DD2458, Problemlösning och programmering under press

Föreläsning 11: Beräkningsgeometri

Datum: 2009-11-24
Skribenter: David Björklund, Christoffer Lundell Johansson och Mårten Selin
Föreläsare: Fredrik Niemelä

1 Inledning

För geometriska problem, både enkla och mer avancerade, finns det vissa saker som gör att det försvåras eller blir rent ut av sagt fel. Det man tänker mest på är dock specialfall och precision/overflow.

2 Specialfall

Om man tänker efter, många gånger hittar man en funktion eller metod som löser ett problem väldigt bra, men det finns nästan alltid specialfall som gör att metoden inte beter sig som man kanske vill eller ger fel svar helt och hållet. Låt oss ta som exempel en metod att beräkna om linjer skär varandra, då kanske man har en koll som ser om det finns en chans att de skär varandra, den kollen kan helt missa specialfallet att de ligger i samma linje, att den andra linjen börjar och följer den första linjen en tid, hur hanteras det specialfallet?

- **Strunta i det:** Kan vi på något sätt gå runt problemet? Kanske det bara uppstår för värden under 1? gör så man inte får använda dessa värden. Man kan också antaga eller bevisa att detta specialfall inte kommer eller kan uppstå.
- **Hantera det:** Vi har specialfall, kan vi hantera det? Blir det fel värde vid 0 och 1 men för alla andra tal över är det ok? Skriv in dessa för direkt uppslagning!
- **Ljug:** Låter lite konstigt men ändå. Ta indata och ändra det eller på annat sätt gör så att problemet inte kan uppstå eller liknande bakom kulisserna lösning på problemet.

2.1 Precisions och overflow

Andra problem man kan få är med precision och overflow. Om man gör stora eller långa beräkningar kan tals begränsade precision ge fel som avviker med flera storleksordningar, inte bara små avrundingsfel. Men vad kan man göra åt detta då? Det finns ett antal åtgärder:

- **Heltals aritmetik:** Att använda heltal när det är möjligt ger mycket bättre precision. Som att representera ett decimaltal med två long long. En representerar siffrorna före och en siffrorna efter decimal tecknet. Man kan också använda specialgjorda data strukturer som BigFloat eller javas BigDecimal. Dessa datastrukturer är specialgjorda för att representera godtyckligt stora tal där datorns totala minne sätter gränsen.

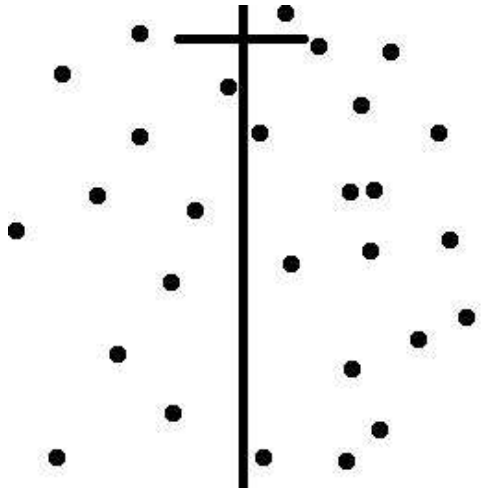
- **double/long double:** Vissa gånger behövs det kanske inte allt för hög precision och då funkar dessa bra.

3 Punkter

3.1 Kortaste avståndet mellan punkter

Att naivt räkna ut det kortaste avståndet mellan två punkter i en en punktmängd ger $n(n-1)/2$ uträkningar och faller in i $O(n^2)$ komplexitet. Men det finns bättre sätt att göra det på. Det handlar om att minska ner problemet för att sedan utnyttja vad vi vet från delproblemen med att hitta snabbare lösning.

Algoritm 1: Hitta kortaste avståndet i en punktmängd
Input: En punktmängd P
Output: Kortaste avståndet mellan två punkter
ShortestDist(P)
(1) **if** $|P| = 2$
(2) **return** Avståndet mellan de två punkterna i P.
(3) Hitta ett x värde där vi får hälften av alla punkter på var sin sida av linjen
(4) Räkna naivt ut kortaste distansen mellan dessa två delmängders punkter
(5) Skapa 2 arrayer, sorterad på x och y för punkterna för snabb upp slagning
(6) Spara minsta värdet av del höger och del vänster och spara det som eventuell lösning i d
(7) **foreach** $p \in$ vänstra halvan i stigande ordning
(8) **if** p ligger närmare skiljelinjen än d
(9) Se om någon punkt på andra sidan skiljelinjen (inom intressant intervall) ligger närmare än d
(10) Uppdatera d om nödvändigt
(11) **return** d



Figur 1: Ett snitt tas så att punktmängden delas upp i två lagom stora delmängder.

