SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Euklides algoritm - i teori och praktik

av

Sebastian Timosson

2023 - L5

Euklides algoritm - i teori och praktik

Sebastian Timosson

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Olof Sisask

2023

Innehåll

Abstract

Euclid's algorithm, named after the ancient Greek mathematician Euclid, is a fundamental method in mathematics for computing the greatest common divisor (GCD) of two integers. Despite its ancient origins, Euclid's algorithm remains relevant and widely used today due to its efficiency and its numerous theoretical and practical applications. This paper explores the depth of Euclid's algorithm, its complexity, and its performance in both worst-case and average-case scenarios. We also examine a modified version of Euclid's algorithm and compare its performance with the original version. Furthermore, we delve into the algorithm's role in number theory, including its connection to the fundamental theorem of arithmetic and prime factorization. The paper includes proofs of key theorems such as Bézout's identity and Euclid's lemma, which are central to the understanding of these topics. The exploration of these concepts provides a deeper understanding of the structure and properties of these mathematical structures and the central role of Euclid's algorithm within them.

1 Introduktion

Euklides algoritm, uppkallad efter den antika grekiska matematikern Euklides, är en effektiv metod inom matematiken för att beräkna den största gemensamma delaren (SGD) av två heltal a och b. SGD, ofta betecknad som $sgd(a, b)$, är det största heltal som kan dela både a och b utan att lämna någon rest.

Trots att algoritmen är namngiven efter Euklides, som först beskrev den i sitt verk 'Elementa' runt 300 f.Kr., verkar den inte ha upptäckts av Euklides själv. Istället samlade han resultat från tidigare matematiker. Euklides algoritm är en av de äldsta kända algoritmerna som fortfarande används idag och har många teoretiska och praktiska tillämpningar [1]. Den används bland annat för att reducera bråk till deras enklaste form, utföra division i modulär aritmetik, lösa diofantiska ekvationer och hitta irreducibla faktorer i polynom.

I detta arbete kommer vi att undersöka några av de praktiska tillämpningarna av Euklides algoritm och djupdyka i dess komplexitet för att bedöma dess effektivitet. Vi kommer att utforska värsta-fall-scenariot för algoritmen, där vi analyserar de specifika villkoren under vilka algoritmen tar längst tid att slutföra. Dessutom kommer vi att undersöka det genomsnittliga antalet steg som algoritmen tar för olika par av tal. Denna analys kommer att ge oss en djupare förståelse för algoritmens typiska prestanda i praktiken. Slutligen kommer vi att undersöka en modifierad version av Euklides algoritm och jämföra dess prestanda med den ursprungliga versionen.

Vidare kommer vi att dyka djupare in i talteorin och undersöka hur Euklides algoritm kopplar till aritmetikens fundamentalsats och primtalsfaktorisering. Vi kommer att bevisa viktiga satser som Bézouts identitet och Euklides lemma, och visa hur dessa leder till beviset för aritmetikens fundamentalsats. Slutligen kommer vi att utforska hur dessa koncept kan generaliseras till polynomringar, och visa att Euklides algoritm fortfarande spelar en central roll i denna mer generella kontext.

Euklides algoritm fungerar genom att upprepade gånger dela det större talet med det mindre och sedan ersätta det större talet med resten från divisionen. På detta sätt minskar talen stegvis tills vi når noll. Det sista icke-nolltalet som vi får är den största gemensamma delaren. Denna process kan beskrivas mer formellt med följande sats:

Sats 1.1. *Om två heltal $a > b$ är givna, där $b > 0$, så kan man skriva $a = q_0 b + r$, där q_0 är ett heltal och $b > r \geq 0$. Vi kallar q_0 för kvoten och r för den principala resten.*

Bevis. Tänk på hur vi normalt utför division. Om vi har två heltal a och b och vi vill dela a med b, vad vi gör är att vi subtraherar b från a så många gånger som möjligt tills vi inte kan göra det längre utan att få ett negativt tal.
Antalet gånger vi kan subtrahera b från a är kvoten q, och det tal som vi har kvar

efter alla dessa subtraktioner är resten r. Eftersom vi stoppar subtraktionerna innan vi får ett negativt tal, vet vi att $b > r \geq 0$.
Så vi har $a = q_0 b + r$ och $b > r \geq 0$, vilket är precis vad vi ville visa. Att q och r är unika följer direkt från definitionen av division med rest. □

Delbarhet är ett centralt koncept i Euklides algoritm. Algoritmen fungerar genom att upprepade gånger dela tal och betrakta resten, tills vi hittar ett tal som delar både a och b utan rest, vilket är den största gemensamma delaren.

