P-uppgift Konvexa höljet
DD1345, grupdat13

Örjan Ekeberg/Ann Bengtson

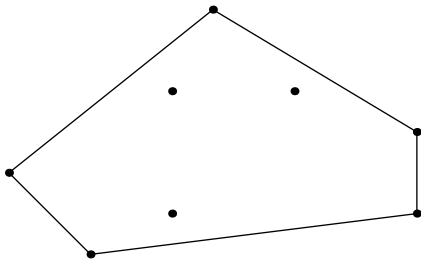
10 februari 2014

Målsättning

I den här uppgiften får du övning i att implementera en algoritm utgående från en beskrivning av algoritmen. Du ska få insikt i hur man kan manipulera geometriska objekt som punkter, linjer och polygoner. Resultatet av algoritmen ska visas grafiskt med MATLAB eller något av Pythons grafikpaket. **Betygsgradering: C-A**

Uppgiften

Skriv ett program som räknar ut det *konvexa höljet* för en punktmängd. Punkterna läser du in från en textfil där varje rad innehåller två tal som ska tolkas som x- och y-koordinater. Det finns tre testfiler. De två små filerna är främst för teständamål men resultatet av den större filen `sweden.txt` ska redvoisas.
Det konvexa höljet är den minsta konvexa polygonen där ingen punkt ligger utanför. Figur 1 visar ett exempel. Det konvexa höljet till en mängd punkter i planet kan beräknas med en algoritm som kallas GRAHAM-SCAN. Din uppgift är att implementera denna algoritm och använda den för att hitta och rita ut det konvexa höljet till punkterna i filen.



Figur 1: Konvexa höljet till en punktmängd i planet. Det konvexa höljet är en polygon som byggs upp av en delmängd av punkterna.

Algoritmen

Graham-Scan går till på följande sätt:

1. Låt p_0 vara den punkt som har minst x-koordinat.

   Om fler har samma x-koordinat, välj den (av dessa) som har störst y-koordinat.

2. Låt $[p_1, p_2, \ldots, p_n]$ vara resten av punkterna, ordnade motsols runt punkten p_0.

3. Lägg in punkterna p_0, p_1 och p_2 i en stack s.

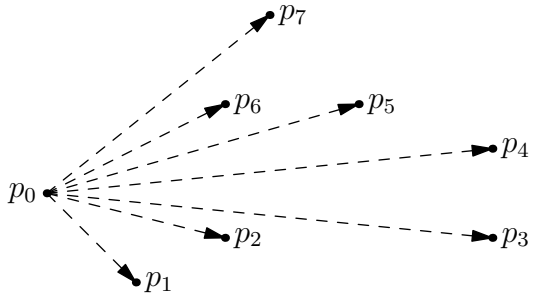
4. För varje punkt p i $\{p_3, p_4, \ldots, p_n\}$

   - Så länge punkten överst på s ligger till vänster om linjen från punkten näst överst på s till p så ta bort stackens översta element.
   - Lägg in p på stacken.

