Jöran Petersson

Algoritmer + datastrukturer = program

Gymnasieelevers fråga om hur miniräknaren beräknar ”roten ur” kan fördjupa deras matematikkunskaper om exempelvis iterationsformler, stoppvärden och intervallhalvering. Författaren visar hur programmering kan användas som ett medel för att ge eleverna svar på frågan och hur svaret både kan förfinas och leda till generaliseringar.

Rubriken ovan är titeln på en känd bok vars författare Niklaus Wirth är prisbelönt programmeringsdidaktiker, men kanske mer känd som den som, just med syftet att undervisa om programmering, skapade Pascal och en rad andra programmeringsspråk. Boktiteln understryker att innan man börjar programmera måste man ha ett innehåll – nämligen algoritmer och datastrukturer. Det innebär att programmering inte blir ett mål i sig utan istället ett medel för att lösa ett problem. I fallet matematikundervisning kan sådana problem hämtas dels inifrån själva matematiken, till exempel hur miniräknaren beräknar kvadratrötter och dels från tillämpningar utanför matematiken, exempelvis med en principmodell för hur blodsockerkurvan ser ut och kan jämföras efter en läsk och efter ett knäckebröd.

Hur beräknar miniräknaren ”roten ur”?

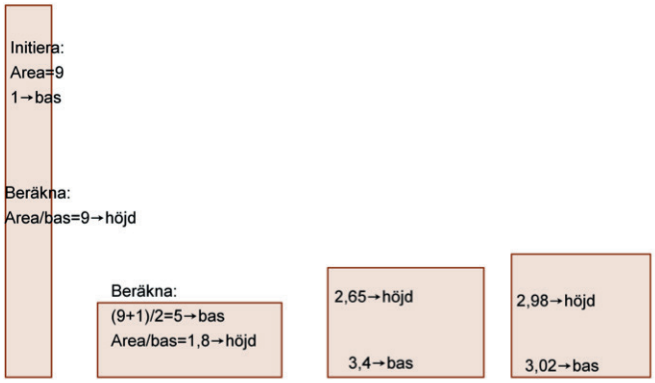
Redan under min första termin som lärare på gymnasiet, och flera gånger senare, fick jag frågan av eleverna i årskurs 1: ”Magistern, hur gör miniräknaren för att beräkna roten ur?” Det är ett gyllene tillfälle när eleverna kommer med grundade frågor, som rimligen kräver matematiska svar. Just denna fråga kan besvaras på flera sätt som kan anpassas dels till elevernas nivå och dels till vilket matematikinnehåll man vill behandla. Ett kort svar är att endast ge en iterationsformel. Ett något längre svar är att även diskutera stoppvillkor för iterationen, men det finns också flera längre svar att ge.

Om eleverna är bekanta med räta linjens ekvation, så kan man med sekantmetoden härleda en iterationsformel som bestämmer $\sqrt{A}$ som nollstället till funktionen $f(x) = x^2 - A$. Kan eleverna derivator, så går det att göra detsamma med Newton-Raphsons metod. Detta kan även göras med intervallhalvering, vilket endast kräver grundläggande kunskap om funktionsbegreppet. Här ges en geometrisk framställning, som ger samma iterationsformel som med Newton-Raphsons metod, men endast kräver förkunskaper motsvarande aritmetiskt medelvärde och att bestämma arean av en rektangel.

En algoritm för ”roten ur”

Innan vi börjar lösa problem med okända svar, kan det vara bra att börja med ett problem med känd lösning. Vi väljer därför att beräkna kvadratroten av 9. Då får eleverna ett problem med en känd lösning och kan använda det för att utveckla en algoritm som fungerar även för problem med okända lösningar.

Tänk dig en rektangel med arean 9 och basen 1. Då blir höjden 9/1 = 9. Eftersom vi vill skapa en kvadrat, alltså en rektangel med samma värden på bas och höjd, blir nästa steg att välja ny bas som vilket värde som helst mellan den gamla basen och höjden. Ett välbekant sätt att göra detta är medelvärdet *(gammal bas + gammal höjd)/2 → ny bas* och beräkna *area/ny bas → ny höjd*. Här inför vi begreppet iteration. Genom att upprepa (iterera) detta får vi en serie rektanglar, alla med arean 9, och vars höjd och bas närmar sig varandra, dvs det liknar alltmer en kvadrat.



