Υπολογιστική Γεωμετρία με χρήση της Python

```
from pycompgeom import *
```

Χριστόδουλος Φραγκουδάκης

ΕΜΠ - Κέντρο Υπολογιστών

Άνοιξη 2013

Περιεχόμενα (υπό κατασκευή)

Γιατί με χρήση της Python;

- Το πρόγραμμα “Hello World!” σε τρεις γλώσσες προγραμματισμού:

C++
```
#include<iostream.h>
using namespace std;
void main(){cout << "Hello world!" << end;}
```

Java
```
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello World!");}}
```

Python
```
print "Hello World!"
```

Γιατί με χρήση της Python;

- Σχεδόν μοιάζει με ψευδογλώσσα:

```
def factorial(num):
  return 1 if num==1 else num*factorial(num-1)

def symmetric_difference(set1, set2):
  return [x for x in set1 if x not in set2] +
    [x for x in set2 if x not in set1]
```

- “Οι μπαταρίες περιέχονται μέσα στη συσκευασία”:
  - εξαιρετική τεκμηρίωση μέσα στην ίδια τη γλώσσα,
  - πληθώρα ενσωματωμένων δομών δεδομένων,
  - πληθώρα βιβλιοθηκών για επιστημονικούς υπολογισμούς,
  - ενθουσιώδη κοινότητα.

Το στυλ της Python

- Duck Typing: *If it walks like a duck, swims like a duck and quacks like a duck, it must be a duck*.
- Δυναμικοί τύποι αντικειμένων: οι μέθοδοι και τα χαρακτηριστικά τους προσδιορίζουν *έγκυρη σημασιολογία* παρά κληρονομικότητα.
- *Όψη της χρήσης* vs *Τύπος του* αντικειμένου.
- Δεν ελέγχουμε τον τύπο αλλά καλούμε τις μεθόδους *walk swim* και *quack*: το αντικείμενο ανταποκρίνεται ή έχουμε run-time error.
- Exception handling: `try` $\cdots$ `except` blocks.
- EAFP: *Easier to Ask for Forgiveness than Permission*.

Duck Typing

Χρήση του ίδιου κώδικα για διαφορετικούς τύπους δεδομένων

```
def insertion_sort(sequence):
  for j in range(1, len(sequence)):
    key = sequence[j]
    i = j-1
    while i >= 0 and sequence[i] > key:
      sequence[i+1] = sequence[i]
      i = i-1
    sequence[i+1] = key
  return sequence

print insertion_sort([2,7,5,3])
print insertion_sort(['s','c','f','l','e','i'])
print insertion_sort([('b',(2,1)),('a',(9,7)),('b',(1,1))])

# prints:
# [2, 3, 5, 7]
# ['c', 'e', 'f', 'i', 'l', 's']
# [('a', (9, 7)), ('b', (1, 1)), ('b', (2, 1))]
```

Σύνταξη της Python

- ```
  import math
  class Point2(object):
  →def __init__(self, x, y):
  →→self.x = x
  →→self.y = y
  →def __repr__(self):
  →→return "Point(%s, %s)" % (self.x, self.y)
  →def distance_to(self, other):
  →→return math.hypot(self.x - other.x, self.y - other.y)
  ```
- Το → είναι το `Tab` ή το `Space`. Δεν υπάρχουν `BEGIN END { }`.
- Block εντολών με επίπεδα indendation.
- `__init__` (constructor) και `__repr__` είναι ειδικά ονόματα.
- Οι μέθοδοι των στιγμιοτύπων έχουν πάντα πρώτο όρισμα το `self`.
- `self.x`, `self.y` είναι μεταβλητές των στιγμιοτύπων της κλάσης.
  ```

Visualization αλγορίθμων

- Ζεστό πεδίο έρευνας: `http://algoviz.org`
- Εργαλείο για τη διδασκαλία: εμπλουτίζει την ενόραση, ιδιαίτερα σε πολύπλοκες, οριακές ή εκφυλισμένες καταστάσεις.
- Διάφορα συστήματα visualization: BALSA, TANGO, Zeus, POLKA, Leonardo,CATAI, JSamba, JAWAA, JHAVÉ, TRAKLA2, Alice, AliceLive.
- Θεωρίες μάθησης, levels of engagement ...
- Συνήθως δεν υπάρχει ολιστική αντιμετώπιση αλλά μεμονωμένες εξειδικευμένες απόπειρες.

Visualization γεωμετρικών αλγορίθμων

- Για το visualization των γεωμετρικών αλγορίθμων ένας στόχος είναι η εγκαθίδρυση ενός *γεωμετρικού ιδιώματος* για την Python:
  - θα αντικαθιστά τον ψευδοκώδικα στα συγγράμματα,
  - θα εμπλουτίζει τον ψευδοκώδικα με δυνατότητα εκτέλεσης,
  - θα δίνει δυνατότητα οπτικής εισόδου και εξόδου,
  - *όμως θα υπολείπεται σημαντικά σε ταχύτητα εκτέλεσης*.
- Η μειωμένη ταχύτητα εκτέλεσης φαίνεται λογικό τίμημα για την αυτόματη εκτέλεση του ψευδοκώδικα.
- Η CGAL (`http://cgal.org`) είναι μια καταξιωμένη γεωμετρική βιβλιοθήκη σε C++ που πρέπει να αποτελεί σημείο αναφοράς.

Υπολογιστικό μοντέλο

Πραγματική (real) RAM (Random Access Machine):

- Ακριβής αναπαράσταση και αποθήκευση πραγματικών σε χώρο $O(1)$.
- Μοναδιαίος χρόνος για προσπέλαση μνήμης.
- Μοναδιαίος χρόνος και απόλυτη ακρίβεια για βασικές πράξεις στο $\mathbb{R}$

Μια ικανοποιητική υλοποίηση του μοντέλου είναι η γλώσσα προγραμματισμού Python.

Περιεχόμενα (υπό κατασκευή)

Ευκλείδιος χώρος

Διανύσματα και σημεία στο επίπεδο

- Εξετάζουμε γεωμετρικά αντικείμενα που είναι σύνολα σημείων του δισδιάστατου Ευκλείδιου χώρου.
- Τα σύνολα σημείων μπορεί να είναι άπειρα όμως πρέπει να είναι *πεπερασμένα ορίσιμα*.
- Ένα σημείο $p = (x, y)$ ταυτίζεται με το διάνυσμα $\overrightarrow{0p}$.

Αναπαράσταση σημείων

- Μια κλάση της Python που υλοποιεί στιγμιότυπα σημείων στο επίπεδο (περιέχεται στο αρχείο κειμένου `point.py`):

```
class Point2(object):
  def __init__(self, x, y):
    self.x = x
    self.y = y
  def __repr__(self):
    return "Point2(%s, %s)" % (self.x, self.y)
```

Αναπαράσταση σημείων

- Μια κλάση της Python που υλοποιεί στιγμιότυπα σημείων στο επίπεδο (περιέχεται στο αρχείο κειμένου `point.py`):

```
class Point2(object):
  def __init__(self, x, y):
    self.x = x
    self.y = y
  def __repr__(self):
    return "Point2(%s, %s)" % (self.x, self.y)
```

- Παράδειγμα χρήσης στον διερμηνέα:

```
>>> from point import Point2
>>> a, b = Point2(7.32, 0), Point2(0.12342, -10.23241)
>>> print a,b
Point2(7.32, 0) Point2(0.12342, -10.23241)
>>>
```

Αναπαράσταση σημείων

- Εμπλουτισμένη κλάση για τα σημεία:

```
class Point2(object):
  def __init__(self, x, y):
    self.x, self.y = x, y
  def __repr__(self):
    return "Point2(%s, %s)" % (self.x, self.y)
  @classmethod
  def from_point2(cls, point2):
    return cls(point2.x, point2.y)
  @classmethod
  def from_tuple(cls, tup):
    return cls(tup[0], tup[1])
  @property
  def coordinates(self):
    return self.x, self.y
  @coordinates.setter
  def coordinates(self, tup):
    self.x, self.y = tup[0], tup[1]
```

Αναπαράσταση σημείων

- Παράδειγμα χρήσης της εμπλουτισμένης κλάσης στον διερμηνέα:

```
>>> from point import Point2
>>> p1 = Point2(3.5,-4)
>>> p2 = Point2.fromPoint2(p1)
>>> p3 = Point2.fromTuple(p2.coordinates)
>>> print p1, p2, p3
Point2(3.5, -4) Point2(3.5, -4) Point2(3.5, -4)
>>>
```

Αναπαράσταση ευθυγράμμων τμημάτων

- Μια κλάση της Python που υλοποιεί στιγμιότυπα ευθυγράμμων τμημάτων στο επίπεδο:

```
class Segment2(object):
  def __init__(self, start, end):
    self.start = start
    self.end = end
  def __repr__(self):
    return "Segment2(%s, %s)" % (self.start, self.end)
```

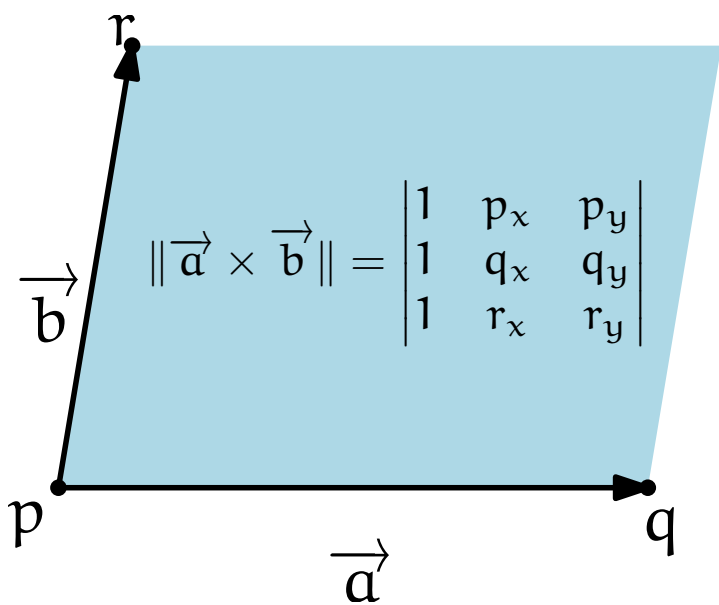
Αναπαράσταση ευθυγράμμων τμημάτων

- Παράδειγμα χρήσης στον διερμηνέα:

```
>>> from point import Point2
>>> from segment import Segment2
>>> p = Point2(1.2, 3.21)
>>> q = Point2(-2, 2.34)
>>> s = Segment2(p, q)
>>> print s
Segment2(Point2(1.2, 3.21), Point2(-2.0, 2.34))
```

Εμβαδό τριγώνου

- Από το σχολείο: $E = \frac{\text{βάση} \times \text{ύψος}}{2}$
- Το τρίγωνο δίνεται σαν μια τριάδα σημείων $p, q, r = (p_x, p_y), (q_x, q_y), (r_x, r_y)$
- Το εξωτερικό γινόμενο δύο διανυσμάτων έχει μέτρο ίσο με το εμβαδό του παραλληλογράμμου που ορίζουν.



Υπολογισμός του εμβαδού

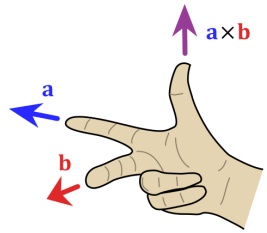
- Μια συνάρτηση Python που υπολογίζει το *διπλάσιο* του εμβαδού του τριγώνου pqr:

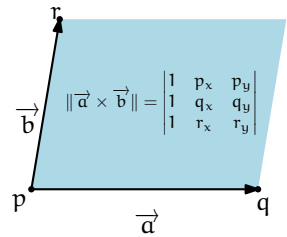
```
def area2(p, q, r):
  return (r.y-p.y) * (q.x-p.x) - (q.y-p.y) * (r.x-p.x)
```

- Παράδειγμα χρήσης στον διερμηνέα:

```
>>> from point import Point2
>>> from utilities import area2
>>> p = Point(0, 0)
>>> q = Point(3, 0)
>>> r = Point(0, 4)
>>> area2(p, q, r)
12.0
>>> area2(q, p, r)
-12.0
>>> area2(p, q, q)
0.0
>>>
```

Πρόσημο του εμβαδού





- Κανόνας του δεξιού χεριού.
- *Θετικό* εμβαδό στο τρίγωνο pqr σημαίνει ότι το διάνυσμα $\vec{a} = \overrightarrow{pq}$ πρέπει να στρίψει αριστερά για να βρει το $\vec{b} = \overrightarrow{pr}$.
- Με άλλα λόγια, θετικό εμβαδό στο τρίγωνο pqr σημαίνει ότι το σημείο r βρίσκεται στα *αριστερά* του $\overrightarrow{pq}$.
- *Αρνητικό* εμβαδό στο τρίγωνο qpr σημαίνει ότι το σημείο r βρίσκεται στα *δεξιά* του $\overrightarrow{qp}$
- *Μηδενικό* εμβαδό στο τρίγωνο pqr σημαίνει ότι τα σημεία p, q, r είναι *συνευθειακά*.

Κατηγορήματα Προσανατολισμού

- Συναρτήσεις της Python που υλοποιούν κατηγορήματα προσανατολισμού:

```
def ccw(p, q, r):
  return area2(p, q, r) > 0

def cw(p, q, r):
  return area2(p, q, r) < 0

def collinear(p, q, r):
  return area2(p, q, r) == 0

def between(p, q, r):
  if not collinear(p, q, r):
    return False
  if p.x != q.x:
    return p.x <= r.x <= q.x or p.x >= r.x >= q.x
  else:
    return p.y <= r.y <= q.y or p.y >= r.y >= q.y
```

Κατηγορήματα Προσανατολισμού

- Παράδειγμα χρήσης στον διερμηνέα:

```
>>> from point import Point2
>>> from predicates import *
>>> p = Point2(-0.00342324, 2.03424345)
>>> q = Point2(23.47029054, 3.34344444)
>>> r = Point2(-2.1, -23.00009389)
>>> ccw(p, q, r)
False
>>> cw(p, q, r)
True
>>> collinear(p, q, r)
False
>>> collinear(p, q, q)
True
>>> collinear(q, q, q)
True
```

Διάταξη σημείων

- Υλοποίηση λεξικογραφικής διάταξης για τα στιγμιότυπα της κλάσης `Point`:

