Υπολογιστική Γεωμετρία

Εργασία 1

Βαμβουρέλλης Ευστράτιος 1115201600014
Αποστόλου Φιλιππος 1115201600007

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

2019-2020

Contents

1. Exercise 2

Εκφώνηση: Given a circle of radius r in the plane with (0, 0) as center, implement an algorithm that finds the total lattice points on the circumference. Lattice Points are points with integer coordinates.

Η πρώτη σκέψη ήταν να κάνουμε έναν εξαντλητικό αλγόριθμο που θα ελέγχει για όλα τα $(x, y), \quad x \equiv int$ που ανήκουν στον κύκλο ποια από αυτά έχουν $y \equiv int$.
Μια πολύ απλή βελτίωση είναι να ψάχνουμε (x, y) μόνο στο πρώτο τεταρτημόριο. Προφανώς κάθε φορά που βρίσκουμε ένα ζευγάρι που ικανοποιεί την συνθήκη θα το μετράμε σαν τέσσερα $\Big((x, y), \quad (-x, y), \quad (x, -y), \quad (-x, -y)\Big)$.
Ο παραπάνω αλγόριθμος έχει πολυπλοκότητα $O\Big(floor(r)\Big)$.
Παρακάτω είναι ο κώδικας σε python3 (δεν έχουμε ψευδοκώδικα καθώς ο αλγόριθμος είναι πολύ απλός):

```python
import math

# count Lattice points
def countLatticePoints(r):
    if r <= 0:  # error
        return 0
    result=0
    # Check every int value
    for x in range(1, int(r)+1):  # range inclusive, r rounded down
        y = math.sqrt(r*r-x*x)  # Find potential y
        if y.is_integer():
            result += 4  # one for each quadrant
    return result


# ----------------main--------------------
r=float(input("enter r for ((0,0),r) circle: "))
print("total lattice points: "+str(countLatticePoints(r)))
```

Για να εκτελέσουμε τον παραπάνω κώδικα χρησιμοποιούμε την εντολή python3 q2.py, πληκτρολογούμε έναν αριθμό και πατάμε enter.
Ένα παράδειγμα εκτέλεσης είναι:

```
enter r for ((0,0),r) circle: 5
total lattice points: 12
```

2. Exercise 3

Εκφώνηση: Implement the incremental 2D algorithm for computing the convex hull of a finite set of points in the plane.

Ο αλγόριθμος που υλοποιώ μοιάζει πολύ με τον beneath-beyond που εξηγήσαμε στο μάθημα. Η μόνη ιδιαιτερότητα είναι η δομή που χρησιμοποιούμε για την αποθήκευση των στοιχείων του convex hull. Η λογική του beneath-beyond είναι η παρακάτω:

- Ταξινομούμε τα σημεία μας με βάση το x.
- Ξεκινάμε με τα 3, πρώτα, μη συνευθειακά σημεία (που κάνουν ένα τρίγωνο).
- Επαναλαμβάνουμε τα παρακάτω για κάθε άλλο, επόμενο, σημείο:
  - προσθέτουμε το σημείο στο περίβλημα
  - αφαιρούμε όλες τις ακμές που δεν πρέπει να υπάρχουν στο τρέχον περίβλημα. Δηλαδή όλες τις ακμές με σημεία που βρίσκονται αυτή τη στιγμή μέσα στο τρέχον περίβλημα (ο έλεγχος γίνεται με ccw).

Στην πραγματικότητα την ιδέα, για την δομή που χρησιμοποιήσαμε, την πήραμε από έναν άλλο αλγόριθμο, Melkman Algorithm, 1987. Η δομή αυτή ονομάζετε deque και είναι μια διπλά συνδεδεμένη ουρά με 2 κεφαλές (μια στην αρχή και μια στο τέλος). Στην python η δομή αυτή βρίσκεται στο collections.deque. Διαλέξαμε αυτή τη δομή γιατί η εισαγωγή και η εξαγωγή στοιχείων από την αρχή και το τέλος γίνεται σε σταθερό χρόνο. Παρακάτω εξηγούμε πιο αναλυτικά, με ψευδοκώδικα, τον αλγόριθμο beneath-beyond χρησιμοποιώντας την δομή deque.