Vi har nu med ett matematiskt resonemang tagit fram ett utkast till en algoritm. Ett lämpligt nästa steg är att införa begreppet konvergens – att följden av rektanglar konvergerar mot en kvadrat. En algoritm som inte konvergerar – inte ”kommer i mål” – gör ingen programmerare glad.

Hade vi tagit fram samma algoritm med exempelvis Newton-Raphsons metod, hade det krävts betydligt mer matematik för att bevisa konvergensen, men just denna geometriska framställning ger oss möjlighet att ge eleverna ett mycket kort och enkelt men samtidigt stringent bevis för konvergensen, nämligen att medelvärdet alltid ligger mellan gammal bas och gammal höjd. Det gör att i varje steg trängs den nya basen och nya höjden in i ett allt smalare intervall för att så småningom hamna så nära varandra att vi är nöjda med svaret och kan avsluta iterationen. Därmed väcks frågan om stoppvillkor: Hur liten ska differensen bas minus höjd vara för att vi ska vara nöjda? Nu har vi alla ingredienser i algoritmen och kan sammanfatta:

- *Initiera*. Välj startvärde, exempelvis alltid tilldela 1 → bas.
- *Iterera*. Area/bas → höjd. (bas + höjd)/2 → höjd.
- *Pröva stoppvillkor.* Stanna om (bas minus höjd) är tillräckligt litet. Annars repetera.

Notera att även om flera programmeringsspråk använder likhetstecknet som tilldelningssymbol, kan det vara lämpligt att använda pilar som symbol för tilldelning för att inte blanda ihop likhetstecknets dynamiska och statiska betydelse för eleverna. Dessutom är det ju en tilldelning och inte en ekvation det handlar om. När vi nu har formulerat grundalgoritmen, så vet vi också vilken datastruktur vi behöver, nämligen indata i form av area; de interna beräkningsvariablerna bas och höjd varav en av dessa blir utdata; samt ett i förväg bestämt värde på stoppvillkoret. Därmed är det dags för nästa fas i arbetet – att pröva och felsöka algoritmen.

Skapa och testa algoritmen, och vid behov felsöka

När grundalgoritmen väl är klar så är det lämpligt att simulera om den alls fungerar och i så fall hur den fungerar i praktiken, exempelvis om konvergensen är långsam eller snabb. För detta exempel är kalkylblad som Excel smidigt att använda av flera skäl. I kalkylblad kan eleverna skriva en programmeringsinstruktion i taget och se att syntaxen fungerar innan de börjar med nästa programmeringsinstruktion. Kalkylblad låter också eleverna se resultatet i varje steg av beräkningarna och därmed göra en rimlighetsbedömning om de har skrivit in rätt formel. Programvara för kalkylblad har i regel en hjälpfunktion så att syntaxen blir rätt. En not är att de flesta programvaror för kalkylblad har likadan syntax. Alltså kan ett kalkylblad skapat i exempelvis Microsoft, öppnas i t ex LibreOffice.

	A	B	C	D
1	**Grundalgoritm för kvadratrot**			
2	startvärde	1		
3	Area	9		
4	(bas+höjd)/2→bas	Area/bas→höjd	intervallängd	krympfaktor
5	1	9	8	
6	5	1,8	3,2	0,4
7	3,4	2,647058824	0,752941176	0,235294118
8	3,023529412	2,976653696	0,046875715	0,062256809
9	3,000091554	2,999908449	0,000183105	0,00390619
10	3,000000001	2,999999999	2,79397E-09	1,52588E-05
11	3	3	0	0
12	3	3	0	#DIV/0!

Cell	Formel i kalkylblad
A5	=B2
A6 (kopieras nedåt)	=(A5+B5)/2
B5 (kopieras nedåt)	=B$3/A5
C5 (kopieras nedåt)	=ABS(B5-A5)
D6 (kopieras nedåt)	=C6/C5

Rutorna i kalkylblad kallas celler. I en cell kan det stå text eller formler. Tabellen ovan visar vilka formler som står i cellerna. Dollartecknet i cell B5 gör att formeln B$3/A5 blir B$3/A6 när det kopieras nedåt, alltså att den alltid hänvisar till cellen med värdet på arean. Om istället formeln B3/A5 kopieras nedåt, så blir den B4/A6, vilket ger ett felmeddelande eftersom cell B4 innehåller text. Notera att i ruta B12 ges ett felmeddelande vid division med noll eftersom algoritmen konvergerade så pass väl att intervallängden blev noll. Krympfaktorn definieras i kalkylbladet som en intervallängd delad med förra intervallängden.