```
class Point2(object):
    ...
  def __eq__(self, other):
    return (self.x, self.y) == (other.x, other.y)
  def __ne__(self, other):
    return (self.x, self.y) != (other.x, other.y)
  def __lt__(self, other):
    return (self.x, self.y) < (other.x, other.y)
  def __gt__(self, other):
    return (self.x, self.y) > (other.x, other.y)
  def __le__(self, other):
    return  (self.x, self.y) <= (other.x, other.y)
  def __ge__(self, other):
    return  (self.x, self.y) >= (other.x, other.y)
    ...
```

Διάταξη σημείων

- Παράδειγμα χρήσης στον διερμηνέα:

```
>>> from point import Point2
>>> p1 = Point2(3,4)
>>> p2 = Point2(3,3)
>>> p3 = Point2(2,4)
>>> p4 = Point2(2,2)
>>> p5 = Point2(0,4)
>>> point_list = [p1, p2, p3, p4, p5]
>>> for point in sorted(point_list):
...     print point
...
Point2(0, 4)
Point2(2, 2)
Point2(2, 4)
Point2(3, 3)
Point2(3, 4)
```

Περιεχόμενα (υπό κατασκευή)

Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.

Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.

Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.
- *Κυρτός συνδυασμός* των σημείων $x_1, \ldots, x_k$ είναι το άθροισμα $\alpha_1 x_1 + \cdots + \alpha_k x_k$ με $\alpha_i \geq 0 \; \forall i$ και $\alpha_1 + \cdots + \alpha_k = 1$

Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.
- *Κυρτός συνδυασμός* των σημείων $x_1, \ldots, x_k$ είναι το άθροισμα $\alpha_1 x_1 + \cdots + \alpha_k x_k$ με $\alpha_i \geq 0 \; \forall i$ και $\alpha_1 + \cdots + \alpha_k = 1$
- *Κυρτό περίβλημα ενός συνόλου σημείων* S είναι το σύνολο όλων των κυρτών συνδυασμών των στοιχείων του S.

Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.
- *Κυρτός συνδυασμός* των σημείων $x_1, \ldots, x_k$ είναι το άθροισμα $\alpha_1 x_1 + \cdots + \alpha_k x_k$ με $\alpha_i \geq 0 \; \forall i$ και $\alpha_1 + \cdots + \alpha_k = 1$
- *Κυρτό περίβλημα ενός συνόλου σημείων* S είναι το σύνολο όλων των κυρτών συνδυασμών των στοιχείων του S.
- $\cdots$ στις d διαστάσεις είναι το σύνολο όλων των κυρτών συνδυασμών από $d + 1$ (ή λιγότερα) σημεία του S

Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.
- *Κυρτός συνδυασμός* των σημείων $x_1, \ldots, x_k$ είναι το άθροισμα $\alpha_1 x_1 + \cdots + \alpha_k x_k$ με $\alpha_i \geq 0 \; \forall i$ και $\alpha_1 + \cdots + \alpha_k = 1$
- *Κυρτό περίβλημα ενός συνόλου σημείων* S είναι το σύνολο όλων των κυρτών συνδυασμών των στοιχείων του S.
- $\cdots$ στις d διαστάσεις είναι το σύνολο όλων των κυρτών συνδυασμών από $d + 1$ (ή λιγότερα) σημεία του S
