ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΠΟΛΙΤΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΔΟΜΟΣΤΑΤΙΚΗΣ

Ακαδ. Έτος: 2016-2017

Μάθημα: Εφαρμογές Η/Υ

Δευτέρα, 17/10/2016

Διδάσκοντες:
Ν.Δ. Λαγαρός (Επικ. Καθηγητής), Αθ. Στάμος (ΕΔΙΠ), Χ. Φραγκουδάκης (ΕΔΙΠ)

Παραδείγματα για την 3η παράδοση – Σύνθετοι τύποι αντικειμένων

1. Αραίωση τοπογραφικών σημείων

Το μοντέρνο τοπογραφικό όργανο LIDAR μετράει υψομετρικά τοπογραφικά σημεία ανά περίπου 0.30 m. Το πλήθος των σημείων είναι τεράστιο, για παράδειγμα αντιστοιχεί σε $3 \cdot 10^6$ σημεία για τοπογραφικό οδού μήκους 3 km. Για να μειωθεί ο όγκος εργασίας σε στάδιο προμελέτης τα σημεία πρέπει να γίνουν αραιότερα, περίπου ανά 10m ή και περισσότερο. Να συνταχθεί πρόγραμμα σε Python που να αραιώνει τα σημεία.

Λύση

Τα σημεία θεωρείται ότι είναι αποθηκευμένα σε μία λίστα από πλειάδες. Αν όλο το τοπογραφικό χωρισθεί σε κάναβο βήματος step=10 m, δηλαδή σε πίνακα χωρισμένο σε τετραγωνίδια πλευράς 10m, τότε το τετραγωνίδιο που αντιστοιχεί στη γραμμή i και στήλη j του πίνακα περικλείει τα σημεία με x και y που ικανοποιούν:

$$j \cdot step \le x < (j+1) \cdot step \quad , \quad i \cdot step \le y < (i+1) \cdot step$$

Θεωρούμε ότι η πρώτη γραμμή αντιστοιχεί σε i=0 και η πρώτη στήλη σε j=0. Διαιρώντας δια step βρίσκουμε:

$$j \le \frac{x}{step} < j+1 \quad , \quad i \le \frac{y}{step} < i+1 \quad \Rightarrow \quad j = \left[\frac{x}{step} \right] \quad , \quad i = \left[\frac{y}{step} \right]$$

όπου οι αγκύλες σημαίνουν ακέραιο μέρος με αποκοπή δεκαδικών. Αυτό οδηγεί στον αλγόριθμο:

```
Θεώρησε όλα τα τετραγωνίδια άδεια.
Για κάθε σημείο x,y:
    Υπολόγισε i, j.
    Αν το τετραγωνίδιο i, j είναι άδειο:
        Θεώρησε το τεραγωνίδιο i, j γεμάτο.
        Αποθήκευσε το σημείο x, y.
    Αλλιώς:
        Αγνόησε το σημείο x, y.
    Τέλος Αν.
Τέλος Για.
```

Για να υλοποιήσουμε τον αλγόριθμο, θεωρούμε ένα σύνολο (set) με όνομα grid το οποίο αρχικά είναι άδειο. Ένα τετραγωνίδιο i,j θεωρείται γεμάτο αν το διατεταγμένο ζεύγος i,j υπάρχει μέσα στο σύνολο grid και άδειο αν όχι. Το διατεταγμένο ζεύγος το υλοποιούμε ως την πλειάδα (i, j). Τα αρχικά σημεία τα έχουμε στη λίστα points και τα αραιωμένα στη λίστα sparse.

```
# -*- coding: iso-8859-7 -*-
points = [     (20373.206,      24891.332,        144.504),
               (20360.858,      24871.862,        144.277),
               (20355.826,      24860.952,        144.346),
               (20346.379,      24856.055,        143.848),
               (19529.865,      23707.808,        183.190),
               (19535.464,      23687.263,        176.400),
               (19547.140,      23687.365,        170.260),
               (19547.578,      23707.596,        174.300),
               (19547.424,      23727.489,        178.840),
```

```
            (19545.609,      23760.878,        183.460),
            (19555.842,      23710.546,        170.900),
            (19584.158,      23718.479,        158.330),
            (19599.834,      23698.305,        145.900),
            (19609.027,      23697.602,        138.040),
        ]

def pyMain():
    "Start here."
    for cp in araia(points, 50):
        print(cp)

def araia(points, step):
    "Αραίωση τοπογραφικών σημείων."
    sparse = []
    grid = set()
    for cp in points:
        i = int(cp[1]/step)
        j = int(cp[0]/step)
        if (i, j) in grid: continue
        grid.add((i, j))
        sparse.append(cp)
    return sparse
```

Προσοχή: τα σημεία πρέπει να είναι πλειάδες συντεταγμένων (x, y, z) και όχι λίστες [x, y, z] διότι αλλιώς αν αλλάξουμε πχ το x σε ένα σημείο της λίστας sparse θα αλλάξει και το x του αντίστοιχου σημείου στη λίστας points.

