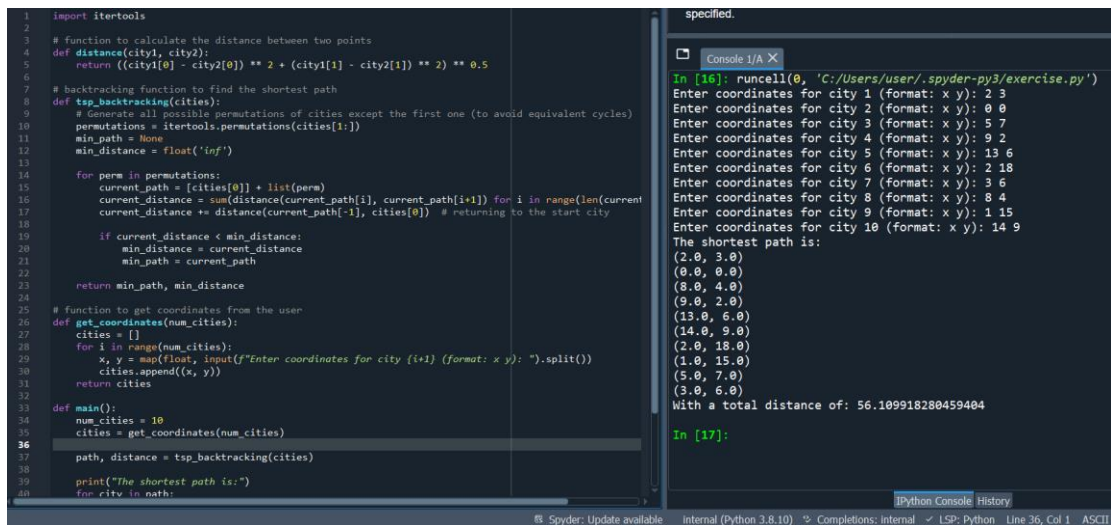
```
    print(city)
```

```
print("With a total distance of: " + distance)
```

```
if __name__ == "__main__" then
```

```
    main()
```

Στιγμιότυπο Εκτέλεσης του Κώδικα



```
1 import itertools
2
3 # function to calculate the distance between two points
4 def distance(city1, city2):
5     return ((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2) ** 0.5
6
7 # backtracking function to find the shortest path
8 def tsp_backtracking(cities):
9     # Generate all possible permutations of cities except the first one (to avoid equivalent cycles)
10    permutations = itertools.permutations(cities[1:])
11    min_path = None
12    min_distance = float('inf')
13
14    for perm in permutations:
15        current_path = [cities[0]] + list(perm)
16        current_distance = sum(distance(current_path[i], current_path[i+1]) for i in range(len(current_path)-1))
17        current_distance += distance(current_path[-1], cities[0]) # returning to the start city
18
19        if current_distance < min_distance:
20            min_distance = current_distance
21            min_path = current_path
22
23    return min_path, min_distance
24
25 # function to get coordinates from the user
26 def get_coordinates(num_cities):
27     cities = []
28     for i in range(num_cities):
29         x, y = map(float, input(f"Enter coordinates for city {i+1} (format: x y): ").split())
30         cities.append((x, y))
31     return cities
32
33 def main():
34     num_cities = 10
35     cities = get_coordinates(num_cities)
36
37     path, distance = tsp_backtracking(cities)
38
39     print("The shortest path is:")
40     for city in path:
```

```
In [16]: runcell(0, 'C:/Users/user/.spyder-py3/exercise.py')
Enter coordinates for city 1 (format: x y): 2 3
Enter coordinates for city 2 (format: x y): 0 0
Enter coordinates for city 3 (format: x y): 5 7
Enter coordinates for city 4 (format: x y): 9 2
Enter coordinates for city 5 (format: x y): 13 6
Enter coordinates for city 6 (format: x y): 2 18
Enter coordinates for city 7 (format: x y): 3 6
Enter coordinates for city 8 (format: x y): 8 4
Enter coordinates for city 9 (format: x y): 1 15
Enter coordinates for city 10 (format: x y): 14 9
The shortest path is:
(2.0, 3.0)
(0.0, 0.0)
(8.0, 4.0)
(9.0, 2.0)
(13.0, 6.0)
(14.0, 9.0)
(2.0, 18.0)
(1.0, 15.0)
(5.0, 7.0)
(3.0, 6.0)
With a total distance of: 56.109918280459404

In [17]:
```