Definition 1.2 (Delbarhet)**.** *Låt a och b vara två heltal. Vi säger att a delar b, skrivet $a|b$, om det finns ett heltal c sådant att $b = ac$*

Nu när vi har definierat vad det innebär för ett tal att dela ett annat, kan vi bevisa följande sats:

Sats 1.3. *En gemensam delare till a och b måste även dela $r(= a - q_0 b)$, och en gemensam delare till b och r måste även dela a. Alltså är $sgd(a, b) = sgd(b, r)$.*

Bevis. Om d är en gemensam delare till a och b, då finns det heltal m och n sådana att $a = md$ och $b = nd$. Då är $r = a - q_0 b = md - q_0 nd = d(m - q_0 n)$. Eftersom $m - q_0 n$ är ett heltal, följer det från definitionen av delbarhet att d delar r.

Å andra sidan, om d är en gemensam delare till b och r, då finns det heltal p och q sådana att $b = pd$ och $r = qd$. Då är $a = b + q_0 r = pd + q_0 qd = d(p + q_0 q)$. Eftersom $p + q_0 q$ är ett heltal, följer det från definitionen av delbarhet att d delar a. Vidare, en gemensam delare till b och r måste även dela a.

Därför är de gemensamma delarna till a och b samma som de gemensamma delarna till b och r, vilket innebär att $\text{sgd}(a, b) = \text{sgd}(b, r)$. □

Med detta kan vi nu går vidare och definiera Euklides algoritm.

Definition 1.4 (Euklides algoritm)**.** *Euklides algoritmen är en process för att hitta den största gemensamma delaren g mellan två heltal $a > b > 0$. Algoritmen består av följande steg:*

1. *Skriv $a = q_0 b + r$ där q_0 är kvoten och r är resten, med $b > r \geq 0$.*
2. *Om $r = 0$, avsluta processen. Talet b är då den största gemensamma delaren g. Annars, gå tillbaka till steg 1 med (b, r) istället för (a, b).*

Nu när vi har definierat Euklides algoritm, kan vi bevisa några viktiga egenskaper hos den. Först ska vi visa att den sista nollskilda resten i algoritmen faktiskt är den största gemensamma delaren av de ursprungliga talen.

Sats 1.5. *Den sista nollskilda resten i Euklides algoritm på talparet (a, b) är lika med $sgd(a, b)$.*

Bevis. Beteckna de successiva resterna i Euklides algoritm som $r_1, r_2, \ldots, r_n, r_{n+1} = 0$, där r_n är den sista nollskilda resten på talparet (a, b). Enligt Sats 1.3, har vi att $\text{sgd}(a, b) = \text{sgd}(b, r_1) = \text{sgd}(r_1, r_2) = \ldots = \text{sgd}(r_{n-1}, r_n)$. Eftersom r_n är den sista nollskilda resten, är nästa rest $r_{n+1} = 0$. Då har vi att $\text{sgd}(r_n, r_{n+1}) = \text{sgd}(r_n, 0) = r_n$. Därför är $\text{sgd}(a, b) = r_n$. □

Ett annat viktigt resultat gäller hur effektiv Euklides algoritm är. Vi ska nu visa att algoritmen alltid terminerar inom ett begränsat antal steg.

Sats 1.6. *Euklides algoritm på talparet (a, b), där $a > b > 0$, tar högst b steg att terminera.*

Bevis. Vid varje steg av algoritmen minskar vi det större talet av (a, b) till resten r, som är mindre än b, och då måste algoritmen terminera inom högst b steg. □

Men innan vi fortsätter med att förstå hur denna algoritm fungerar med några exempel, låt oss introducera en funktion som representerar tillämpningen av Euklides algoritm på ett par tal. Vi kallar denna funktion $\text{EUKL}(a, b)$.

Definition 1.7. *Vi definierar* $\text{EUKL}(a, b)$ *som resultatet av att tillämpa Euklides algoritm på talparet (a, b). Detta ger oss den största gemensamma delaren mellan a och b. Vi antar att $a > b > 0$ för att tillämpa algoritmen korrekt.*

Nu är vi redo att visa några exempel på hur algoritmen fungerar.