Att förbättra en algoritm

Trots att datorer räknar snabbt, finns det ibland behov av att modifiera en algoritm så att den kräver färre iterationer, åtminstone om varje iteration kräver omfattande beräkningar. Det kan gå till på följande sätt och illustrerar även att i en del sammanhang är matematiken ett empiriskt ämne som uppmuntrar till experimentlusta. Om vi använder vår algoritm, finner vi att slutvärdet 3 ligger betydligt närmre startbasen 1 än starthöjden 9 och detta blir tydligare om vi prövar algoritmen på ett större tal, exempelvis $\sqrt{36}$. Medelvärdet väljer ett värde mitt emellan dessa och ett förslag är att byta medelvärdet mot ett värde som ligger lite närmre 1 än 9. Det kan vi göra på följande sätt: Notera att medelvärdet av bas och höjd kan skrivas som $0{,}5 \cdot \text{bas} + 0{,}5 \cdot \text{höjd}$, vilket motsvarar att bas och höjd viktas lika. Kan vi få färre iterationer om vi istället väljer större vikt på det mindre av värdena bas och vikt? Notera att summan av vikterna måste vara exakt 1. Ett exempel på detta är att i cell A6 använda iterationsformeln $0{,}7 \cdot \min(A5; B5) + 0{,}3 \cdot \max(A5; B5)$ och kopiera denna nedåt.

Viktad uppdatering rotalgoritm			
startvärde	1	Vikt 1	0,7
Area	9	Vikt 2	0,3
bas	höjd	intervallängd	krympfaktor
1	9	8	
3,40	2,65	0,75	0,09
2,87	3,13	0,26	0,34
2,95	3,05	0,10	0,38
2,98	3,02	0,04	0,39
2,99	3,01	0,02	0,40
3,00	3,00	0,01	0,40
3,00	3,00	2,5E-03	0,40
3,00	3,00	9,9E-04	0,40
3,00	3,00	3,9E-04	0,40

I grundalgoritmen ser vi att med medelvärdet, dvs viktningen 0,5 krymper intervallets längd långsamt i början men allt fortare mot slutet. Läroböcker i numerisk analys visar att konvergensen är kvadratisk, dvs konvergensen accelererar. I den viktade uppdateringen ser vi att intervallets längd krymper fort i början men att krympfaktorn sedan stannar vid ungefär 0,4 – alltså snabb konvergens i början men sedan med en konstant faktor, vilket kallas linjär konvergens. Vår slutsats blir att det bästa vore att ha den alternativa viktningen några få iterationer i början och sedan byta till viktning 0,5. Eleverna kan experimentera sig fram till en lämplig viktning och lämpligt antal iterationer. Kanske någon elev föreslår att istället för att ha en bestämd viktning ett bestämt antal iterationer i början, så kan man ha den alternativa viktningen så länge kvoten *max(bas/höjd)/min(bas, höjd)* är tillräckligt stor, där gränsen för ”tillräckligt stor” kan experimenteras fram. För att göra detta krävs en villkorssats, vilket i kalkylblad kan se ut ungefär så här:

```
=IF(max(bas/höjd;höjd/bas)<5;0,5*bas+0,5*höjd;0,7*min(bas;höjd)+
0,3*max(bas;höjd))
```

Algoritmer för startvärden

Om eleverna prövar algoritmen ovan på stora tal som 150 000 och små tal som 0,000 015, så finner de att det krävs ett flertal iterationer innan algoritmen kommer i närheten av slutvärdet, men att konvergensen sedan går snabbt. Om det gick att starta närmre slutvärdet, skulle färre iterationer behövs. En del matematiska problem är så knepiga att algoritmer för dem ofta inte konvergerar alls om man inte startar tillräckligt nära målet. Därför är en viktig del av matematiskt arbete att hitta en metod som ger goda startvärden.