- $\cdots$ είναι η τομή όλων των *κυρτών* συνόλων που περιέχουν το S.

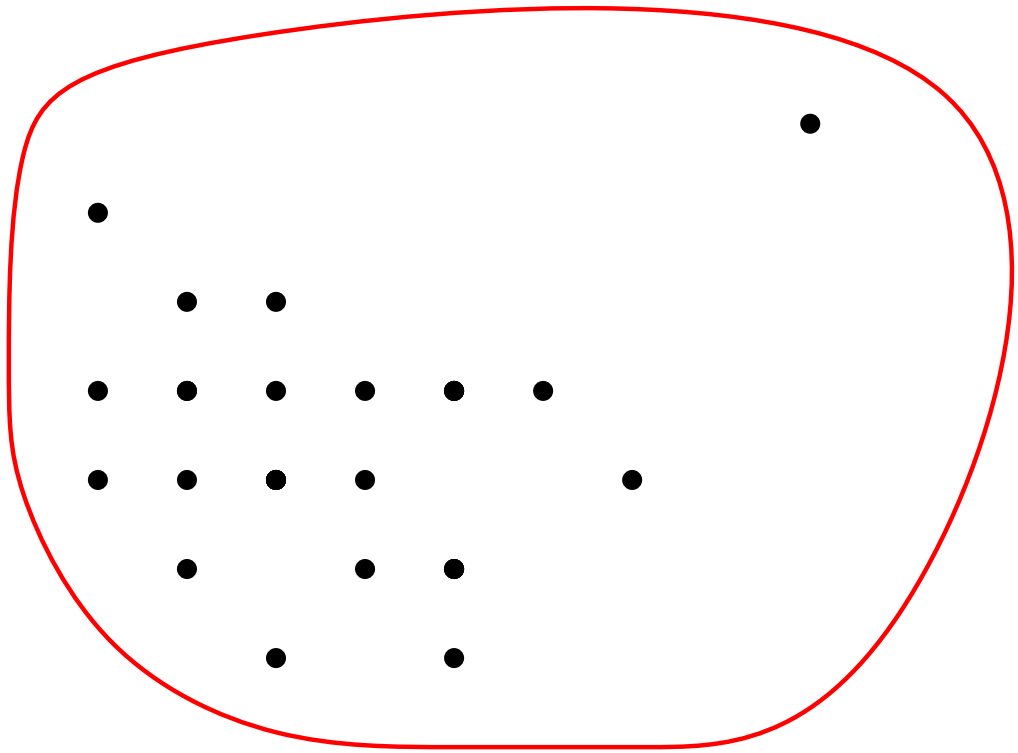
Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.
- *Κυρτός συνδυασμός* των σημείων $x_1, \ldots, x_k$ είναι το άθροισμα $\alpha_1 x_1 + \cdots + \alpha_k x_k$ με $\alpha_i \geq 0 \; \forall i$ και $\alpha_1 + \cdots + \alpha_k = 1$
- *Κυρτό περίβλημα ενός συνόλου σημείων* S είναι το σύνολο όλων των κυρτών συνδυασμών των στοιχείων του S.
- $\cdots$ στις d διαστάσεις είναι το σύνολο όλων των κυρτών συνδυασμών από $d + 1$ (ή λιγότερα) σημεία του S
- $\cdots$ είναι η τομή όλων των *κυρτών* συνόλων που περιέχουν το S.
- $\cdots$ είναι η τομή όλων των ημιχώρων που περιέχουν το S.

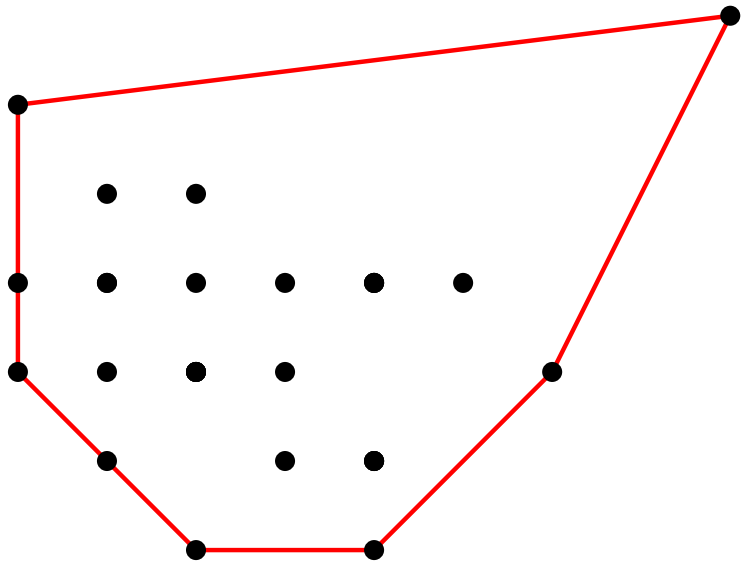
Κυρτότητα και Κυρτό Περίβλημα

- Ένα σύνολο S είναι *κυρτό* αν $x \in S$ και $y \in S$ τότε το *τμήμα* $xy \subseteq S$.
- Το *τμήμα* xy είναι το σύνολο όλων των σημείων $\alpha x + \beta y$ με $\alpha, \beta \geq 0$ και $\alpha + \beta = 1$.
- *Κυρτός συνδυασμός* των σημείων $x_1, \ldots, x_k$ είναι το άθροισμα $\alpha_1 x_1 + \cdots + \alpha_k x_k$ με $\alpha_i \geq 0 \; \forall i$ και $\alpha_1 + \cdots + \alpha_k = 1$
- *Κυρτό περίβλημα ενός συνόλου σημείων* S είναι το σύνολο όλων των κυρτών συνδυασμών των στοιχείων του S.
- $\cdots$ στις d διαστάσεις είναι το σύνολο όλων των κυρτών συνδυασμών από $d + 1$ (ή λιγότερα) σημεία του S
- $\cdots$ είναι η τομή όλων των *κυρτών* συνόλων που περιέχουν το S.
- $\cdots$ είναι η τομή όλων των ημιχώρων που περιέχουν το S.
- $\cdots$ είναι το *μικρότερο κυρτό σύνολο* που περιέχει το S.

Κυρτό Περίβλημα



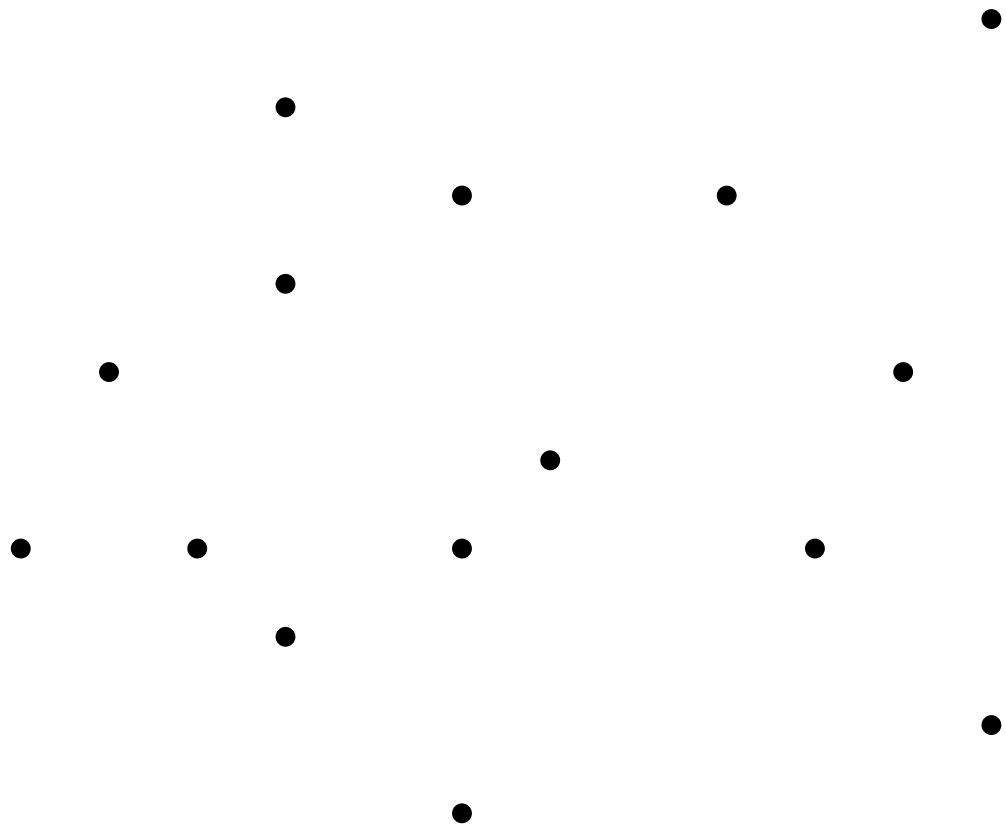
Κυρτό Περίβλημα



Περιεχόμενα

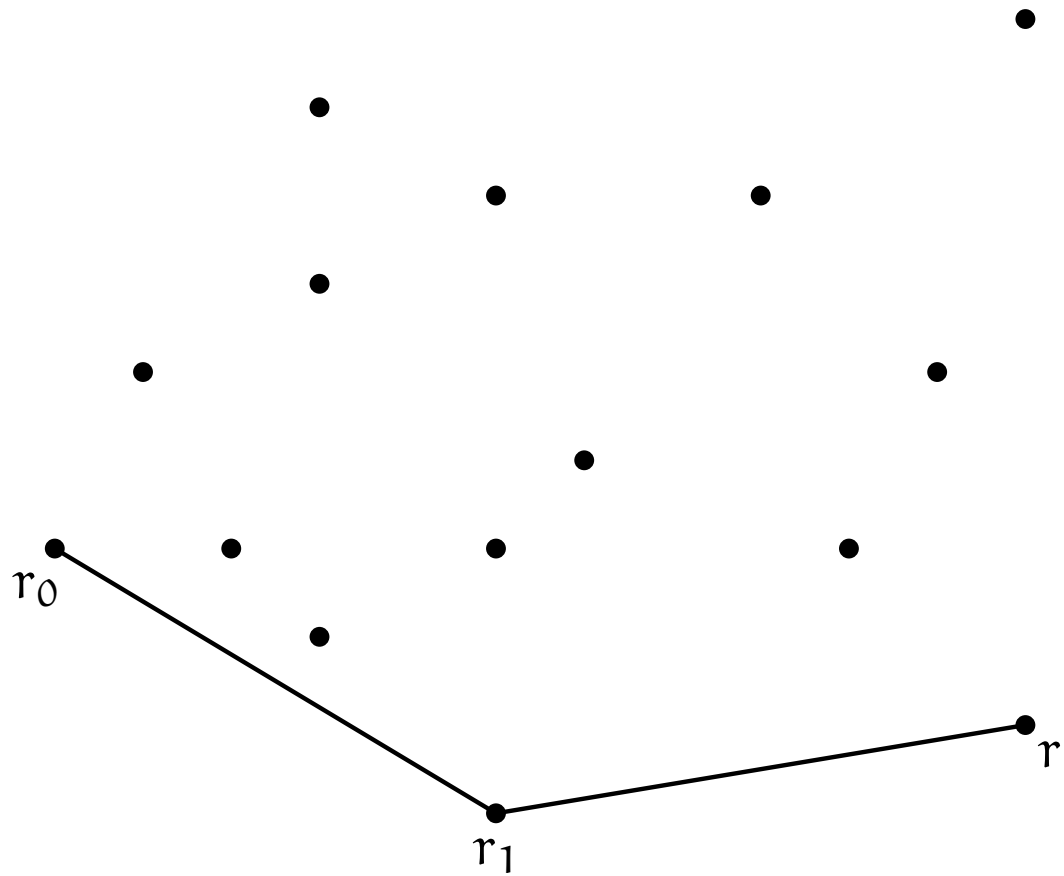
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



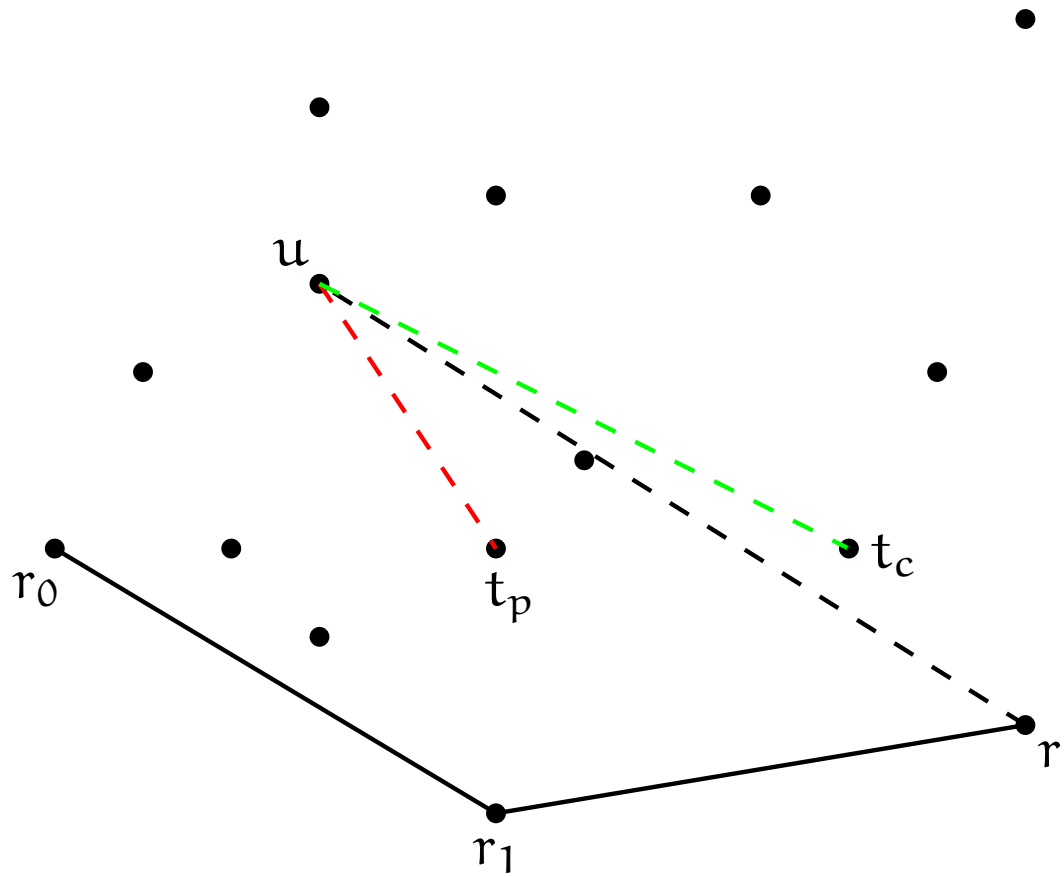
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



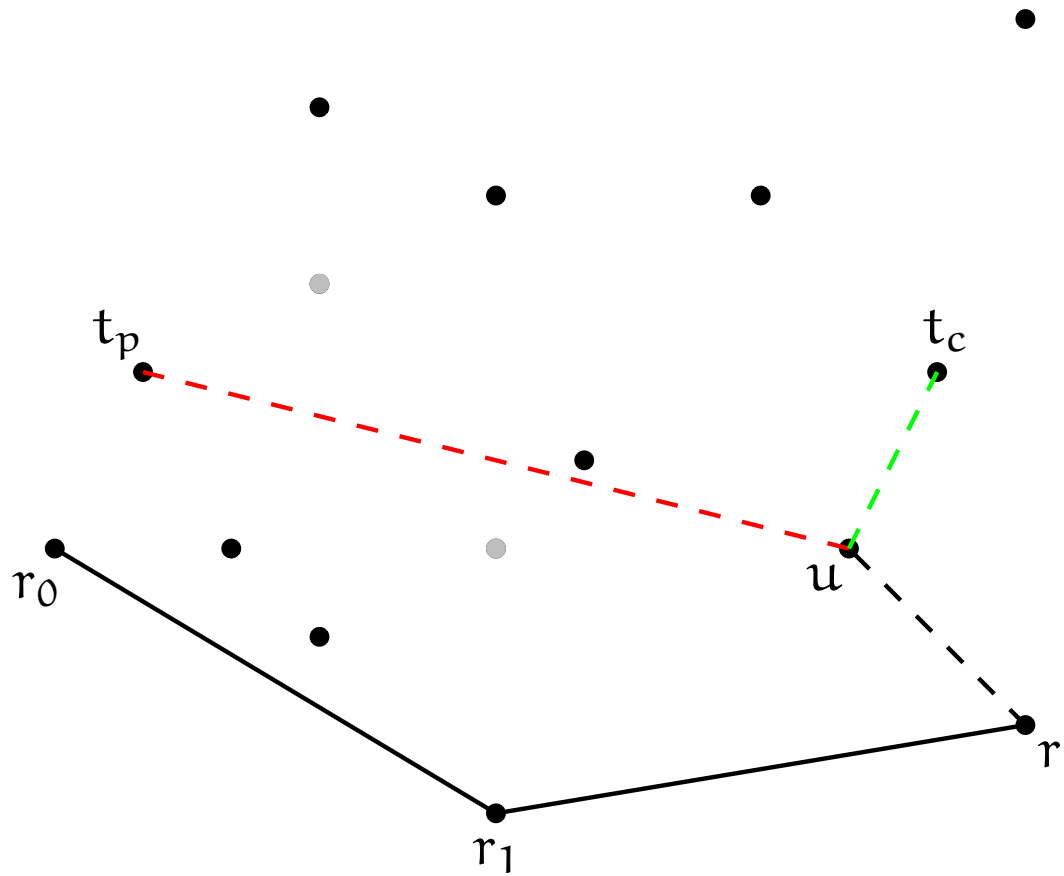
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



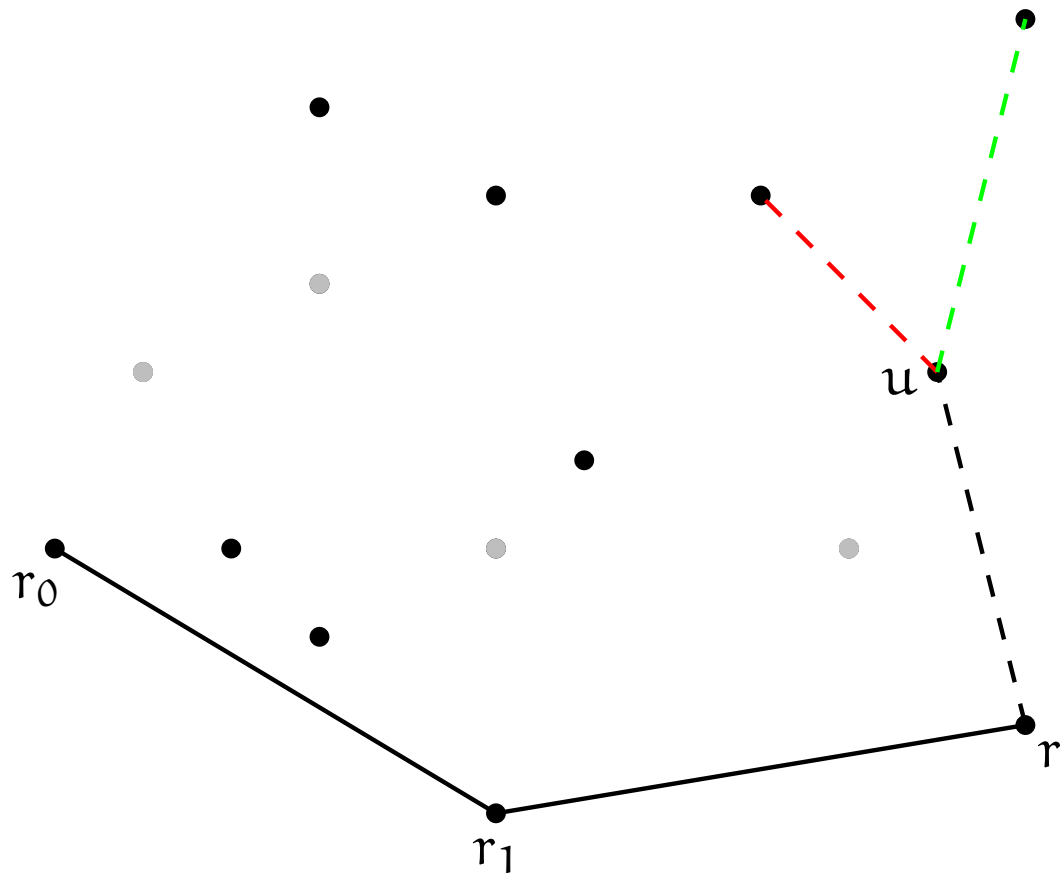
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



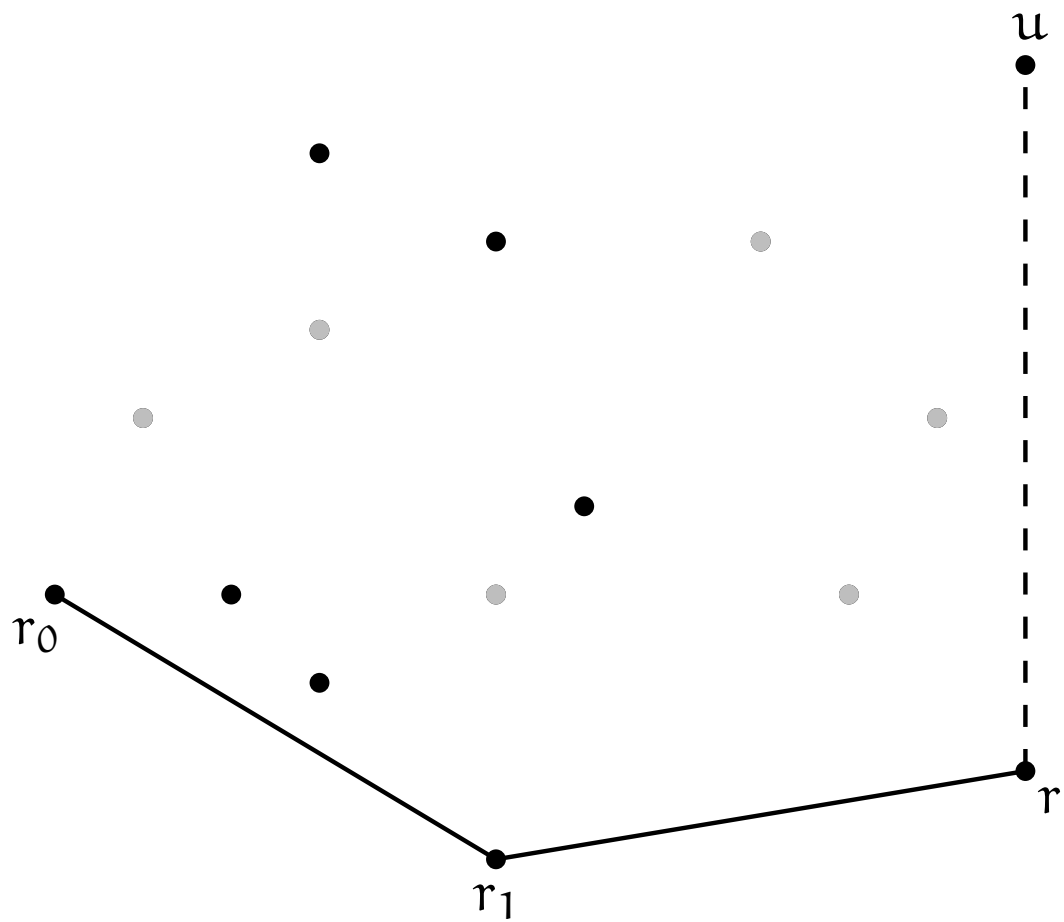
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



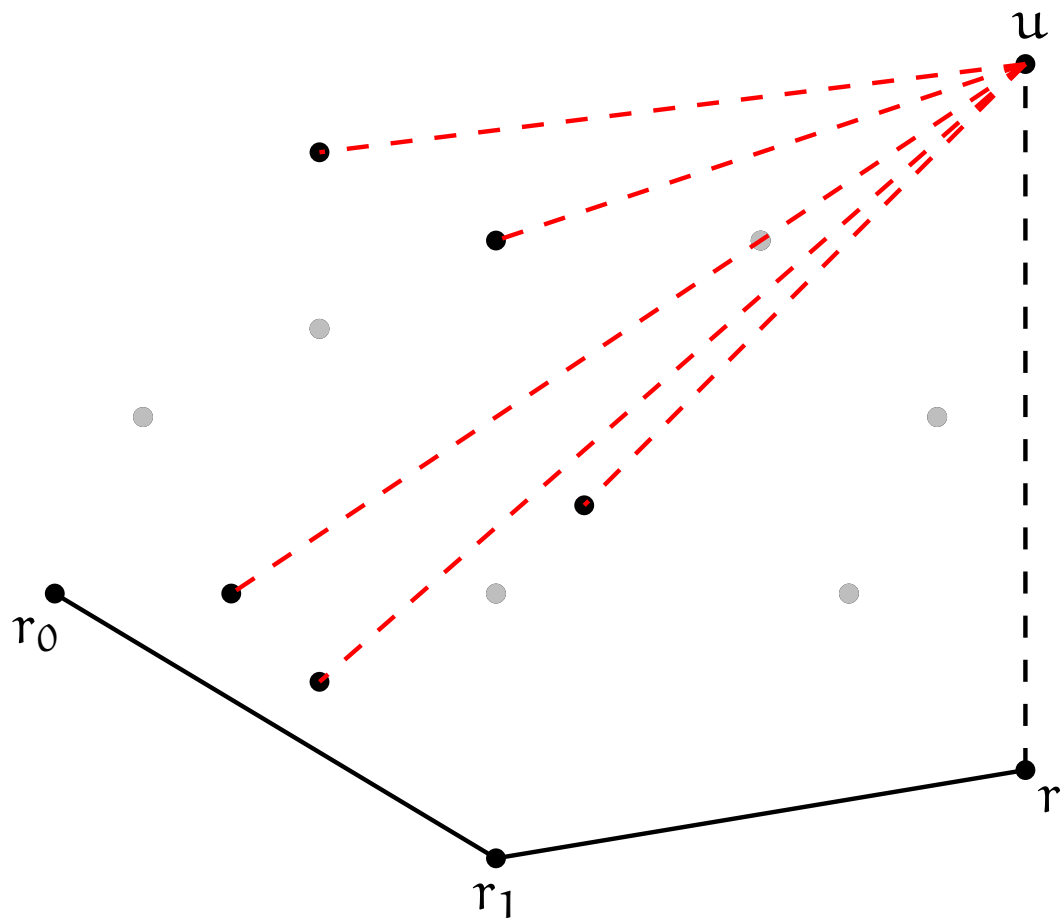
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



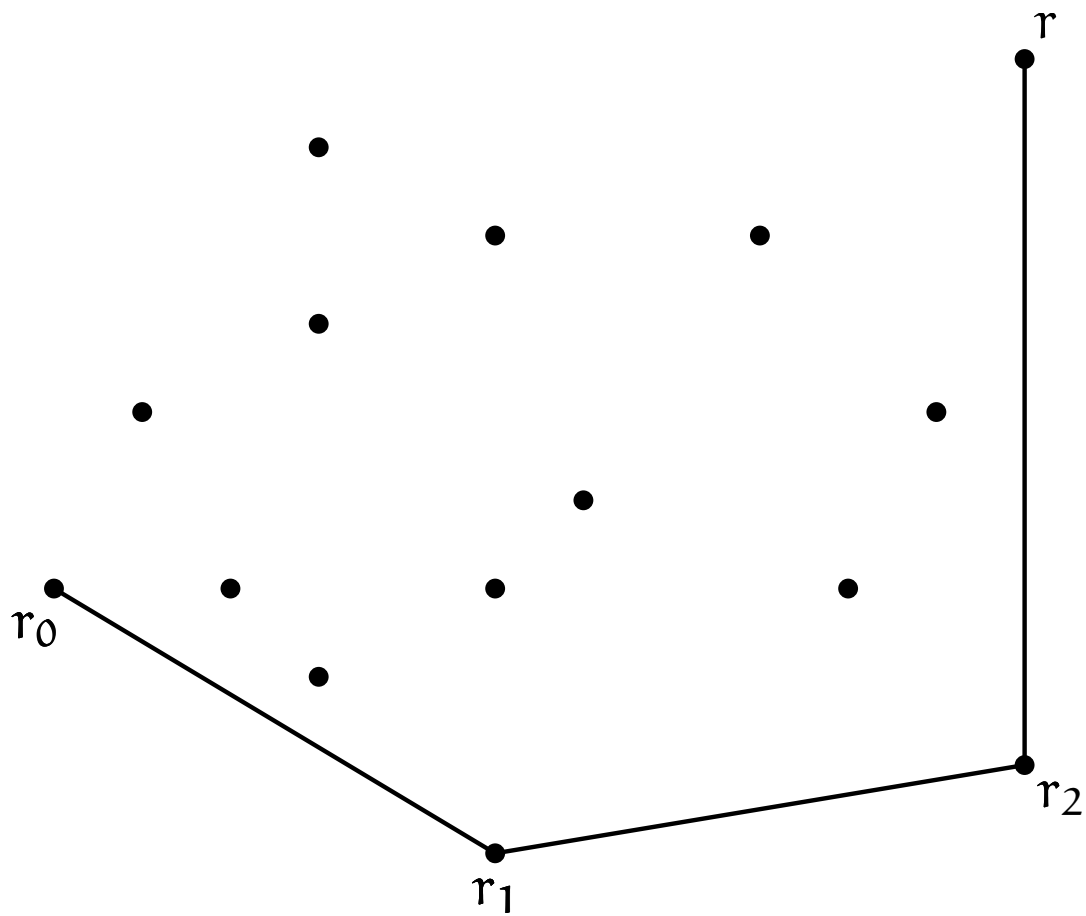
Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



Αλγόριθμος του Jarvis

Εύρεση επόμενης κορυφής



Αλγόριθμος του Jarvis

Υλοποίηση στην Python