4 Linjer och linjesegment

4.1 Avgöra utifall en punkt ligger på en linje

Givet ett linjesegment med ändpunkter i (x_1, y_1) och (x_2, y_2) kan det vara intressant att avgöra utifall en punkt (x, y) ligger på linjen eller inte. Det gör man enklast genom att kolla att kryssprodukten mellan de två vektorerna som utgår från (x_1, y_1) och går till (x_2, y_2) respektive (x, y) blir noll:

$$|(x_2 - x_1, y_2 - y_1) \times (x - x_1, y - y_1)| =^? 0$$

$$\Rightarrow (y - y_1)(x_2 - x_1) =^? (x - x_1)(y_2 - y_1)$$

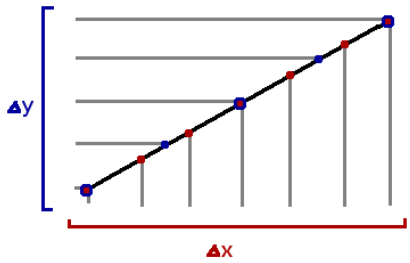
Därför då är vektorerna linjärt beroende, d.v.s. parallella, och de kan de endast vara om (x, y) ligger på linjen.

Hittills har vi bara kollat att punkten ligger på linjen, inte på linjesegmentet. Det är därför också nödvändigt, men enkelt, att kolla att punkten befinner sig inom rätt intervall. $x_1 \leq x \leq x_2 \; OR \; x_1 \geq x \geq x_2$ är ett tillräckligt test.

4.2 Antalet punkter med heltalskoordinater på linjen

Ett annat problem är att givet ett linjesegment med ändpunkter på heltalskoordinater, avgöra hur många andra punkter på linjesegmentet som har heltalskoordinater. Låt Δx och Δy vara differensen i ändpunkternas x- respektive y-koordinat. Punkter på linjesegmentet som har sin x-komponent som heltal av finns det i en: $0, 1/\Delta x, 2/\Delta x, ..., 1$ -te del av linjesegmentet. Samma sak för y: $0, 1/\Delta y, 2/\Delta y, ..., 1$ -te del av linjesegmentet. Punkter som uppfyller båda ligger längs en $n/\Delta x = m/\Delta y$ -te del av linjesegmentet där n och m är heltal. Antalet lösningar till $n/\Delta x = m/\Delta y, \quad (n, m) \in [0, \Delta x] \times [0, \Delta y]$, ger således antalet

punkter vi söker. Men det är ett lätt problem att lösa och antalet punkter blir: $GCD(\Delta x, \Delta y) + 1$ ("+1" kommer ifrån att en av ändpunkterna inte räknas med).



Figur 2: Punkter på linjesegmentet som uppfyller $x, y \in \mathbb{N}$ är $GCD(\Delta x, \Delta y) + 1$, inklusive start- och slutpunkt.

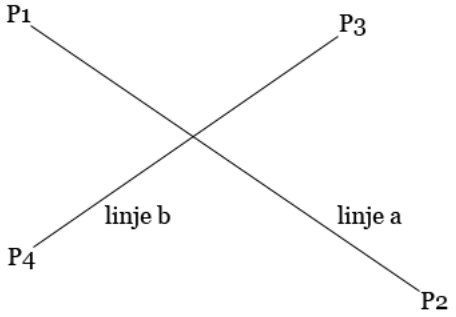
4.3 Korsande linjesegment

Linjesegment kan korsa varandra på tre olika sätt: de kan vara oberoende av varandra (dvs inte korsa varandra alls), de kan korsa varandra på en punkt, eller så kan de vara överlappande.

Det korsade området kan alltså antingen vara tomt, en punkt alternativt en linje.

Linjernas ekvationer är:
$L_a = P1 + u_a(P2 - P1)$
$L_b = P3 + u_b(P4 - P3)$

Vi letar initialt efter punkten där L_a och L_b korsas, dvs på punkten där u_a och u_b har sådana värden att $L_a = L_b$ gäller.