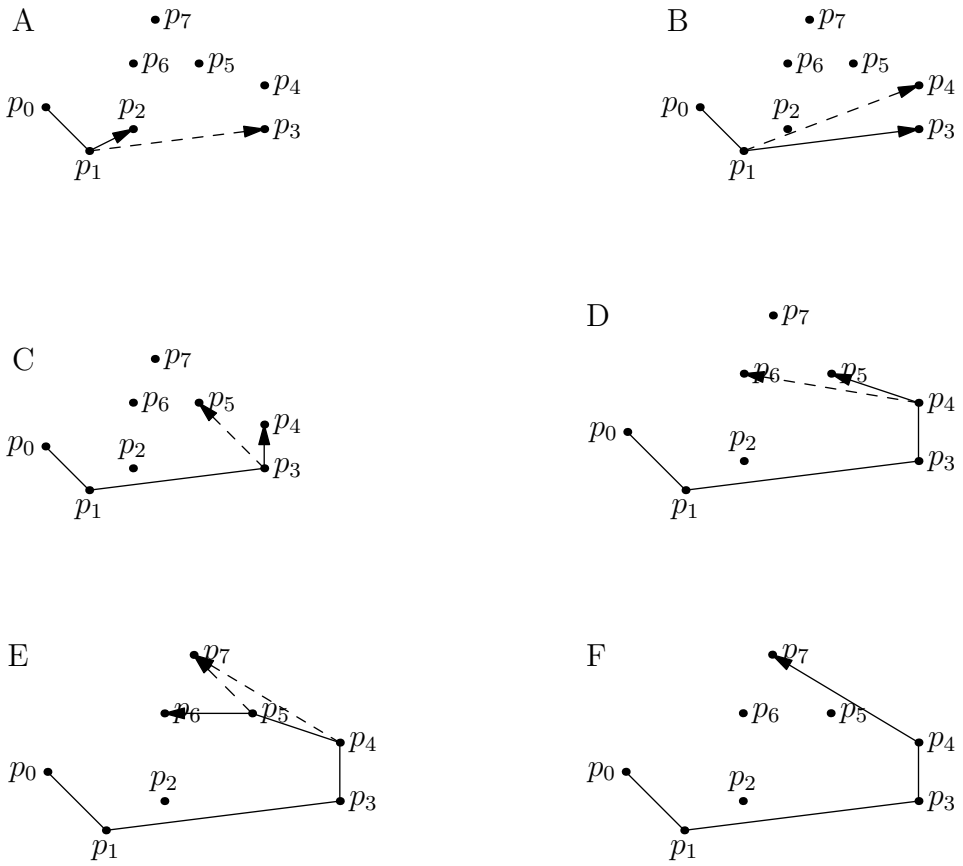
5. Returnera punkterna i s

Steg 1 garanterar att punkten p_0 är en punkt som kommer att tillhöra lösningen. Steg 2 kräver att punkterna sorteras efter stigande vinkel för linjen från p_0 till den aktuella punkten (se figur 2).

Huvuddelen av algoritmen är loopen i steg 4. Hur den fungerar illustreras i figur 3.



Figur 2: Punkterna $p_1, p_2, \ldots, p_n$ sorteras efter stigade vinkel på linjen som utgår från p_0.



Figur 3: Bilderna illustrerar hur GRAHAM-SCAN algoritmen succesivt bygger polygonen. Först (**A**) upptäcker man att punkten p_2 ligger till vänster om linjen från p_1 till p_3. p_2 plockas därför bort från stacken innan p_3 läggs in. I nästa steg (**B**) konstateras att p_3 ligger till höger om linjen $p_1 \rightarrow p_4$. p_3 lämnas därför kvar på stacken när p_4 läggs in. I steg **C** jämförs p_4 med linjen $p_3 \rightarrow p_5$ och eftersom punkten ligger till höger om linjen så lämnas den kvar på stacken. I **D** jämförs p_5 med linjen $p_4 \rightarrow p_6$ och även här ligger punkten till höger om linjen. I nästa steg (**E**) plockas först punkten p_6 bort eftersom den ligger till vänster om $p_5 \rightarrow p_7$, men här plockas även p_5 bort eftersom den ligger till vänster om $p_4 \rightarrow p_7$. Den slutliga lösningen utgörs av punkterna som finns kvar på stacken när alla punkter behandlats (**F**).

Hjälpfunktioner

Graham-Scan använder sig av en återkommande test: befinner sig en punkt p till vänster om en linje $p_0 \rightarrow p_1$? Detta undersöker man enklast (och säkrast) genom att beräkna kryssprodukten av vektorerna $p_0 \rightarrow p$ och $p_0 \rightarrow p_1$ och avgöra om denna är positiv eller ej. Kryssprodukten mellan två vektorer a och b i planet räknas enkelt ut som $a_x b_y - a_y b_x$. Skriv en hjälpfunktion som tar tre punkter och returnerar True om den första punkten ligger till vänster om linjen från den andra till den tredje.

För att kunna sortera punkterna i vinkelordning behöver man räkna ut vinkeln för en vektor. Här är det lämpligt att utnyttja funktionen `math.atan2(y, x)` som ger vinkeln (i radianer) från origo till punkten (x, y). För att sorteringen ska fungera är det viktigt att vinkelberäkningarna inte plötsligt gör ett hopp på 2π. Genom att utgå från punkten med minst x-koordinat är vi garanterade att alla vinklar ligger mellan $-\frac{\pi}{2}$ och $\frac{\pi}{2}$. math.atan2 gör sitt "hopp" vid π så detta bör vara säkert.