För att hitta goda startvärden i kvadratrotsalgoritmen kan eleverna dra nytta av kunskaper i algebra och om potenser. Kanske några elever föreslår omskrivningen $\sqrt{150\,000} = 1\,000 \cdot \sqrt{0{,}15}$, alltså att bryta ut en faktor 10^{2n} så att den andra faktorn hamnar i intervallet [0,1; 10). En algoritm för denna strategi för att hitta startvärden kan se ut så här:

$0 \rightarrow$ dubbelpotens.

Så länge A>10; repetera $A/100 \rightarrow$ A, dubbelpotens + 1 $\rightarrow$ dubbelpotens.

Så länge A<0,1; repetera $A \cdot 100 \rightarrow$ A, dubbelpotens – 1 $\rightarrow$ dubbelpotens.

Beräkna $\sqrt{A}$.

När roten är beräknad, multiplicera den med $10^{\text{dubbelpotens}}$.

Datorer använder en variant av just denna startvärdesalgoritm genom att tal representeras i binär grundpotensform. Vi ser att kvadratroten ur vårt tidigare exempel är $\sqrt{1{,}5 \cdot 10^5} = \sqrt{1{,}5} \cdot \sqrt{10} \cdot 10^2$. Med beteckningen $\textit{mantissa} \cdot \sqrt{10} \cdot 10^2$ där $10^0 < \textit{mantissa} < 10^1$ ser vi att ett lämpligt startvärde för mantissan är $10^{0{,}5} = \sqrt{10} \approx 3{,}2$. En lämplig och effektiv startvärdesalgoritm blir därför följande:

Sätt startvärdet till 3,2 och beräkna roten ut mantissan.

Justera till rätt tiopotens och om tiopotensen var udda dessutom med faktorn $\sqrt{10} \approx 3{,}2$.

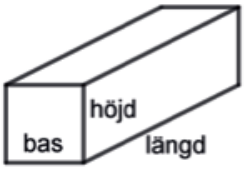
Algoritmen är klar: Presentera resultatet.

Nu är alla moduler i programmet klara och det är dags att koda i valfritt programmeringsspråk, exempelvis i matematikprogrammet Octave. För tydlighetens skull är det ofta en fördel att använda någorlunda beskrivande och inte alltför korta namn på variablerna. Det är en god vana att skriva förklarande kommentarer i programmet så att en annan person som vill ändra i programmet kan förstå vad de olika delarna gör. Den förklarande texten fungerar även som minnesanteckningar för programmeraren själv och bör beskriva programmets begränsningar, exempelvis att vi här inte har hanterat fallen A = 0, A < 0 och A icke-reellt.

Att generalisera en algoritm

Pólya föreslog ett arbetssätt för problemlösningen med de fyra stegen *förstå*, *planera*, *genomföra* och *reflektera* över svaret. Även om eleverna har frågat om rötter i allmänhet är det därför lämpligt att börja i specialfallet med kvadratroten då det är enklare att förstå och att det kan ge oss verktyg för det fortsatta arbetet – att göra en plan och genomföra den. Ett exempel på Pólyas fjärde

steg kan vara om och hur vi kan generalisera algoritmen för kvadratrötter till kubikrötter och högre rötter. Generaliseringen kan påbörjas antingen som en diskussion i helklass eller som utmanande uppgift till några elever i klassen.

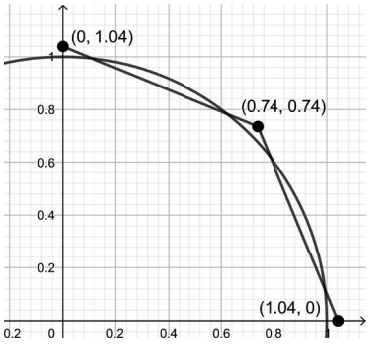


För att beräkna kubikroten ur volymen V kan vi som startvärden ansätta ett rätblock med bas = höjd = 1 och längd = V. Medan en rektangel har två geometriska sidor, så har rätblocket tre geometriska sidor att beräkna medelvärdet av och detta är den enda skillnaden. Eleverna kan förstås även experimentera med viktning och startvärden precis som med algoritmen för kvadratroten. Efter denna generalisering kanske eleverna vågar sig på att generalisera ytterligare genom att beräkna n:te roten genom att ta medelvärdet på sidorna på ett tänkt n-dimensionellt rätblock.