```
from pycompgeom.predicates import *
import random

def jarvis(points):
  r = r0 = min(points)
  hull = [r0]
  u = None
  while u <> r0:
    u = random.choice(points)
    for t in points:
      if cw(r,u,t) or collinear(r,u,t) and between(r,t,u):
        u = t
    r = u
    points.remove(r)
    hull.append(r)
  return hull
```

Αλγόριθμος του Jarvis

Σύγκριση αλγόριθμου και υλοποίησης

Αλγόριθμος

```
Είσοδος: Σύνολο S αποτελούμενο από n σημεία στο επίπεδο.
Έξοδος: Η αλυσίδα των ακμών και των κορυφών του ΚΠ2.


1. Τρέχουσα κορυφή r=r0 είναι το "μικρότερο" σημείο.

2. Αρχικοποίηση αλυσίδας κορυφών με r. S=S-{r}.


3. Έστω r η τρέχουσα κορυφή και u που ανήκει στο S ένα
   οποιοδήποτε σημείο που δεν έχει επιλεγεί ως κορυφή.

   Για κάθε σημείο t που ανήκει στο S-{u}:
      αν ισχύει CW(r,u,t),
      ή αν r,u,t συνευθειακά και u εσωτερικό του (r,t)
   τότε θέσε u=t

4. Αν u=r0 τερμάτισε, αλλιώς
      r=u, S=S-{r},
      πρόσθεσε στην αλυσίδα των κορυφών το r,
      συνέχισε στο βήμα 3.
```

Υλοποίηση στην Python

```
from pycompgeom import *
import random
def jarvis(S):

  r = r0 = min(S)

  hull = [r0]
  u = None

  while u <> r0:
      u = random.choice(S)

      for t in S:
          if cw(r,u,t) or \
            collinear(r,u,t) and between(r,t,u):
            u = t


      r = u
      S.remove(r)
      hull.append(r)