- Ταξινομούμε τα σημεία μας με βάση το x στον πίνακα P[n].
- Αποθηκεύουμε τα 3, πρώτα, μη συνευθειακά σημεία, στην deque D[4], έτσι ώστε και την αρχή D[top] και στο τέλος D[bot] να υπάρχει το πιο ακριανό (αυτό με το μεγαλύτερο i στο P[i])σημείο και τα σημεία να σχηματίζουν ένα ccw τρίγωνο. πχ P[2] P[0] P[1] P[2].
- Για κάθε σημείο P[i] επαναλαμβάνουμε:
  Όσο το P[i] είναι δεξιά του τμήματος D[bot]D[bot-1] επαναλαμβάνουμε:
  Διαγράφουμε το D[bot]
  Εισάγουμε το P[i] στο D[bot+1]
  Όσο το P[i] είναι δεξιά του τμήματος D[top]D[top+1] επαναλαμβάνουμε:
  Διαγράφουμε το D[top]
  Εισάγουμε το P[i] στο D[top+1]

Παρακάτω είναι ο κώδικας σε python3, η πολυπλοκότητά του είναι $O\Big(n \log n\Big)$:

```
from collections import deque
import numpy as np
import sys


# returns ccw
def ccw(A: np.array, B: np.array, C: np.array) -> float:
    # Tests whether the turn formed by A, B, and C is ccw
    return (B['x'] - A['x']) * (C['y'] - A['y']) - (B['y'] - A['y'])
                * (C['x'] - A['x'])


def convexHull2d(points_input: np.array) -> np.array:
    # sort points declining
    points=np.sort(points_input, kind='mergesort', order=['x','y'])

    # find the last non collinear point to the first two
    last_non_inline=2
    while ccw(points[0], points[1],points[last_non_inline]) == 0:
        last_non_inline += 1
        if last_non_inline == len(points):
            return np.empty(shape=(0, 0))

    # initialise deque with
    if ccw(points[0], points[last_non_inline-1], points[last_non_inline]) > 0:
        que = deque([points[last_non_inline], points[0], points[last_non_inline-1],
                     points[last_non_inline]])
    elif ccw(points[0], points[last_non_inline-1], points[last_non_inline]) < 0:
        que = deque([points[last_non_inline], points[last_non_inline-1], points[0],
                     points[last_non_inline]])
    else:
        print("error")
        return []

    for i in range(last_non_inline, len(points)):
        # remove all bottom points inside o (soon to be) polygon
        while ccw(que[-2], que[-1], points[i]) <= 0:
            que.pop()
        que.append(points[i])

        # remove all top points inside our (soon to be) polygon
        while ccw(que[0], que[1], points[i]) <= 0:
            que.popleft()
        que.appendleft(points[i])
```

```
    # reform output (not really needed)
    ans=np.empty(shape=(que.__len__()-1), dtype=[('x', float), ('y', float)])
    for i in range(0,que.__len__()-1):
        ans[i]=que.pop()
    return ans

# ------------------------------------main----------------------

# read from file
file_name = sys.stdin.readline()
inputt = []
with open(file_name[:-1]) as myfile:
    for line in myfile:
        x, y = line.partition(" ")[::2]
        inputt.append((float(x), float(y)))

input = np.array(inputt, dtype=[('x', float), ('y', float)])
ans=convexHull2d(input)
for i in ans:
    print(i)
```

Για να εκτελέσουμε τον παραπάνω κώδικα χρησιμοποιούμε την εντολή python3 q3.py, πληκτρολογούμε το relevant path το αρχείο με τα δεδομένα και πατάμε enter. Στην ιδική περίπτωση που έχουμε ως είσοδο μόνο συνευθειακά σημεία το αποτέλεσμα είναι το κενό.
Το αρχείο με τα δεδομένα πρέπει να είναι της παρακάτω μορφής: αριθμός κενό αριθμός αλλαγή γραμμής. Στο τέλος του αρχείου δεν πρέπει να υπάρχει αλλαγή γραμμής.
Ένα παράδειγμα εκτέλεσης είναι:

```
./in_out/input2.txt
(7., 0.75)
(7., 0.5)
(5., -1.)
(1., -1.)
(0., 0.)
(1., 1.)
(6., 1.)
```