Figur 3: Korsande linjesegment

$$
\begin{aligned}
&L_a = L_b \Rightarrow \\
&P1 + u_a(P2 - P1) = P3 + u_b(P4 - P3) \Rightarrow \\
&P1_x + u_a(P2_x - P1_x) = P3_x + u_b(P4_x - P3_x) \\
&P1_y + u_a(P2_y - P1_y) = P3_y + u_b(P4_y - P3_y) \Rightarrow \\
&u_a = \frac{(P4_x - P3_x)(P1_y - P3_y) - (P4_y - P3_y)(P1_x - P3_x)}{(P4_y - P3_y)(P2_x - P1_x) - (P4_x - P3_x)(P2_y - P1_y)} \\
&u_b = \frac{(P2_x - P1_x)(P1_y - P3_y) - (P2_y - P1_y)(P1_x - P3_x)}{(P4_y - P3_y)(P2_x - P1_x) - (P4_x - P3_x)(P2_y - P1_y)}
\end{aligned}
$$

Värt att notera är att både u_a och u_b har samma nämnare.

Om nämnaren till u_a/u_b är 0 innebär detta att L_a och L_b är parallella.

Är både täljarrna och nämnaren till u_a samt u_b 0 innebär det att L_a och L_b är överlappande, vilket innebär att L_a och L_b korsas den gemensamma delen av L_a och L_b. Ett knepigt specialfall är då den gemensamma delen av L_a och L_b enbart är en punkt.

För linjesegment gäller dessutom att $0 \geq u_a, u_b \geq 1$ måste gälla.

När väl två giltiga värden på u_a, u_b hittats kan den korsande punkten x, y räknas ut med de tidigare definierade begreppen, t.ex. $x = P1_x + u_a(P2_x - P1_x)$ och $y = P1_y + u_a(P2_y - P1_y)$.

5 Trianglar

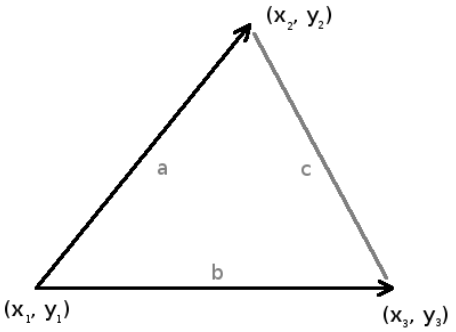
Ibland vill man räkna ut arean på trianglar. Då finns det två enkla formler. Vilken man använder beror på vilken information om trianglarna som är given. Vet man längderna så kan man använda Herons formel:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

där a, b och c är kantlängderna och s är hälften av omkretsen ($s = (a+b+c)/2$).

Vet man istället hörnpunkternas koordinater är det bättre att räkna på kryssprodukten. Absolutbeloppet på kryssprodukten mellan två vektorer blir nämligen arean av det uppspända parallellogrammet. Tar man därför två kanter som vektorer och delar absolutbeloppet av deras kryssprodukt med två får man därför triangelns area:

$$A = |(x_2 - x_1, y_2 - y_1) \times (x_3 - x_1, y_3 - y_1)|/2 = |(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)|/2$$

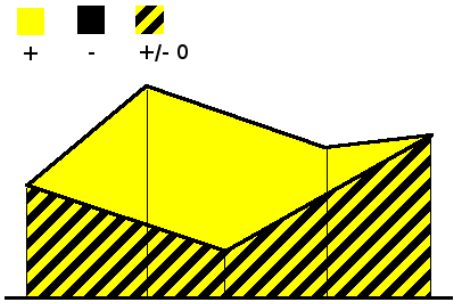


6 Polygoner

Även polygoner har areor, och även dem finns det två enkla sätt att räkna ut: antingen med trapetser eller med trianglar.

6.1 Area m.h.a. trapetser

Varje linjesegment i polygonen bildar en trapets med x-axeln (alternativt y-axeln om det är enklare, men nu antar vi att vi valde x-axeln). Dessa bidrar till polygonens area antingen positivt eller negativt. Det är inte uppenbart vilka som ska vara positiva och vilka som ska vara negativa, men om vi ser till att följa polygonen runt och alltid ta arean av trapetsen som: $(y_2 + y_1)(x_2 - x_1)/2$, där (x_1, y_1) är linjesegmentets startpunkt och (x_2, y_2) är linjesegmentets slutpunkt får vi en area som är rätt förutom kanske tecknet.