  return hull
```

Περιεχόμενα

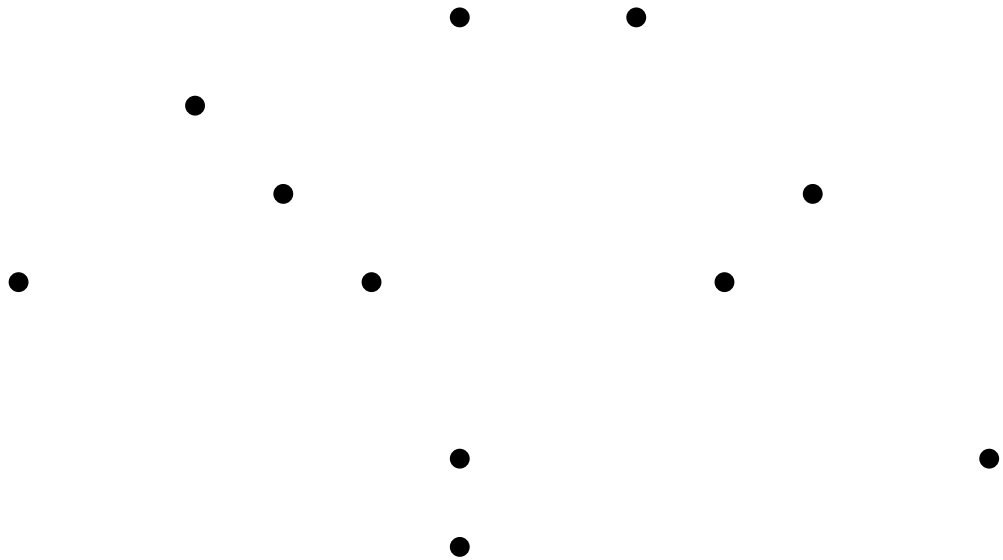
Αλγόριθμος του Andrew

Ιδέα

- Εξετάζουμε τα σημεία με αύξουσα λεξικογραφική σειρά.
- Τα δύο πρώτα σημεία αρχικοποιούν το άνω περίβλημα (αΠ) και το κάτω περίβλημα (κΠ).
- Σε κάθε βήμα τηρούμε το αΠ (κΠ) σαν μια `cw` (`ccw`) λίστα σημείων.
- Κάθε επόμενο σημείο p ενημερώνει το αΠ και το κΠ εξετάζοντας τη στροφή των δύο τελευταίων κάθε λίστας προς το p:
  - Όσο η στροφή είναι `ccw` κάνουμε `pop` από τη λίστα του αΠ.
  - Όσο η στροφή είναι `cw` κάνουμε `pop` από τη λίστα του κΠ.
  - Προσθέτουμε το p στο τέλος των λιστών αΠ και κΠ.
- Επιστρέφουμε τις λίστες αΠ και κΠ ή τις συγχωνεύουμε κατάλληλα για να πάρουμε το κυρτό περίβλημα.

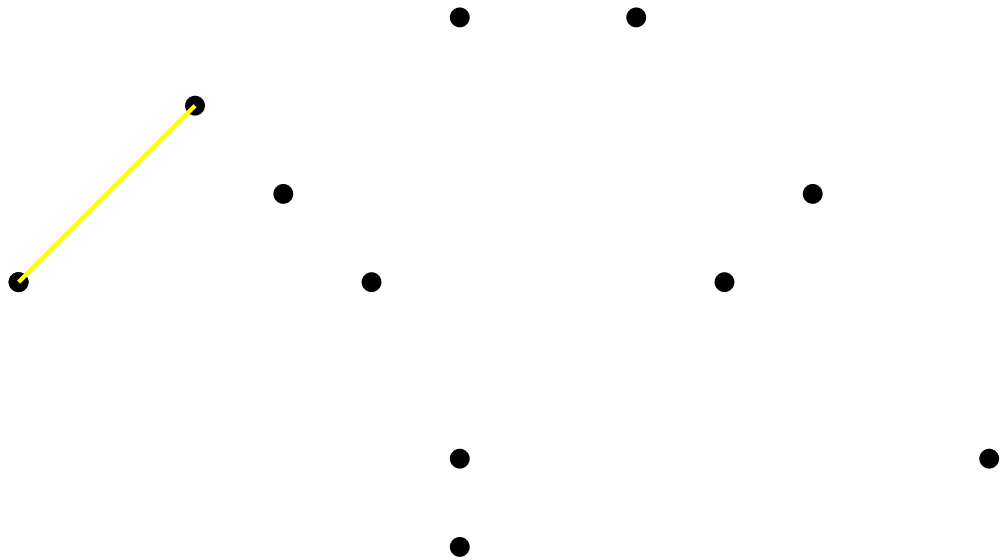
Αλγόριθμος του Andrew

Παράδειγμα



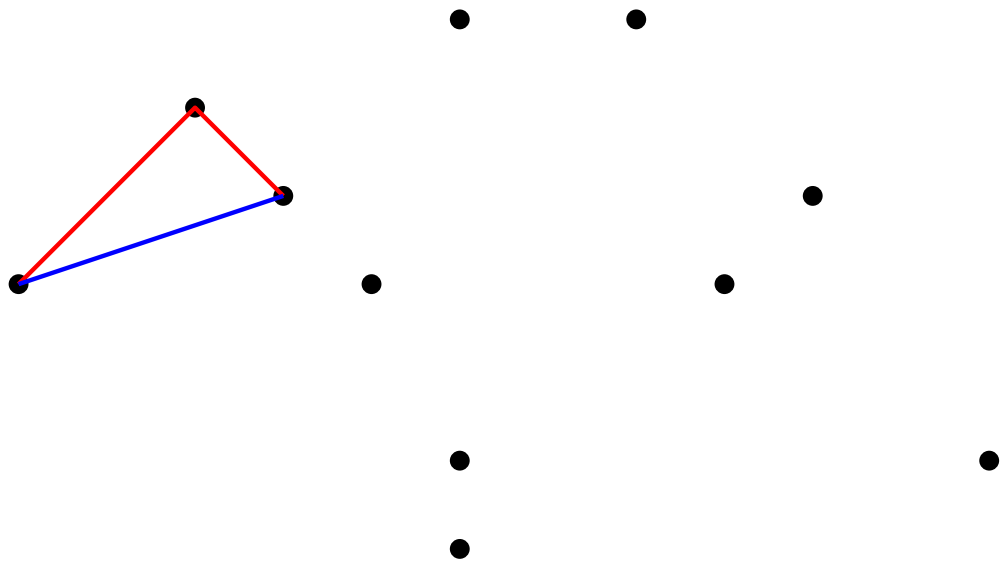
Αλγόριθμος του Andrew

Παράδειγμα



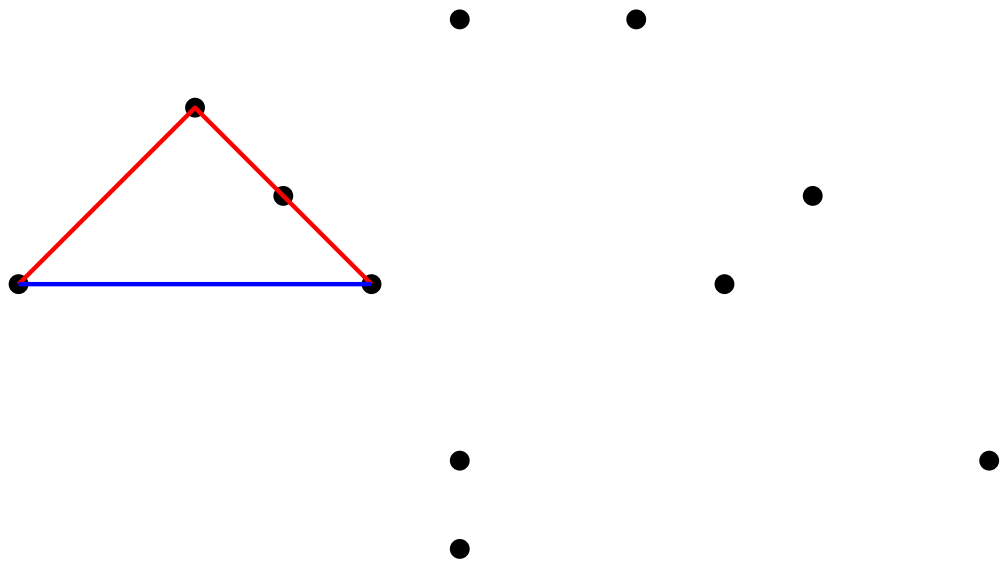
Αλγόριθμος του Andrew

Παράδειγμα



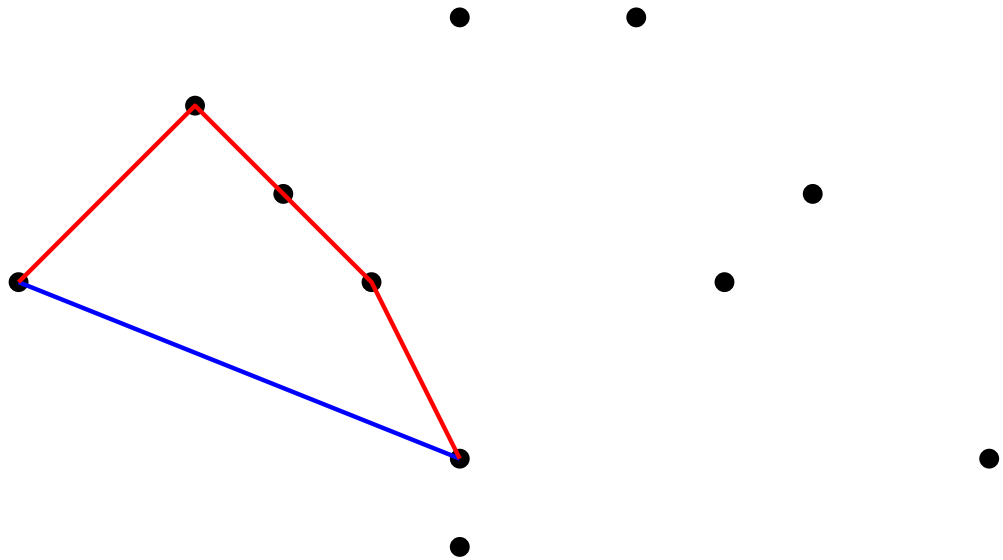
Αλγόριθμος του Andrew

Παράδειγμα



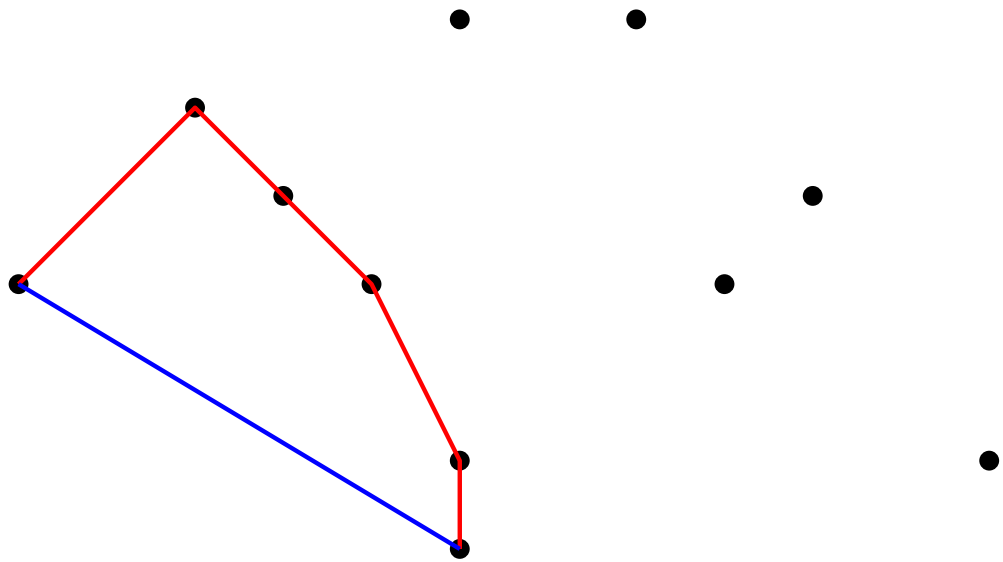
Αλγόριθμος του Andrew

Παράδειγμα



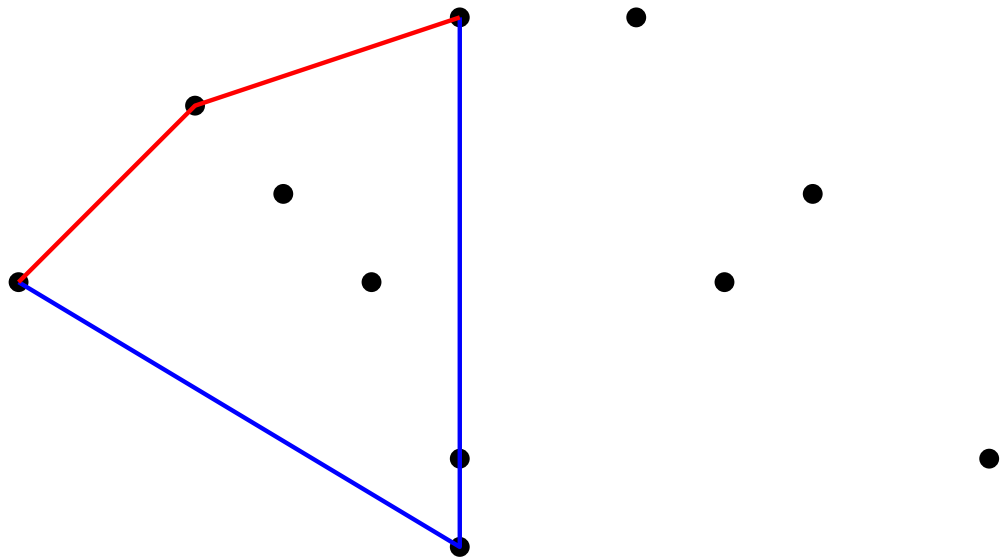
Αλγόριθμος του Andrew

Παράδειγμα



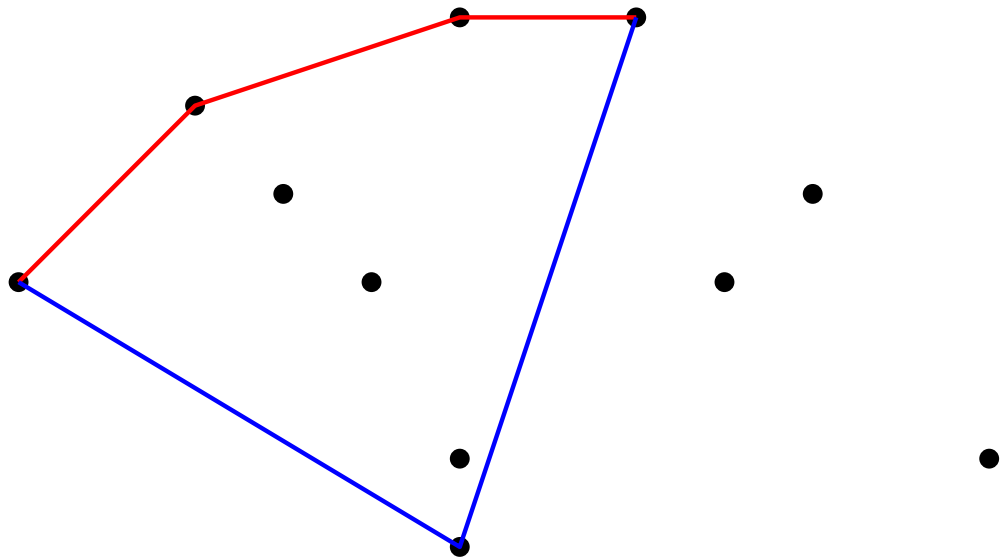
Αλγόριθμος του Andrew

Παράδειγμα



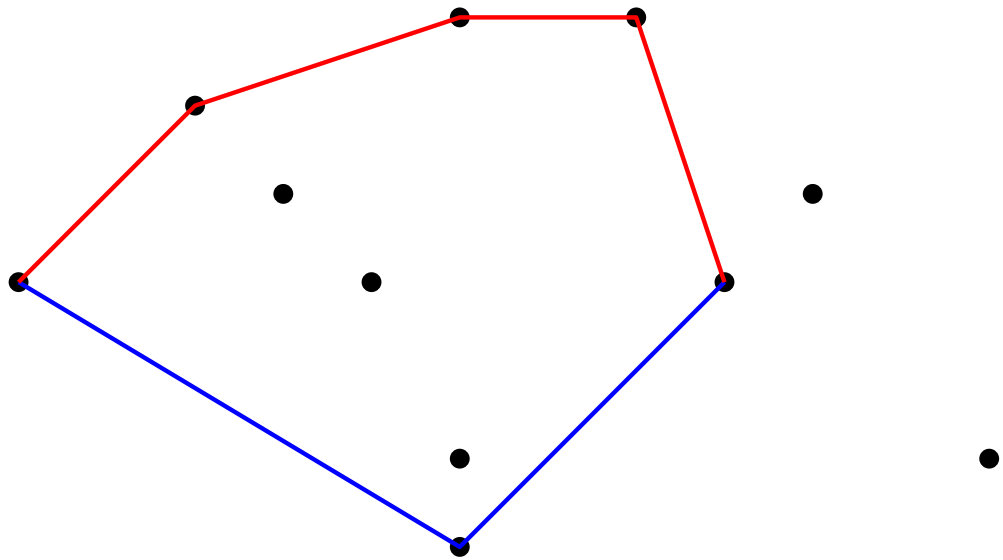
Αλγόριθμος του Andrew

Παράδειγμα



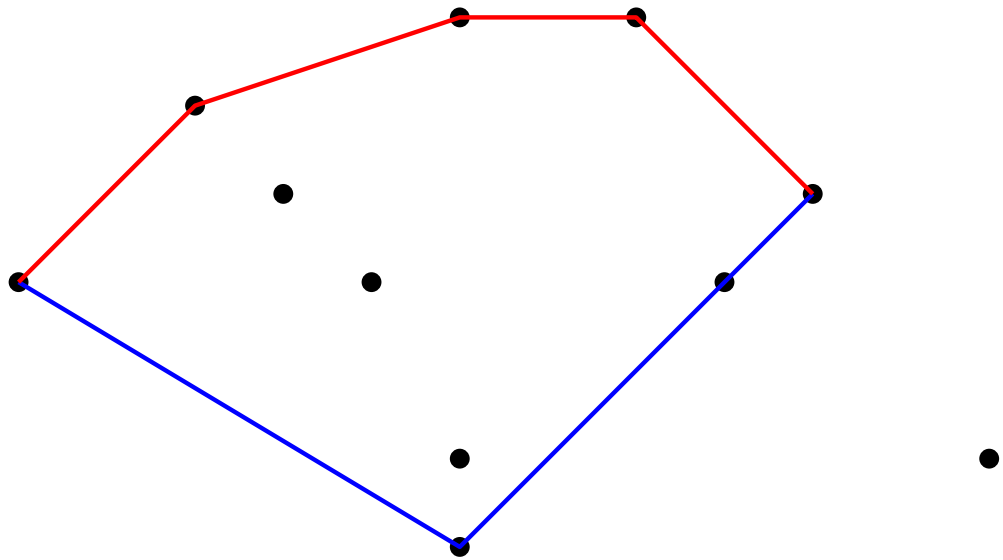
Αλγόριθμος του Andrew

Παράδειγμα



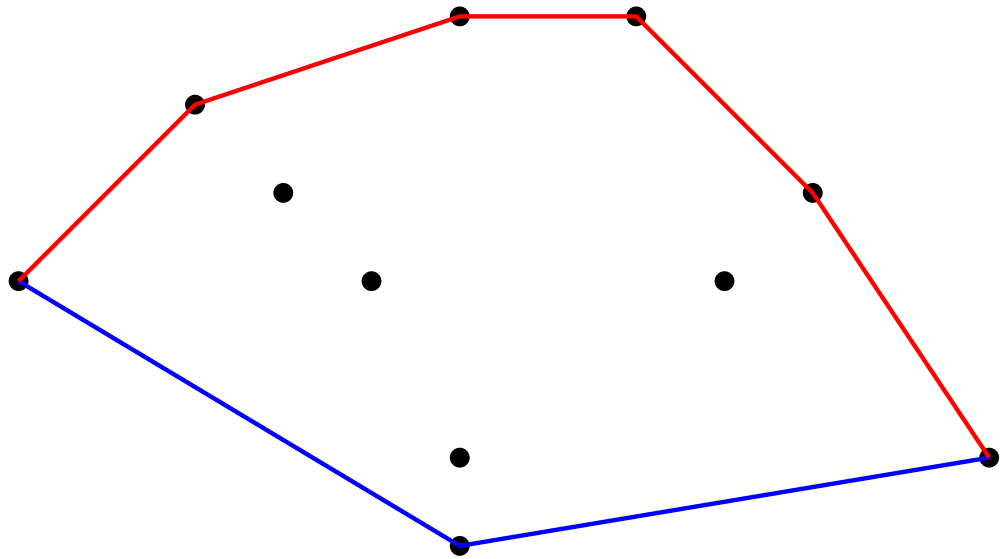
Αλγόριθμος του Andrew

Παράδειγμα



Αλγόριθμος του Andrew

Παράδειγμα



Αλγόριθμος του Andrew

Υλοποίηση στην Python

```
def andrew(points, return_hull=True):
  upper = []
  lower = []
  for p in sorted(points):
    while len(upper)>1 and ccwon(upper[-2], upper[-1], p):
      upper.pop()
    while len(lower)>1 and cwon(lower[-2], lower[-1], p):
      lower.pop()
    upper.append(point)
    lower.append(point)
  if return_hull:
    return lower[:-1]+ [x for x in reversed(upper[1:])]
  else:
    return upper, lower
```

- Επιστρέφει το κυρτό περίβλημα σαν μια `ccw` λίστα σημείων.
- Αν η παράμετρος `return_hull` είναι `False` επιστρέφει δύο λίστες για το αΠ και το κΠ σε `cw` και `ccw` σειρά αντίτοιχα.

Αλγόριθμος του Andrew

Ανάλυση πολυπλοκότητας

- Το βήμα της ταξινόμησης κοστίζει $O(n \log n)$.
- Κάθε σημείο θα εισαχθεί το πολύ μια φορά στις λίστες αΠ και κΠ.
- Κάθε σημείο θα εξαχθεί το πολύ μια φορά από τις λίστες αΠ και κΠ.
- Κόστος $O(1)$ για το χειρισμό κάθε επόμενου συμείου. Συνολικά $O(n)$ για το χειρισμό όλων των σημείων.

Διάμετρος συνόλου σημείων

Από ένα σύνολο σημείων S να βρεθούν τα δύο σημεία που απέχουν την μεγαλύτερη απόσταση.