Συμπληρωματικά έχουμε κάνει και ένα testing script. Μαζί έχουμε παραδώσει και έναν φάκελο in_out με ενδεικτικά δεδομένα και αποτελέσματα. Μπορείτε να εκτελέσετε το πρόγραμμα για όλα τα ενδεικτικά δεδομένα με το testing script. Για να το κάνετε αυτό γράφεται python test.py. Αν εκτελεστούν όλα τα τεστ με επιτυχία θα έχετε την παρακάτω έξοδο:

```
!! SUCCESS !!
```

Αλλιώς θα έχετε μια έξοδο του παρόμοια με την παρακάτω:
MISMATCH IN: ./in_out/input2.txt

3. Exercise 4

Εκφώνηση: Implement the gift wrap algorithm for computing the convex hull of a finite set of points in the plane

Βασική ιδέα είναι ότι αρχίζουμε από ένα σημείο που ανήκει στο κυρτό περίβλημα και έπειτα βρίσκουμε κάθε φορά το δεξιότερο του σημείο στο σύνολο και τα ενώνουμε με μία ακμή και γενικά αυτό μοιάζει με περιτύλιξη δώρου προς τα δεξιά.Ως υπόθεση παρακάτω έχω ότι δεν υπάρχουν συνευθειακά σημεία

Παρακάτω είναι ο κώδικας σε python3, η πολυπλοκότητά του είναι $O\left(n * h\right)$:

```
import ast

def CCW(p1,p2,p3):
  if (p3[1]-p1[1])*(p2[0]-p1[0]) > (p2[1]-p1[1])*(p3[0]-p1[0]):
    return True
  return False

def GiftWrapping(Points):
  last_hull_point=min(a)            # το ελάχιστο σημειο ανηκει στο Hull
  Hull=[]                           # Αρχικοποίηση το hull ως κενό σύνολο
  while True:
    Hull.append(last_hull_point)    # βαζει το last_Hull_point που βρέθηκε
    point_checking=Points[0]        # Διαλέγετε ένα σημείο τυχαία ως υποψήφιο hall point
    n = len(Points)
    for j in range(1,n):        # Ψάχνει το πιο δεξιό το σημείο από το last point
      if not CCW(Points[j],Hull[-1],point_checking):
        point_checking = Points[j]
    last_hull_point = point_checking
    if last_hull_point in Points:   # Points-H Κάθε φορά που βρίσκει new point
      Points.remove(last_hull_point)
    if point_checking == Hull[0]:   #Το Hull πρέπει να κλείσει στο αρχικό σημείο
      break
  return Hull

# --------------------main----------------------
```

```
a = ast.literal_eval(input('points: '))
h=GiftWrapping(a)
for x in range(len(h)):
    print(h[x])
```

Για να εκτελέσουμε τον παραπάνω κώδικα χρησιμοποιούμε την εντολή python3 q4.py και τα σημεία πρέπει να δοθούν από την είσοδο σε αυτό το format [(,),(,),...,(,)]
Ένα παράδειγμα εκτέλεσης είναι:

```
points:[(1,1),(-1,1),(0,-1)]
(1,1)
(-1,1)
(0,-1)
```

Προτάσεις

1) αν το $ccw(p_0, p_1, p_2)$ είναι θετικό τότε το p_2 είναι δεξιότερα του p_3, αν είναι αρνητικό τότε το αντίστροφο κι αν είναι ίσον με το μηδέν τότε όλα τα σημεία είναι συνευθειακά.

Απόδειξη: Μπορώ να υποθέσω τα εξής διανύσματα $v_1 = (x_1 - x_0, y_1 - y_0), v_2 = (x_2 - x_0, y_2 - y_0)$ στον δισδιάστατο χώρο για τα οποία θέλω να δω ποιο είναι δεξιότερα.Θα πρέπει να χρησιμοποιήσω εξωτερικό γινόμενο ,για αυτό τον λόγο ανάγω τα διανύσματα μου στον τρισδιάστατο χώρο$(x_1 - x_0, y_1 - y_0, 0), (x_2 - x_0, y_2 - y_0, 0)$καθώς το αποτέλεσμα το εξωτερικού γινομένου είναι διάνυσμα όπου το μέτρο του είναι το εμβαδόν του παραλληλόγραμμου πού σχηματίζεται από τα v_1 και v_2.

$$v1 \times v2 = \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix} = (x_1 - x_0) * (y_2 - y_0) - (y_1 - y_0) * (x_2 - x_0)$$

Τέλος το πρόσημο του εξωτερικού γινομένου μας δίνει σύμφωνα με τον κανόνα του δεξιού χεριού ποιο διάνυσμα είναι πιο δεξιά και στην περίπτωση που όλα είναι συνευθειακά εμβαδόν του παραλληλόγραμμου των δύο διανυσμάτων είναι μηδέν.