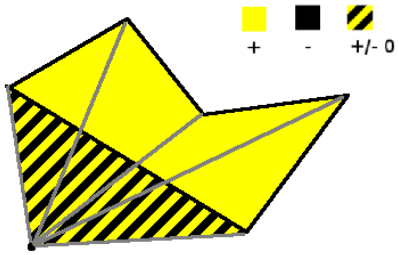


Så om vi har en enkelt sammanhängande polygon given av $(x_0, y_0), (x_1, y_1), ...(x_{n-1}, y_{n-1})$. Så får vi polygonens area av:

$$A = \frac{1}{2} |\Sigma_{k=1}^{n}(y_k + y_{k-1})(x_k - x_{k-1})|$$

6.2 Area m.h.a. trianglar

Använder vi trianglar gör vi på nästan samma sätt som för trapetser. Nu tar vi en godtycklig punkt (x, y) i planet och bildar trianglar alla trianglar bestående av punkten och linjesegment från polygonen. Formeln för arean av en triangel givet hörnens koordinater har vi redan från (5). Samma fråga som tidigare: vilka ska vara positiva och vilka negativa?



Och det är samma svar som tidigare: Följer vi bara polygonen runt så blir det rätt förutom kanske på tecknet:

$$A = \frac{1}{2} |\Sigma_{k=1}^{n}(x_{k-1} - x)(y_k - y) - (x_k - x)(y_{k-1} - y)|$$

6.3 Antalet "heltalspunkter" i en polygon

Det finns ett special fall för att räkna ut arean av en polygon om den uppfyller två kriterier. Den är representerad med heltalskoordinater och den är en simpel polygon. Denna egenskap beskrivs i Picks sats, vilket är som följer:

$$A = i + (b/2) - 1$$

Där A är den totala arean av polygonen, i är det totala antalet punkter med heltalskoordinater som är strikt innanför polygonens kanter. b är totala antalet heltalskoordinater som ligger direkt på polygonens kanter. Med hjälp Picks sats kan vi nu räkna ut antalet punkter med heltalskoordinater inuti polygonen:

$$i = A - (b/2) - 1$$

För att räkna ut b använder vi resultatet från 4.2. Lägger vi ihop för alla linjesegment (och drar ifrån ett från varje så att inte hörnen räknas två ggr) får vi:

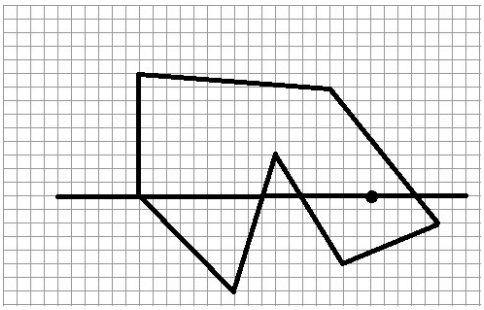
$$b = \Sigma_k GCD(\Delta x_k, \Delta y_k)$$

6.4 Punkt i Polygon

När man har en polygon kan det vara intressant att veta om en punkt man har ligger inuti denna eller om den ligger utanför polygonen. Detta kan, om polygonen är simpel, räknas ut med antalet skärningar av polygonens kanter.

Som man ser på bilden, kan man skapa en linje som går igenom polygonen. Sedan kan vi kolla hur många gånger vi skär polygonens kanter, om vi skär ett jämn antal kanter innan vi når punkten är den utanför, eftersom vi börjar utanför kommer varje jämn skärning vara att vi går ut. Man behöver dock inte spara antalet skärningar, räcker med en bool variabel som är false till att börja med.

Algoritm 2: Avgör om en punkt ligger i polygonen.
Input: En punkt p och en polygon P
Output: En boolsk sant eller falskt
InsidePoly(p, P)
(1) inside := false
(2) p_line := linje från punkten till oändligheten längs en horisontell linje
(3) **foreach** linjesegment $m \in P$
(4) **if** p_line och m korsar varandra
(5) inside = **not** inside
(6) **return** inside



Denna metod fungerar dock inte på komplexa polygoner med korsande kanter då detta kan ge fel värde fast den är innanför då den korsar kanter den inte ska korsa.