2. Έλεγχος πεπερασμένων στοιχείων

Σε πρόγραμμα πεπερασμένων στοιχείων τυπικά ορίζονται κόμβοι με όνομα, x, y, z ως εξής:

```
K1  0.0  0.0
K2  3.0  0.0
K3  6.0  0.0
A1  1.5  2.0
A2  4.5  2.0
```

Επίσης ορίζονται ράβδοι που συνδέουν τους κόμβους ως όνομα, κόμβος αρχής, κόμβος τέλους, ως εξής:

```
PK1  K1  K2
PK2  K2  K3
PA1  K4  K5
PΔ1  K1  A1
PΔ2  A1  K2
PΔ3  K2  A2
PΔ4  A2  K3
```

Για να αποφευχθούν πολλά προβλήματα κατά την επίλυση πρέπει να εξασφαλιστεί:
α) Ο κόμβος αρχής και τέλους μίας ράβδου πρέπει να έχει οριστεί (να έχει συντεταγμένες)
β) Κάθε κόμβος πρέπει να συνδέεται με τουλάχιστον 2 ράβδους
Να συνταχθεί πρόγραμμα σε Python που να κάνει αυτούς τους ελέγχους και επιπλέον να υπολογίζει το μήκος της κάθε ράβδου.

Λύση

θα μετατρέψουμε τους κόμβους ως λεξικό (joint) με κλειδί το όνομα και τιμή μια πλειάδα με τις συντεταγμένες, διότι οι συντεταγμένες χρειάζονται για τον υπολογισμό του μήκους. Ο πρώτος έλεγχος είναι απλός: και οι δύο κόμβοι κάθε ράβδου πρέπει να βρίσκονται στο λεξικό joint. Για το δεύτερο έλεγχο, δημιουργείται ένα λεξικό (nb) με κλειδί το όνομα κάθε κόμβου και με τιμή ακέραιο που περιέχει το πλήθος ράβδων που συνδέουν τον κόμβο. Για κάθε κόμβο κάθε ράβδου προσθέτουμε ένα στον ακέραιο αυτό. Στη συνέχεια ελέγχουμε αν για κάποιο κόμβο ο ακέραιος είναι μικρότερος του 2. Τέλος το μήκος υπολογίζεται με το πυθαγόρειο θεώρημα.

```
# -*- coding: iso-8859-7 -*-
from math import hypot
```

```
js = [("K1",  0.0,  0.0),
      ("K2",  3.0,  0.0),
      ("K3",  6.0,  0.0),
      ("A1",  1.5,  2.0),
      ("A2",  4.5,  2.0),
     ]
bs = [("PK1",  "K1",  "K12"),
      ("PK2",  "K2",  "K3"),
      ("PA1",  "A1",  "A2"),
      ("PΔ1",  "K1",  "A1"),
      ("PΔ2",  "A1",  "K2"),
      ("PΔ3",  "K2",  "A2"),
      ("PΔ4",  "A2",  "K3"),
     ]

def pyMain():
    "Transform coordinates to dictionary and call tests."
    joint = {}
    for j in js:
        joint[j[0]] = j[1:]
    testBars(joint)
    testJoints()
    barLength(joint)

def testBars(joint):
    "Test if the joints of the bars are defined."
    for b in bs:
        for j in b[1:]:
            if j not in joint:
                print("bar:", b[0], ": Joint", j, "not found")

def testJoints():
    "Test if the joints link at least 2 bars."
    bn = {}
    for b in bs:
        for j in b[1:]:
            if j not in bn:
                bn[j] = 0
            bn[j] += 1
    for j in bn:
        if bn[j] < 2:
            print("Joint", j, "links less than 2 bars")

def barLength(joint):
    "Compute the length of all bars."
    for b in bs:
        xa, ya = joint[b[1]]
        xt, yt = joint[b[2]]
        print("Bar", b[0], ": length=", hypot(xt-xa, yt-ya))

pyMain()
```

Λύση 2

Το λεξικό joint μπορεί να δημιουργηθεί από μία λίστα με ζεύγη (πλειάδες) κλειδιού, τιμής. Η λίστα μπορεί να γίνει με comprehension. Επίσης στη testJoints() στο λεξικό nb ελέγχουμε αν το κλειδί j υπάρχει μέσα του και αν δεν υπάρχει τότε το βάζουμε με τιμή 0. Αυτό ακριβώς κάνει και η μέθοδος setdefault().

```
def pyMain():
    "Transform coordinates to dictionary and call tests."
    temp = [ (j[0], j[1:]) for j in js ]
    joint = dict(temp)
```

```
    testBars(joint)
    testJoints()
    barLength(joint)


def testJoints():
    "Test if the joints link at least 2 bars."
    bn = {}
    for b in bs:
        for j in b[1:]:
            bn.setdefault(j, 0)
            bn[j] += 1
    for j in bn:
        if bn[j] < 2:
            print("Joint", j, "links less than 2 bars")
```

Λύση 3

Το λεξικό joint μπορεί να δημιουργηθεί comprehension λεξικού. Επίσης στη testJoints() η μέθοδος setdefault() επιστρέφει την τομή που δίνουμε στο κλειδί j και συνεπώς μπορεί να χρησιμοποιηθεί σε αριθμητικές παραστάσεις. Τέλος στο if αν οι εντολές που εκτελούνται υπό συνθήκη δεν είναι πολλές αλλά είναι μόνο μία, μπορούμε να την βάλουμε δεξιά από το if αντί για από κάτω.

```
def pyMain():
    "Transform coordinates to dictionary and call tests."
    joint = { j[0]: j[1:] for j in js}
    testBars(joint)
    testJoints()
    barLength(joint)

def testBars(joint):
    "Test if the joints of the bars are defined."
    for b in bs:
        for j in b[1:]:
            if j not in joint: print("bar:", b[0], ": Joint", j, "not found")

def testJoints():
    "Test if the joints link at least 2 bars."
    bn = {}
    for b in bs:
        for j in b[1:]: bn[j] = bn.setdefault(j, 0) + 1
    for j in bn:
        if bn[j] < 2: print("Joint", j, "links less than 2 bars")
```