2) το αριστερότερο το σημείο από το σύνολο των σημείων που μας δίνετε ανήκει στο convex hull,στην περίπτωσή 2 ή περισσότερων σημείων το αριστερότερο αυτό με το μικρότερο y.

Απόδειξη:Έστω m το ελάχιστο σημείο του συνόλου P ισχύει :$x_m \leq x_p, \forall p \in P$ Έστω ότι το m είναι εσωτερικό σημείο του convex hull τότε είναι δεξιότερα του αριστερότερου σημείου στο convex hull $h < x_m, h \in H$, όμως τα σημεία του convex hull είναι και σημεία του P ,$h \geq x_m, h \in P$, άρα άτοπο.

3) στο τέλος της επανάληψης του for j in range(1,n) το point-checking = είναι το δεξιότερο σημείο απο τα $points[i], i \in 0, 1, 2, ..., j$ από την προοπτική του προηγούμενου σημείου πού ανήκει στο convex hull .

Απόδειξη:Με επαγωγή στο j Βήμα Βάσης: Για $j = 1$ CCW(Points[1],Hull[-1],point checking) οπού point-checking = points[0], Hull[-1] είναι το τελευταίο σημείο που βρήκαμε. Αν είναι το point-checking πιο δεξιά τότε είναι το πιο δεξιά σημείο από το σύνολο των points[i] με $i \in (0, 1)$ αλλιώς point-checking = Points[1] άρα πάλι ισχύει

Επαγωγική υπόθεσή: έστω ότι ισχύει για $1 < k < n - 1$ δηλαδή το point-checking είναι το δεξιότερο σημείο από το σύνολο σημείων points[i] με $i \in 0, ..., k$

Επαγωγικό βήμα: θα δείξω την πρόταση για $j = k + 1$ CCW(Points[k+1],Hull[-1],point checking) αν είναι αρνητικό τότε μπορώ να προσθέσω το point-checking=points[k+1] στο σύνολο ως το πλέον δεξιότερο,αλλιώς το πρόσθετο στο σύνολο κι το point checking είναι ακόμα το δεξιότερο.

4)αν έχουμε βρει όλα τα σημεία του convex hull τότε στην τελευταία επανάληψη το last-hull-point=point-checking θα είναι ίσο με Hull[0]

Απόδειξη: Έστω H_0 το πρώτο σημείο του convex hull και έστω H_k τελευταίο. Έστω τώρα ότι το point checking δεν είναι το H_0 και είναι κάποιο άλλο σημείο P τότε διακρίνονται οι εξής περιπτώσεις:$CCW(H_k, H_0, P) \leq 0$ τότε το P είναι δεξιότερα του H_0 ως προς το H_k και προφανώς όλων των άλλων σημείων λόγω της προηγούμενης πρότασης άρα είναι σημείο του convex hull άρα το H_k δεν είναι το τελευταίο σημείο του convex hull, άτοπο. Αν CCW(H_k,H_0,P)>0 τότε δεν είναι μέρος του κυρ του περιβλήματος διότι υπάρχει ένα σημείο που είναι πιο δεξιά από αυτό άρα άτοπο.

4. Exercise 1

Εκφώνηση : Implement an algorithm that takes as input three points in the plane. checks that they form a triangle and whether the interior of the triangle contains the origin (0, 0) or not.