```
def andipodal(points):
  U, L = andrew(points, return_hull=False)
  i, j = 0, len(L)-1
  while i<len(U)-1 or j>0:
    yield U[i], L[j]
    if i == len(U)-1: j -= 1
    elif j == 0: i += 1
    elif (U[i+1].y-U[i].y) * (L[j].x-L[j-1].x) > \
        (L[j].y-L[j-1].y) * (U[i+1].x-U[i].x):
      i += 1
    else: j -= 1

def diameter(points):
  dlist = [((p.x-q.x)**2+(p.y-q.y)**2,(p,q)) \
    for p,q in antipodal(points)]
  diam, pair = max(dlist)
  return pair
```

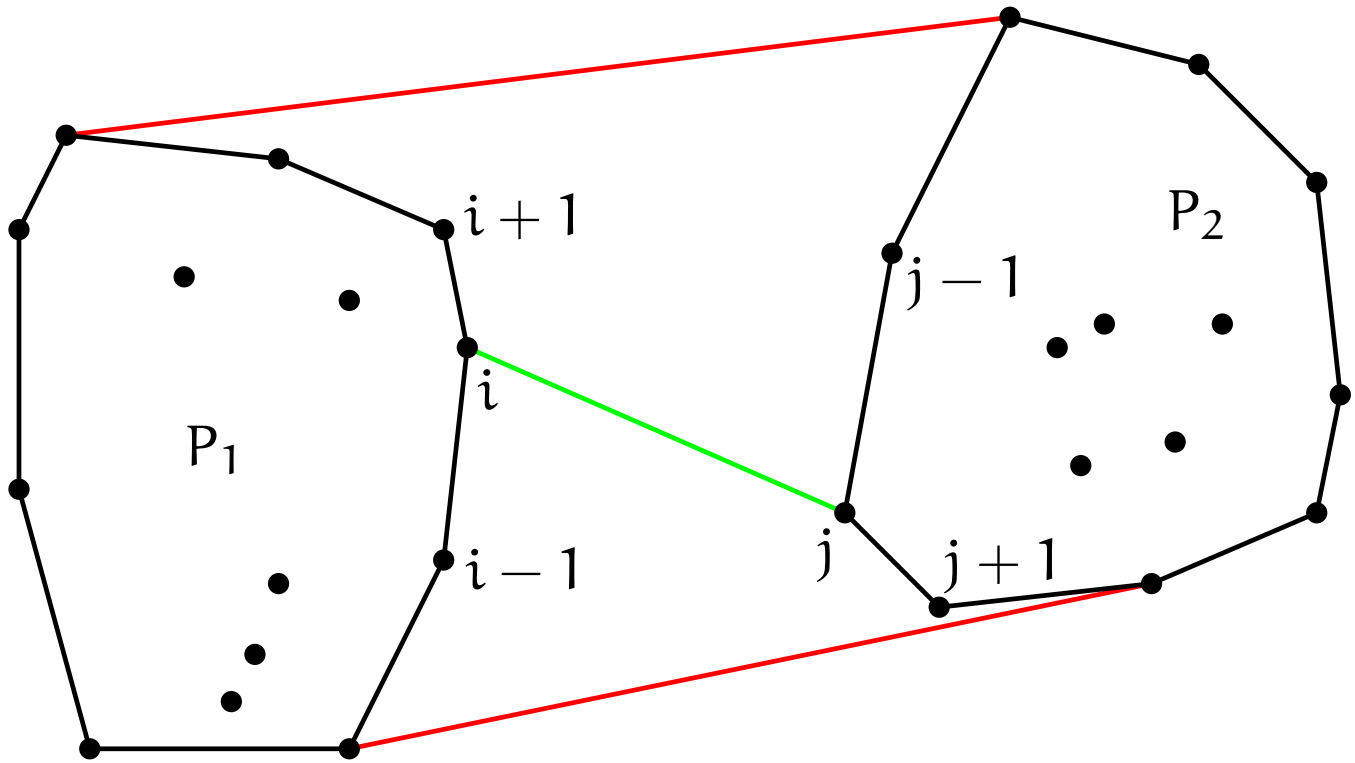
Περιεχόμενα

Αλγόριθμος Διαίρει και Βασίλευε

- Υπολογίζουμε το σημείο με την κεντρική τετμημένη και χωρίζουμε το αρχικό σημειοσύνολο στα δύο αντίστοιχα υποσύνολα.
- Εφαρμόζουμε τον αλγόριθμο για τα ΚΠ των δύο υποσυνόλων.
- Συνθέτουμε το συνολικό ΚΠ από τα δύο κυρτά περιβλήματα.
- Σύνθεση: εντοπισμός ακμών (*γέφυρες*) του συνολικού ΚΠ που $\notin$ σε κανένα από τα δύο ΚΠ των υποπροβλημάτων.
- Οι γέφυρες ενώνουν μια κορυφή του αριστερού ΚΠ με μια κορυφή του δεξιού ΚΠ.
- Η *πάνω γέφυρα*, από τα αριστερά προς τα δεξιά, αφήνει στα δεξιά τις υπόλοιπες κορυφές των ΚΠ των υποπροβλημάτων.
- Η *κάτω γέφυρα*, με την ίδια φορά, αφήνει στα αριστερά τις υπόλοιπες κορυφές των ΚΠ των υποπροβλημάτων.

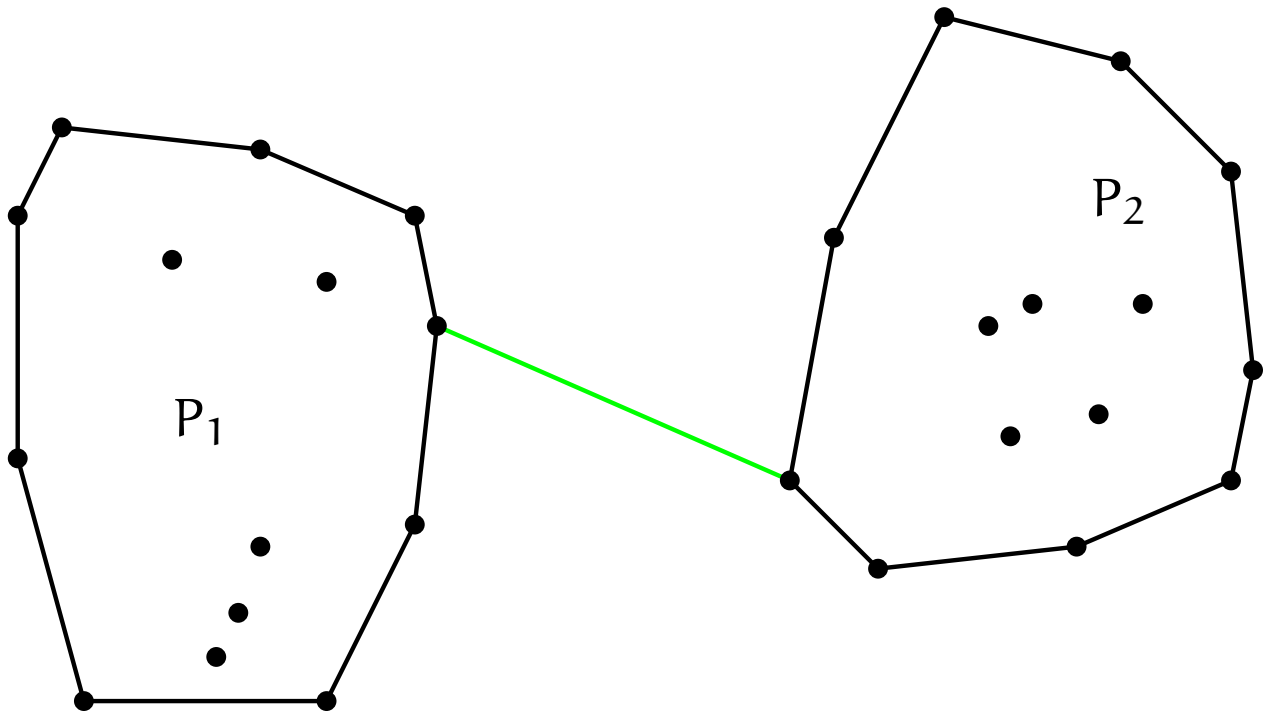
Αλγόριθμος Διαίρει και Βασίλευε