Grunduppgift, betyg C/B

- Implementera Graham-Scan
- Presentera resultatet grafiskt. Ursprungliga punkter och det resulterande konvexa höljet ska visas tydligt.
- Markera största avståndet i punktmängden.

Implementera de funktioner som behövs för att räkna ut vilken polygon som utgör det konvexa höljet till punkterna i en textfil. För sorteringen kan du med fördel utnyttja den inbyggda funktionen `sorted`. Lägg märke till att `sorted` kan ta en extraparameter (en s.k. nyckelordsparameter) i form av en funktion som plockar fram den nyckel man vill sortera efter; i vårt fall vinkeln. Sorteringen kan därför göras på detta sätt:

```
def myKey(point):
  return angleBetween(origin, point)

origin = p[0]
v = sorted(p[1:], key=myKey)
```

Punktmängdens största utsträckning $\mathcal{O}(n^2)$

Finn de två punkter som är längst ifrån varandra i den ursprungliga punktmängden. Dessa två punkter måste ingå i konvexa höljet, eller hur?
En *brute force* algoritm som beräknar alla parvisa avstånd kommer att kräva $\mathcal{O}(n^2)$ tid.

Punktmängdens största utsträckning $\mathcal{O}(n)$

För att komma ned till $\mathcal{O}(n)$ måste man vara smartare. n är här antalet punkter i konvexa höljet.

Uppgift för betyg A

Förutsätter betyg B på grunduppgiften.
Ett sätt att hitta största avståndet i punktmängden $\mathcal{O}(n)$ är *Rotating Calipers algorithm*. Leta rätt på information om och implementera den istället för *brute force*-varianten.

Strukturering av uppgiften

Det är tillåtet att strukturera lösningen på ett annat sätt än som föreslås ovan. För högsta betyg krävs förstås en tydlig lösning utan koduppprepning.

Grafisk redovisning

Resultatet ska presenteras grafiskt och *tydligt*. Det gäller alla betygsnivåer. Ursprunglig polygon (eller punktmängd) ska visas och konvexa höljet ritas ut. Det är nog snyggast att inte dra streck mellan punkterna i startmängden. Punkterna i konvexa höljet ska sammanbindas med streck. Dessutom ska polygonens största utsträckning markeras. Använd olika färger för de tre.

Matlab

Låt Python-programmet skriva ut filer som kan läsas direkt av Matlab och skriv ett Matlab-program som visar resultatet snyggt. Det går förstås bra att använda Octave istället för Matlab.

Turtle-graphics

Se nästa avsnitt!

Annat grafik-paket

Leta reda på information själv! Länk till en del info finns på kurshemsidan under *Litteratur & länkar*.

Sköldpaddsgrafik

Under 1960-talet utvecklades ett experimentellt programmeringsspråk *Logo* speciellt för att lära barn programmering. Logo introducerade en speciell sorts grafik kallad *Turtle Graphics*, där namnet syftar på den typ av figurer som skapas när sköldpaddor går omkring på en sandstrand. Python har en modul `turtle` som implementerar denna typ av grafik. Prova gärna följande:

```
import turtle

turtle.down()
turtle.goto(100, 0)
turtle.up()
turtle.goto(50, 0)
turtle.down()
turtle.circle(10)
turtle.color('blue')
turtle.right(90)
turtle.forward(20)
```

Här är ett urval av de funktioner som finns:

`down`	Sänk pennan
`up`	Lyft pennan
`goto`	Gå till angivna koordinater
`right`	Sväng höger
`forward`	Gå framåt
`circle`	Gå runt
`color`	Byt färg på pennan
`speed`	Trimma sköldpaddan
`tracer`	Animerad uppritning på/av

Mer information för du genom `help(turtle)` i Python, bl.a. hur du fyller ytor och skriver text.

Använd sköldpaddsgrafik för att rita ut alla punkterna och det konvexa höljet. Använd t.ex. små cirklar för punkterna och linjer för det konvexa höljet. Rita gärna lite större cirklar för de punkter som ingår i höljet.

Sköldpaddsgrafiken är från början inställd att rita långsamt för att man ska hinna se vad som händer. Om du tycker att tempot är lite väl lågt kan du öka sköldpaddans fart genom `turtle.speed('fastest')`. Alternativt kan du helt stänga av animeringen under uppritningen genom `turtle.tracer(False)`.