Exempel 1.8. $\text{EUKL}(91, 50)$
$91 = 1 \cdot 50 + 41$
$50 = 1 \cdot 41 + 9$
$41 = 4 \cdot 9 + 5$
$9 = 1 \cdot 5 + 4$
$5 = 1 \cdot 4 + 1$
$4 = 4 \cdot 1$

$SGD(91, 50) = 1$

Exempel 1.9. EUKL$(90, 45)$
$90 = 2 \cdot 45$

$SGD(90, 45) = 45$

Exempel 1.10. EUKL$(89, 55)$
$89 = 1 \cdot 55 + 34$
$55 = 1 \cdot 34 + 21$
$34 = 1 \cdot 21 + 13$
$21 = 1 \cdot 13 + 8$
$13 = 1 \cdot 8 + 5$
$8 = 1 \cdot 5 + 3$
$5 = 1 \cdot 3 + 2$
$3 = 1 \cdot 2 + 1$
$2 = 2 \cdot 1$

$SGD(89, 55) = 1$

Som vi kan se i exemplen ovan, kan antalet steg som krävs för att utföra Euklides algoritm variera betydligt, även för talpar där talen är relativt nära varandra. Detta kan verka förvånande vid första anblicken, men det finns en logisk förklaring till detta. I de följande avsnitten kommer vi att undersöka dessa olika "fall" som till exempel "Det värsta fallet av antal steg".

2 Algoritmens komplexitet

2.1 Värsta fallet av antal steg

När vi tillämpar Euklides algoritm på ett talpar ser vi att det alltid tar ett begränsat antal divisionssteg. Av detta kan vi också dra slutsatsen att det måste finnas en övre gräns för hur många steg algoritmen tar att utföra. Denna övre gräns eller så kallat "Det värsta fallet" av algoritmen inträffar när talparet är på varandra följande Fibonaccital [2]. Fibonaccital är tal som ingår i en heltalsföljd där varje tal är summan av de två föregående Fibonaccitalen; de två första talen är 0 och 1. Fibonaccitalen är en sekvens F_n som definieras enligt:

Definition 2.1 (Fibonaccital)**.**

$$F_n = \begin{cases} 0 & om\ n = 0 \\ 1 & om\ n = 1 \\ F_{n-1} + F_{n-2} & om\ n > 1 \end{cases}$$

Exempel 2.2. Fibonaccisekvensen börjar $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181$ osv.

Att algoritmens övre gräns inträffar när talparet är på varandra följande Fibonaccital förklaras av Gabriel Lamé i följande sats [2]:

Sats 2.3 (G.Lamé, 1845)**.** *För $N \geq 1$, låt a och b vara heltal där $a > b > 0$ sådana att Euklides algoritm tillämpad på a och b kräver exakt N divisionssteg. Då är $a \geq F_{N+2}$ och $b \geq F_{N+1}$. Vidare tar Euklides algoritm precis N steg för talparet (F_{N+2}, F_{N+1}).*

Denna sats har för övrigt det historiska anspråket att vara den första praktiska tillämpningen av Fibonacci-sekvensen; sedan dess har många andra tillämpningar av Fibonacci-tal till andra algoritmer och studier av algoritmer upptäckts.

För att bevisa denna sats, kommer vi att använda ett induktionsargument. Vi kommer att visa att ett visst påstående, som vi kallar $P(N)$, gäller för varje heltal $N \geq 1$. Detta påstående $P(N)$ lyder:

$P(N)$: *Om $a > b > 0$ och* EUKL(a, b) *tar N steg , då är $a \geq F_{N+2}$ och $b \geq F_{N+1}$.*

Bevis av Sats 2.3. Vi visar att påståendet $P(N)$ gäller för varje $N \geq 1$.
1) Vi börjar med basfallet $N = 1$. Algoritmen tar 1 steg när b delar a utan någon rest; de minsta heltalen för att detta ska vara sant är $a = 2$ och $b = 1$. Vilket överensstämmer med vårt påstående.

2) Vi antar nu att påståendet $P(N)$ gäller för $N = M$, där M är något godtyckligt tal. Då ska vi visa att det även gäller för $N = M+1$. Vi skall alltså visa:

$P(M+1)$: Om $a > b > 0$ och EUKL(a, b) tar $M + 1$ steg , då är $a \geq F_{M+3}$ och $b \geq F_{M+2}$.