Βήμα σύνθεσης των υποπροβλημάτων



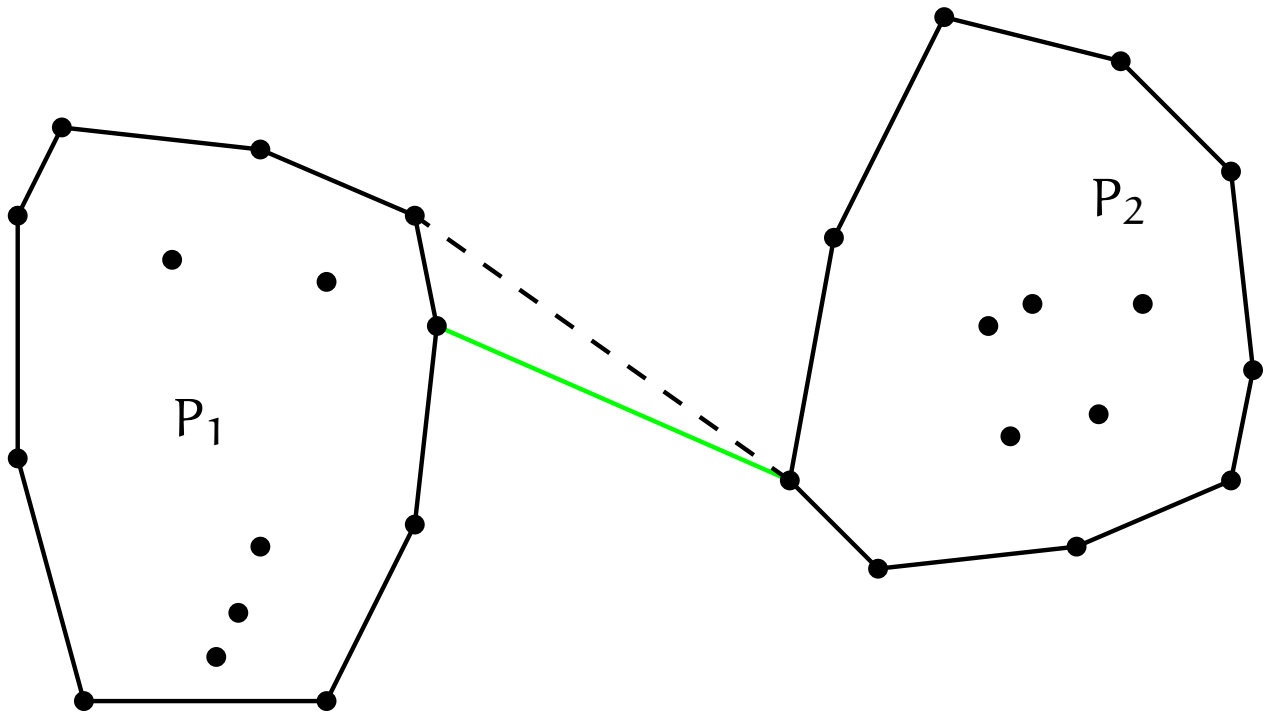
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



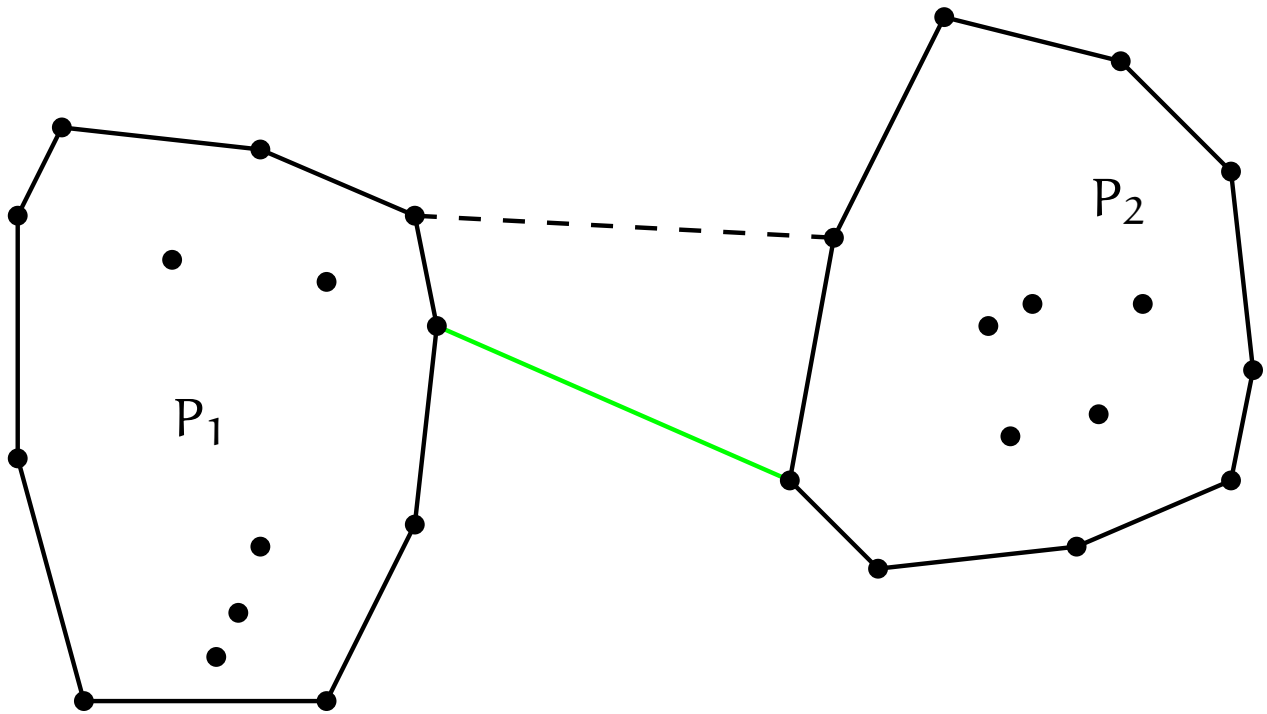
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



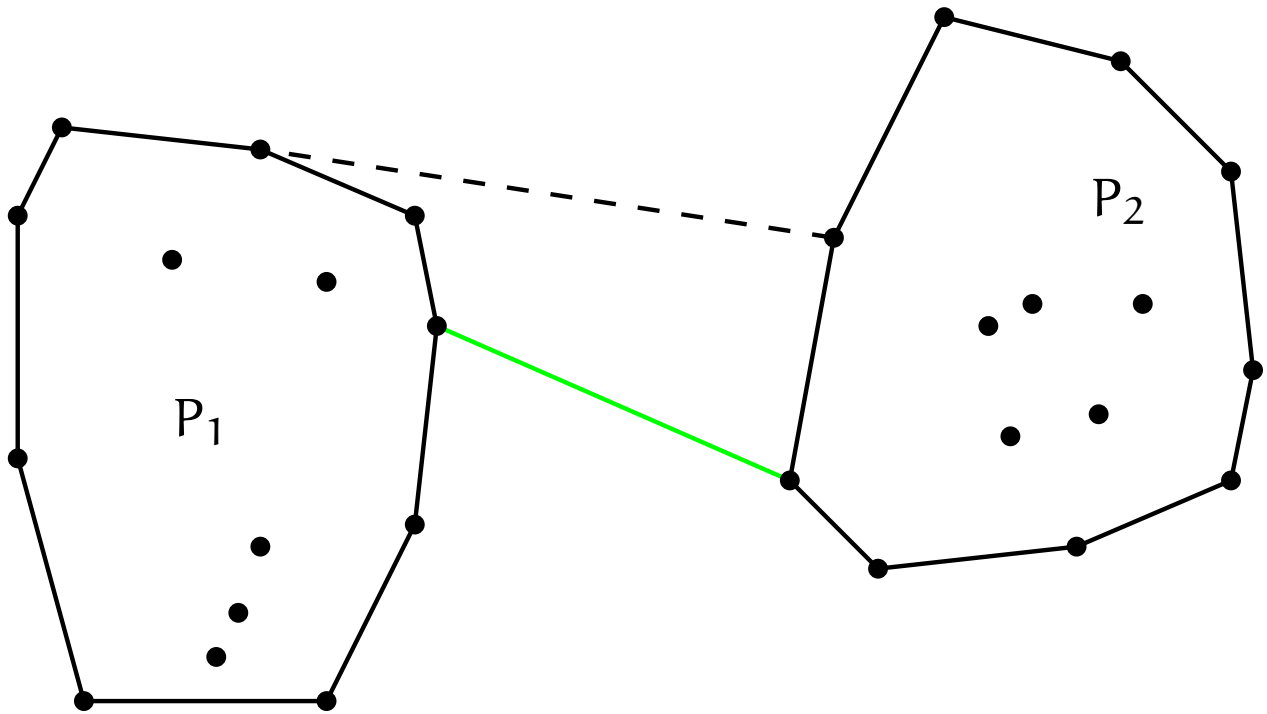
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



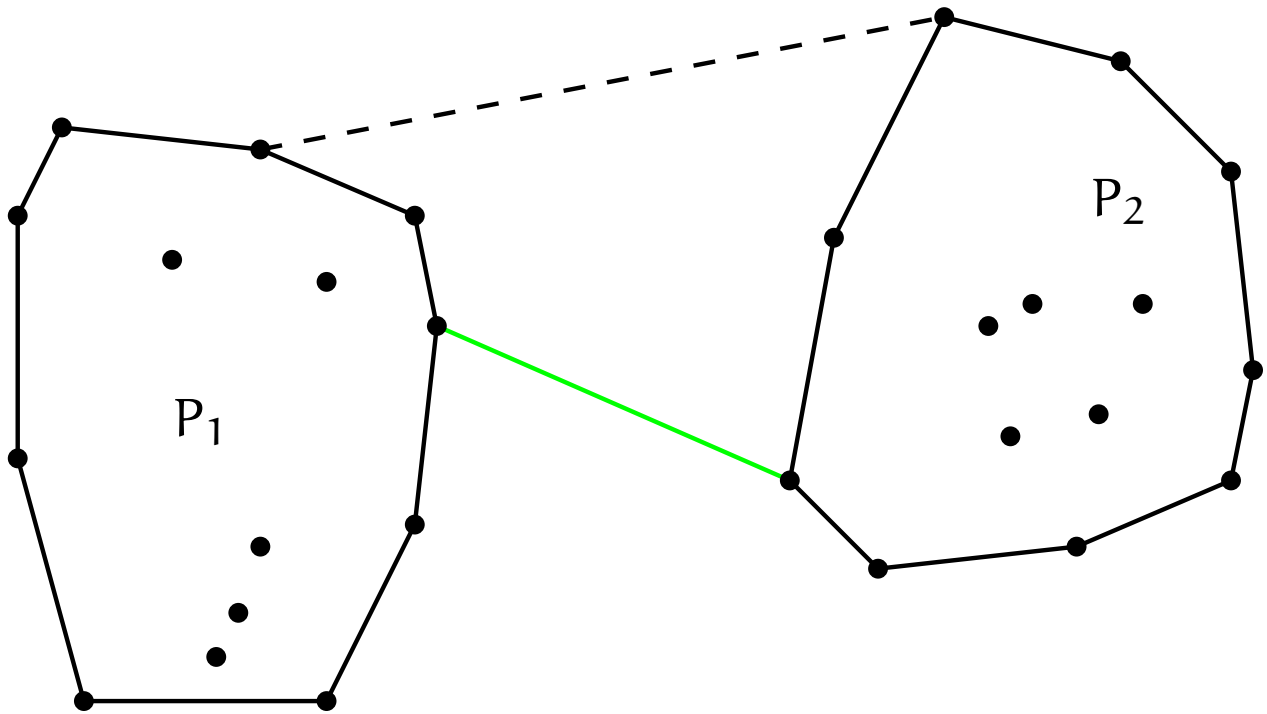
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



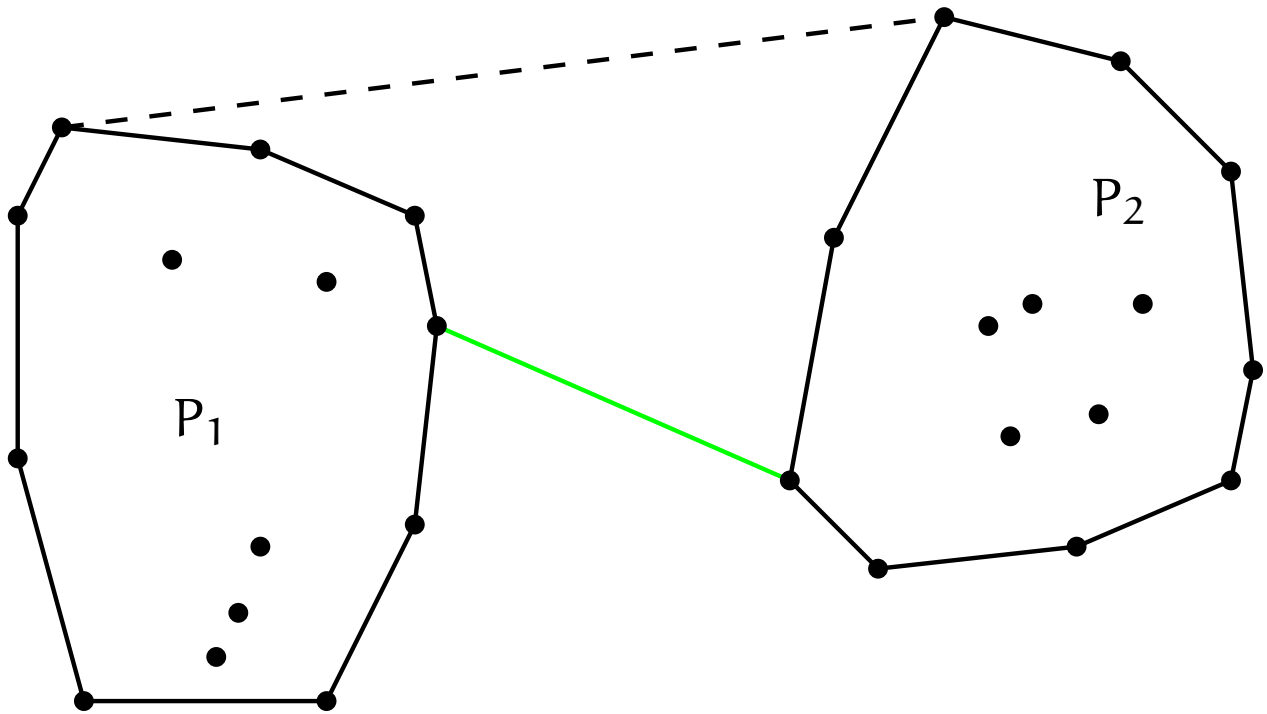
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



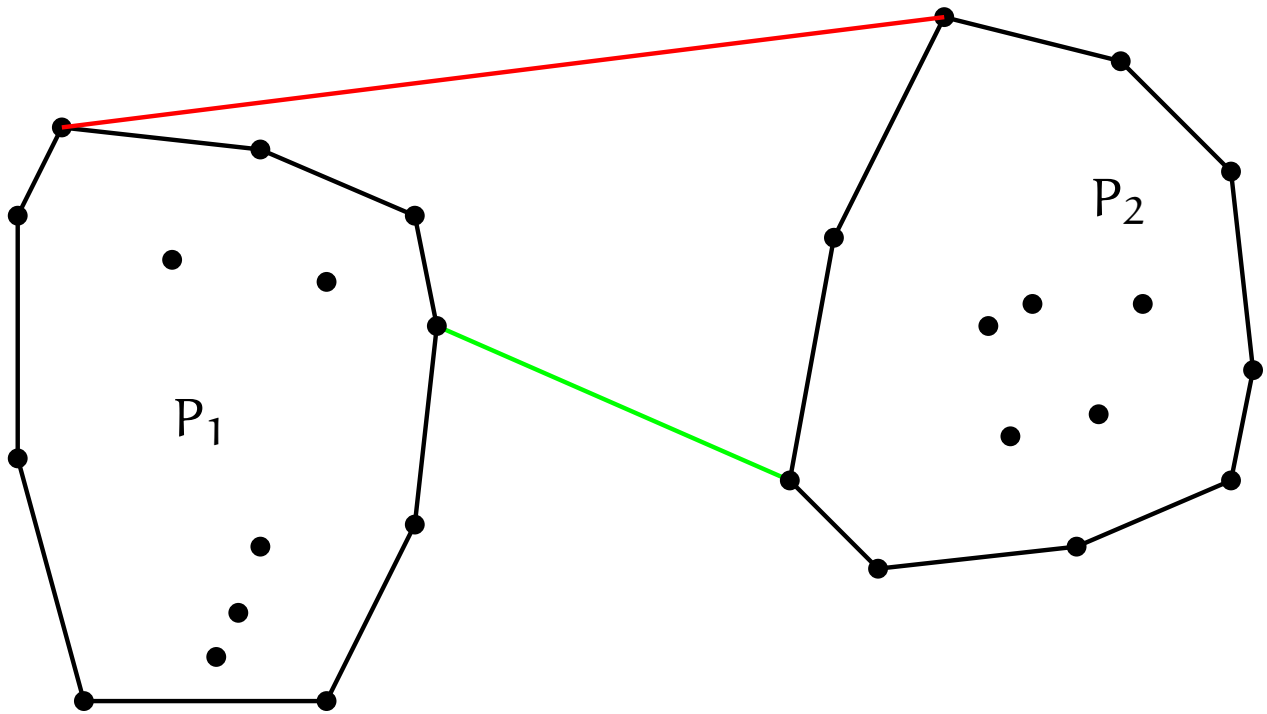
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



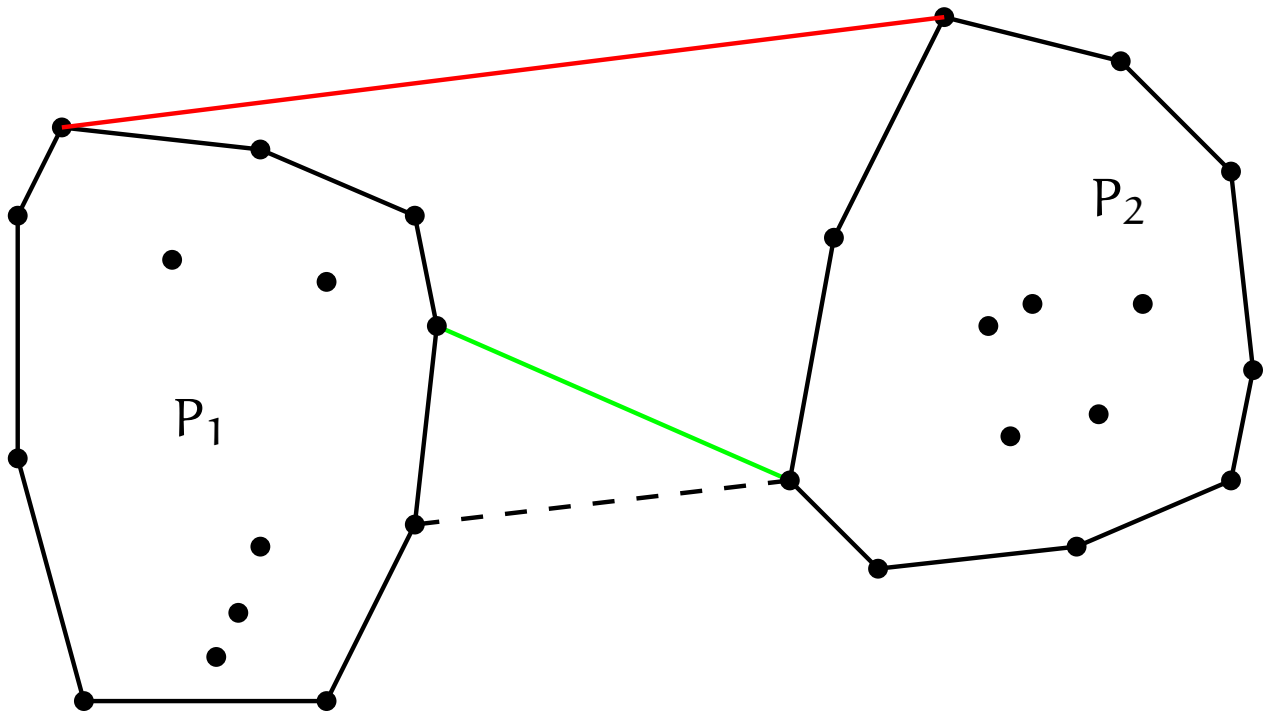
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



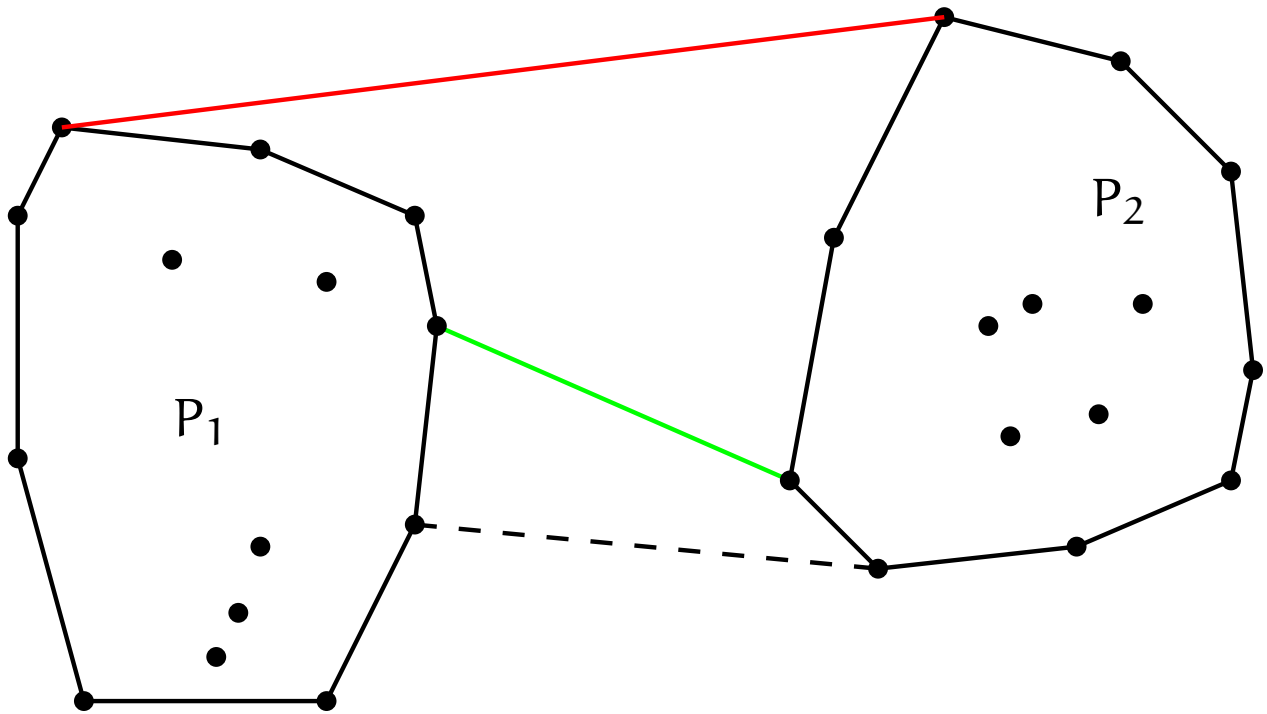
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



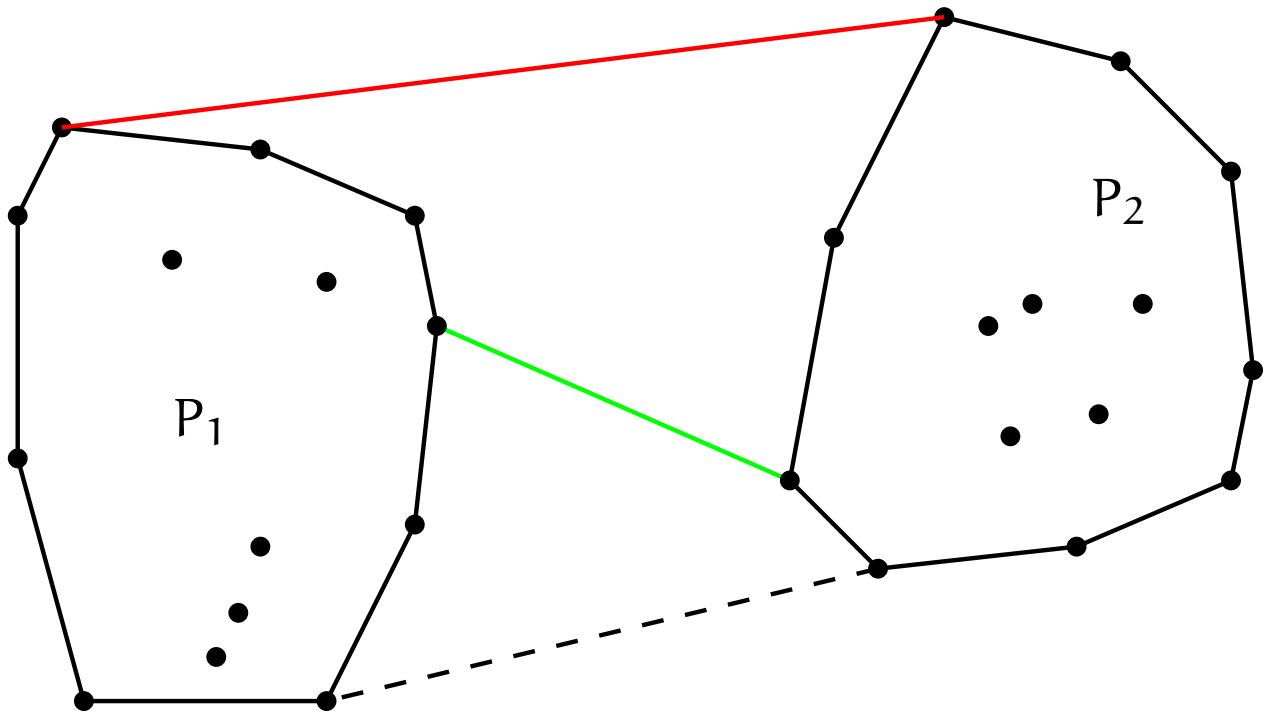
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



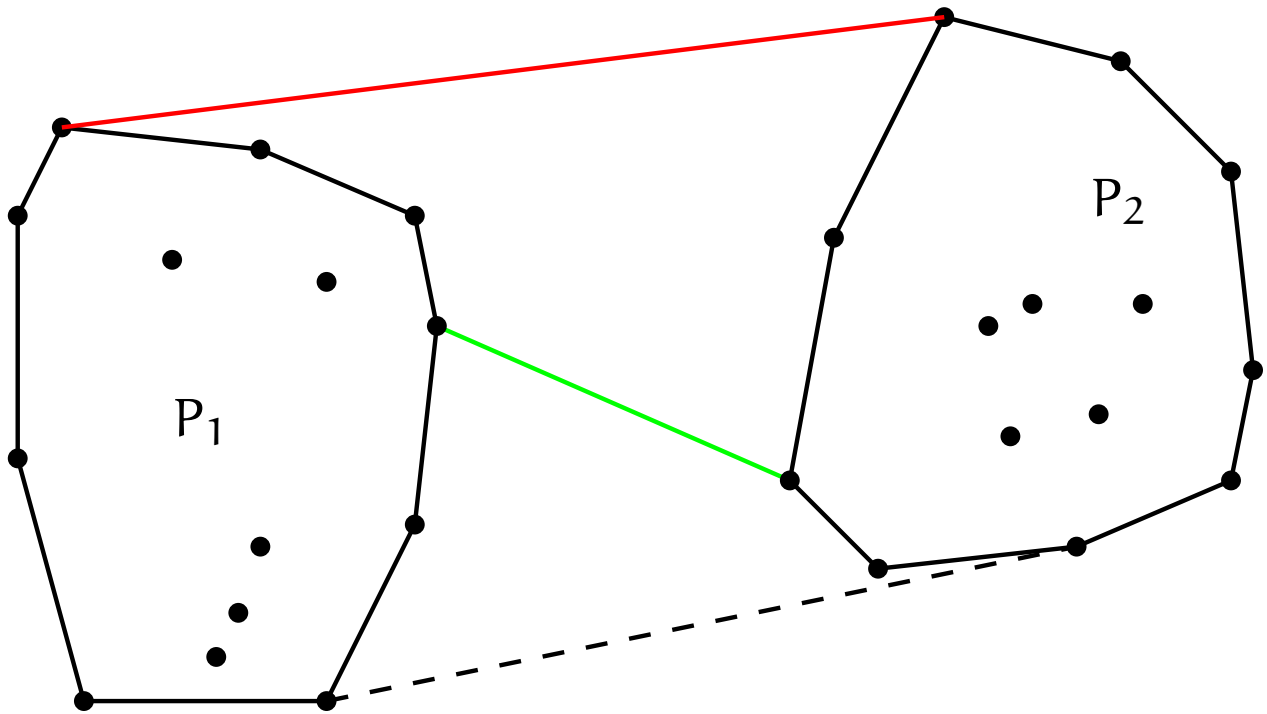
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



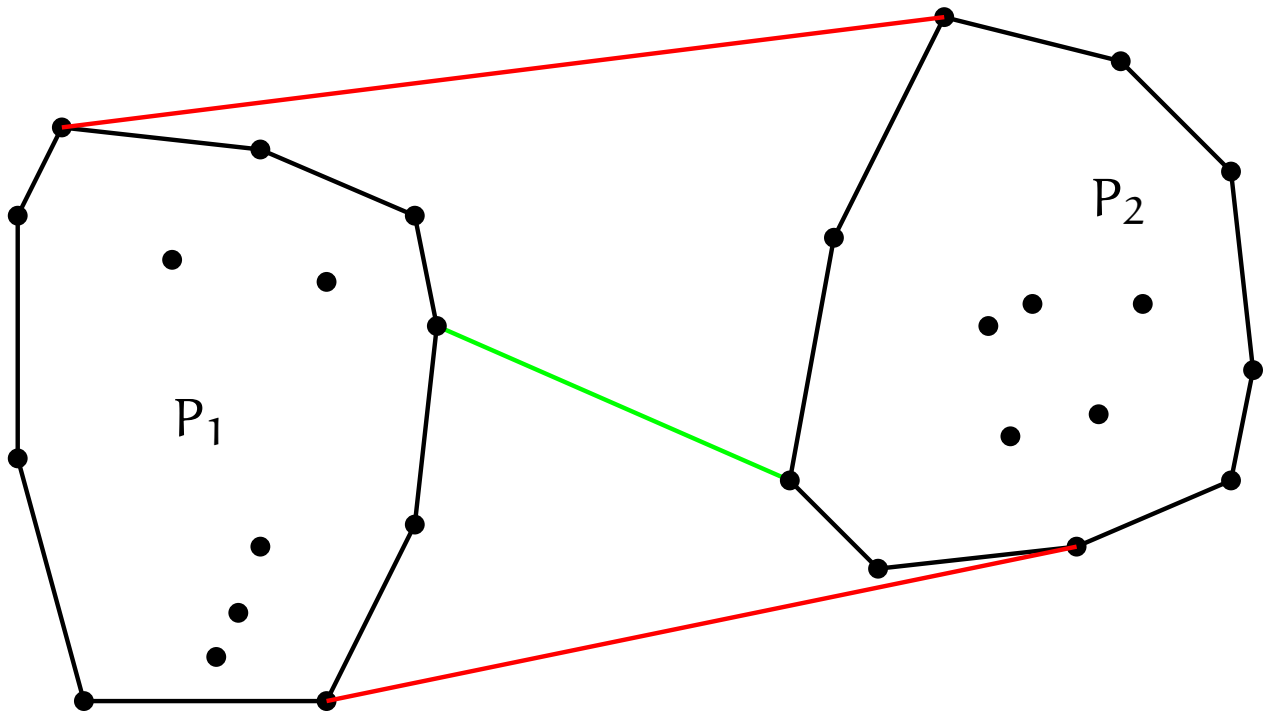
Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



Αλγόριθμος Διαίρει και Βασίλευε

Παράδειγμα



Αλγόριθμος Διαίρει και Βασίλευε

Υλοποίηση της σύνθεσης των υποπροβλημάτων στην Python