Η αρχική ιδέα ήταν να πάρω την κλήση των ευθύγραμμων τμημάτων ΑΒ και ΑΓ δηλαδή τις δύο πλευρές του Τριγώνου και θέλω να δω αν το σημείο μηδέν είναι εσωτερικά της γωνίας ΒΑΓ , δηλαδή η κλήση του Τμήματος ΑΟ να είναι ανάμεσα στην κλήση των τμημάτων ΑΒ ,ΑΓ . Και τέλος Αρκεί να δείξω το ίδιο για μία άλλη γωνία του Τριγώνου. Όμως αυτό παρουσιάζει δυσκολίες καθώς η κλήση δεν ορίζεται αν δύο σημεία είναι κάθετα μεταξύ τους και επίσης δεν είναι ξεκάθαρο τι γίνεται όταν η κλήση των ΑΒ ,ΑΓ δεν είναι ομόσημοι. Η επόμενη λύση που σκέφτηκα περιλαμβάνει τον αλγόριθμο ccw πού χρησιμοποιείται στον αλγόριθμο του gift wrapping με την ιδέα ότι περνώντας τα σημεία Α Β Γ ανά δύο ακολουθώντας μία φορά πχ την Α->Β->Γ->Α τότε αλγόριθμος ccw των δύο σημείων και του μηδενός θα πρέπει να έχει το ίδιο πρόσημο και λέω το ίδιο πρόσημο γιατί εξαρτάται από τον τρόπο που προσπελαύνω τις γωνίες του Τριγώνου.

Παρακάτω είναι ο κώδικας σε python3, η πολυπλοκότητά του είναι $O\left(1\right)$:

```
import ast

def CCW(p1,p2,p3):
  return (p3[1]-p1[1])*(p2[0]-p1[0])-(p2[1]-p1[1])*(p3[0]-p1[0])

# --------------------main----------------------

points = ast.literal_eval(input('points: '))
if(CCW(points[0],points[1],points[2]) == 0):
  print("points are collinear there is no tringle")

rot1 = CCW(points[0],points[1],(0,0))
rot2 = CCW(points[1],points[2],(0,0))
rot3 = CCW(points[2],points[0],(0,0))

if(rot1*rot2 >= 0 and rot2*rot3 >=0): # Ισοδύναμο με οι τρεις αριθμοί είναι ομόσημοι
  print("zero is inside the tringle")
else:
  print("zero is outside of the tringle")
```

Για να εκτελέσουμε τον παραπάνω κώδικα χρησιμοποιούμε την εντολή python3 q1.py και τα σημεία πρέπει να δοθούν από την είσοδο σε αυτό το format [(,),(,),...,(,)]
Ένα παράδειγμα εκτέλεσης είναι:

```
points:[(1,1),(-1,1),(0,-1)]
zero is inside the tringle
```

Προτάσεις

5) ως πόρισμα του **(5)** για κάθε δύο σημεία του κυρτού περιβλήματος πού βρέθηκαν διαδοχικά ισχύει ότι CCW(h_i,h_{i+1},p)≥ 0 για κάθε σημείο p που ανήκει στο εσωτερικό του περιβλήματος.

6) αν η αρχή των αξόνων είναι μέσα στο Τρίγωνο τότε όλα τα ccw(p1,p2,O),ccw(p2,p3,O),ccw(p3,p1,O) είναι ομοσημα

Απόδειξη: Δεδομένων των τεσσάρων σημείων το τρίγωνο σχηματίζει ένα convex hull αν το σημείο $(0, 0)$ είναι εσωτερικό του Τριγώνου. Τότε σύμφωνα με την **πρόταση 5** αν παίρνω ανά δύο τις κορυφές του Τριγώνου και δεξιόστροφα μαζί με οποιοδήποτε σημείο μέσα στο Τρίγωνο θα έχει ccw θετικό. Υπάρχουν όμως δύο τρόποι όπου μπορούμε να προσπελάσουμε τρία σημεία α β γ αρχίζοντας πάντα από το α ,άρα δύο τρόποι για να προσπεράσουμε τους κόμβους ενός τριγώνου ,δεξιόστροφα και αριστερόστροφα καθώς δεν ξέρουμε ποια ακμή είναι δεξιά/αριστερά από το α. Οπότε ο ccw θα έχει αρνητικές τιμές αν προσπεράσουμε αριστερόστροφα αντιστρέφοντας την πρόταση 5 καθώς τα σημεία ανακαλύφθηκαν δεξιόστροφα σε αυτή την πρόταση.

5. Βιβλιογραφία

http://geomalgorithms.com/a12-_hull-3.html
http://www.ams.sunysb.edu/~jsbm/courses/345/13/melkman.pdf
https://docs.python.org/2/library/collections.html#collections.deque
https://wiki.python.org/moin/TimeComplexity
https://eclass.uoa.gr/modules/document/?course=D42