Vi kan generalisera ytterligare ett steg och fundera över hur vi skulle kunna beräkna potenser i allmänhet. Som ett exempel, kan eleverna faktorisera $x^{3,72} = x^3 x^{0,72}$ och inse att endast $x^{0,72}$ kräver specialhantering. Eftersom vi kan skriva om decimaltalet 0,72 som bråket 72/100 = 16/25, så skulle vi kunna beräkna potensen $x^{0,72}$ som $\sqrt[25]{x^{16}}$ eller som $(\sqrt[25]{x})^{16}$. Att beräkna n:te roten ur har vi redan utrett och att beräkna en heltalspotens är endast upprepad multiplikation. Därmed är problemet löst om vi bortser från utmaningen att hålla avrundningsfelen på en låg nivå.

Styckevis approximation

Men hur gör då miniräknaren för att beräkna logaritmer och trigonometriska funktioner? Här finns inga enkla iterationsformler. Istället används styckevis approximation, vilket innebär att med Taylorutveckling bestämma ett polynom av låg grad, ofta en linjär funktion, som ger en god approximation i ett kort intervall. Att härleda en Taylorutveckling kräver kunskap om partiell integration, vilket kommer sent i gymnasiematematiken. I tidigare kurser är ett alternativ att läraren ger ett färdigt polynom eller att eleverna själva får i uppgift att ta fram ett linjärt uttryck. Exempelvis kan de ta fram två närliggande punkter på cirkelns ekvation och på papper eller med geometriprogram mäta vilka vinklar de motsvarar. Vi illustrerar med följande exempel.



Ekvationen $y = \dfrac{(0,74 - 1,04)}{(0,74 - 0)} \cdot \dfrac{(x - 45)}{(90 - 45)}$ beskriver linjestycket genom

(0; 1,04) och (0,74; 0,74). Detta linjestycke approximerar sinus(x) i intervallet [45; 90] grader med en noggrannhet av ungefär 0,04 (en ruta). För intervallet [0; 45] grader ser linjen ut att ligga nära cirkeln. Dock lutar linjen brant, vilket gör att det vertikala avståndet är stort, som mest cirka 3 rutor (0,12). En viktig slutsats att understryka för eleverna är att ju brantare linjen lutar, desto kortare styckeindelning behövs.

Sammanfattning

När grafritande miniräknare slog igenom på 1990-talet var en diskussionspunkt vilket fabrikat eleverna skulle rekommenderas köpa. Eftersom teknikträskeln är betydligt högre för denna typ av miniräknare än för en vanlig fickräknare var detta en relevant fråga då en del lärare kände sig mer bekanta med en viss modell eller fabrikat än en annan. Fokus låg dock på hur miniräknarna skulle användas för att stödja det matematiska lärandet. På samma sätt kan det vara relevant att fundera över vilket programmeringsspråk den enskilde läraren kan, men att huvudfokus ska vara på hur programmering kan användas för att stödja det matematiska lärandet. Just rubriken på denna artikel antyder att vi, innan vi börjar programmera, måste ha ett relevant problem att lösa, vilket kräver någon form av algoritmer och datastrukturer. Först därefter är det dags att börja programmera, annars riskerar det att bli det som kallas ”torrsim” och programmering riskerar blir lite som att lära sig grammatik utan varken någon att prata med eller något att prata om. Det är lämpligt att dela upp algoritmen i moduler, som var för sig kan programmeras, felsökas och finslipas. Annars blir programmeringsarbetet ofta svåröverskådligt även för det enkla fallet att ta fram ett program för att beräkna kvadratroten. För att ta fram en algoritm för rotberäkningar, har denna artikel berört flertalet av läroplanens matematikområden. Nämligen problemlösning i allmänhet, beräkningsmetoder i tal, algoritmer i algebra, geometriska illustrationer samt att tolka tabeller.

LITTERATUR

Wirth, N. (1976). *Algorithms + data structures = programs*. Englewood Cliffs, N.J.: Prentice-Hall.

Artikeln följs upp i nästa nummer av Nämnaren.