```
def find_upper_bridge(polygon1, polygon2):
  i = polygon1.index(max(polygon1.vertices))
  j = polygon2.index(min(polygon2.vertices))
  i_changed = j_changed = False
  while not i_changed and not j_changed:
    if not ccw(polygon1[i], polygon1[i+1], polygon2[j]):
      i = i+1
      i_changed = True
    else:
      i_changed = False
    if not cw(polygon2[j], polygon2[j-1], polygon1[i]):
      j = j-1
      j_changed = True
    else:
      j_changed = False
  return Segment2(polygon1[i], polygon2[j])
```

Αλγόριθμος Διαίρει και Βασίλευε

Ανάλυση πολυπλοκότητας

- Χρονική πολυπλοκότητα $T(n) = 2T(\frac{n}{2}) + O(n)$ αν το βήμα της σύνθεσης των υποπροβλημάτων κοστίζει $O(n)$.
- Αν $n = 2^k$ τότε $k = \log n$. Θέτουμε $O(n) = cn$ και αναπτύσουμε το $T(n) = T(2^k)$ σύμφωνα με την αναδρομή $T(n) = 2T(\frac{n}{2}) + cn$:

$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + c2^k \\ &= 4T(2^{k-2}) + 2c2^k \\ &= 8T(2^{k-3}) + 3c2^k \\ &= \cdots \\ &= 2^m(2^{k-m}) + mc2^k \end{aligned}$$

- Αν επαναλάβουμε k φορές τότε

$$T(2^k) = 2^k + ck2^k = n + cn\log n = O(n\log n)$$