Då finns det en annan metod, och det är att man går runt alla polygonens kanter och summerar ihop vinkeln från punkten till linjesegmentets båda ändpunkter för alla linjesegment (och tillåter att vinkeln är negativ när ändpunkterna är i "omvänd ordning"). Summan kommer bli $2\pi m$ där m är antalet varv polygonen går runt punkten. Om summan är noll är punkten alltså utanför eftersom polygonen går noll varv runt punkten. Annars är punkten innanför.

6.5 Konvexa höljet

Det konvexa höljet är den polygon med minsta möjliga antal kanter som innefattar en hela punktmängden. För att lösa detta finns det bland annat en metod som kallas Graham scan som löser problemet i O(n log n):

Några delar borde dock gås igenom lite närmare. Om man gör en kryss produkt

Algoritm 3: Hitta konvexa höljet till en punktmängd
Input: En punktmängd P
Output: Minsta polygon som omfattar alla punkter
GrahamScan(P)
(1) Sök upp den punkt som har den absolut lägsta y värdet, använd lägsta x för tie breaker. O(n)
(2) Sortera punkterna på vinkel mot x planet sett från punkten man fann i förra steget. Om två punkter har samma vinkel ta bort den som är närmast utgångs punkten.
(3) Lägg utgångs punkten och de två nästkommande från sorteringen på en stack.
(4) **foreach** $i \in [3n-1]$
(5) Lägg P_i på stacken.
(6) **while** stackens tre översta punkter inte bildar vänstersväng
(7) Ta bort näst översta elementet i stacken.
(8) **return** Punkter i stacken

mellan de tre senaste punkterna ser man om det är en höger eller vänstersväng. Positivt för vänster negativt för höger, får man noll är punkterna kolinjära. Är det negativt värde säger vi att det är falskt i kollen i sista loopen. Detta görs i linjär tid då varje punkt inte kollas mer än två gånger i värsta fallet (efter andra gången tas den bort.

Så det som händer när en punkt ratas är att den helt sonikt hoppas över. Som bilden visar när det sker en "fel sväng" så tar man bort just den punkten och leder förbi den.

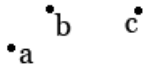
7 Cirklar

7.1 Ett centrum och en radie

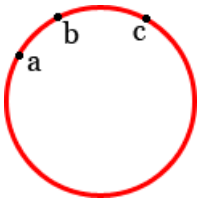
Vanligast är att vi representerar en cirkel med hjälp av ett centrum och en radie. Radien är avståndet från centrum till cirkelns kant, dvs alla punkter på kanten har ekvidistanta avstånd från centrum.

7.2 Entydligt given av tre punkter

Givet de tre punkterna a, b och c på ett plan (där punkterna inte ligger i linje med varandra) kan man entydligt hitta en cirkel där de tre punkterna alla ligger på cirkelns kant. En annan punkt existerar på planet som är ekvidistant från de tre punterna, vilket innebär att cirkelns centrum (punkten på det ekvidistant avstånd från a, b, c) samt radie (det ekvidistanta avståndet) existerar.



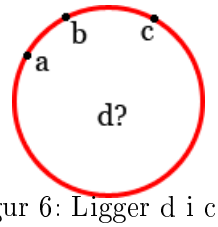
Figur 4: 3 punkter i planet



Figur 5: Cirkeln bestämd efter punkterna.

7.3 Ligger en punkt på, utanför eller inuti cirkeln?

Genom att använda att man av 3 punkter på planet entydligt kan bestämma en cirkel kan man räkna ut huruvida en fjärde punkt ligger på, utanför eller inuti cirkeln. Anta att vi har en cirkel bestående av punkterna a, b samt c. Ligger punkten d på, utanför eller inuti cirkeln?



Figur 6: Ligger d i cirkeln?

Beräkna determinanten D:

$$\begin{vmatrix} a_x & a_y & a_x^2 a_y^2 & 1 \\ b_x & b_y & b_x^2 b_y^2 & 1 \\ c_x & c_y & c_x^2 c_y^2 & 1 \\ d_x & d_y & d_x^2 d_y^2 & 1 \end{vmatrix}$$

Är D negativ ligger d utanför cirkeln, är D positiv ligger d innanför cirkeln. Är D 0 innebär det att punkten d ligger på cirkelns kant.

Om vi istället har cirkelns radie, cirkelns centrum samt en punkt och undrar huruvida den ligger inom cirkeln går också ovanstående formel att använda, då genom att ta ut tre punkter från cirkeln och använda dessa som a, b och c.