Vi betraktar ett par av tal (a, b) där $a > b > 0$ och EUKL(a, b) tar $M + 1$ steg. Det första steget av EUKL(a, b) är $a = q_0 b + r$ där $b > r \geq 0$ för något $q_0 \in \mathbb{N}_0$. Först ska vi visa att $q_0 \geq 1$:

Claim: $q_0 \geq 1$

Bevis. Om $q_0 = 0$ skulle vi ha att $a = r < b$, vilket strider mot vårt antagande att $a > b$. Därmed måste $q_0 \geq 1$. □

Nu ska vi visa att Euklides algoritm tar precis N steg för talparet (F_{N+2}, F_{N+1}). Vi tillämpar Euklides algoritm på detta talpar och observerar följande:

$$\begin{aligned}
\text{Steg 1:} \quad & F_{N+2} = q_1 F_{N+1} + F_N, \\
\text{Steg 2:} \quad & F_{N+1} = q_2 F_N + F_{N-1}, \\
& \vdots \\
\text{Steg N:} \quad & F_3 = q_N F_2 + F_1.
\end{aligned}$$

Dessa steg visar att resterna vi får bildar Fibonacci-sekvensen baklänges. Eftersom det finns exakt N tal i Fibonacci-sekvensen från F_{N+2} till F_1, tar algoritmen precis N steg. Detta slutför beviset. □

Som en konsekvens av Sats 2.3 har vi en viktig följdsats som lyder:

Följdsats 2.4. *Om $0 \leq a, b < T$, då är antalet divisionssteg som krävs när Euklides algoritm tillämpas på a och b högst $\lceil \log_\phi(\sqrt{5}T) \rceil - 2$. [2]*

För att förstå varför detta är fallet, behöver vi först titta på hur snabbt Fibonaccitalen växer. Detta kan beskrivas med följande sats:

Sats 2.5. *Det n:te talet i Fibonacci-sekvensen, F_n, kan ges av formeln*

$$F_n = \frac{(\phi^n - (-\phi)^{-n})}{\sqrt{5}},$$

där ϕ är det gyllene snittet, $\phi = \frac{1+\sqrt{5}}{2}$.

Denna sats visar att Fibonacci-talen växer exponentiellt med n. Detta är anledningen till att antalet divisionssteg i Euklides algoritm är begränsat av en logaritmisk funktion av T.

Bevis för Följdsats 2.4. Med Sats 2.5 ser vi att det högsta antalet divisionssteg, N, uppstår när $a = F_{N+2}$ och $b = F_{N+1}$, där N är så stort som möjligt med $F_{N+2} < T$.

Vi kan skriva detta som $T \geq F_{N+2} + 1$. Enligt formeln för Fibonacci-talen vet vi att $F_{N+2} > \phi^{N+2}/\sqrt{5}$, där ϕ är det gyllene snittet. Därmed får vi att $T > \phi^{N+2}/\sqrt{5} + 1$. Detta innebär att $\phi^{N+2} < \sqrt{5}T - \sqrt{5}$.

För att hitta det största N så att $\phi^{N+2} < \sqrt{5}T - \sqrt{5}$, kan vi använda logaritmer. Vi får att $N + 2 < \log_\phi(\sqrt{5}T - \sqrt{5})$. Eftersom N måste vara ett heltal, får vi att N är högst $\lceil \log_\phi(\sqrt{5}T - \sqrt{5}) \rceil - 2$. Detta är det vi ville visa. □

Alltså det följdsatsen säger är att det finns ett tak för de högst antal steg som algoritmen tar att utföra som kan räknas ut med $\lceil \log_\phi(\sqrt{5}T) \rceil - 2$.

2.2 Genomsnittet av antal steg

Det är klart att antalet steg som krävs för att utföra algoritmen varierar beroende på de specifika talen vi använder. Men kan vi hitta ett genomsnittligt antal steg för alla talpar (a, b) där $a > b > 0$? Efter lite forskning hittar vi att det finns tre olika formler för att beräkna detta genomsnitt. Vilken formel vi väljer beror på hur vi väljer våra talpar (a, b). För att undvika förvirring mellan dessa formler, kommer jag i denna text att fokusera på den formel som jag anser passar bäst med de exempel jag kommer att presentera senare.

Sats 2.6. *Låt (a, b) vara ett par av naturliga tal sådana att a och b väljs oberoende och enhetligt från intervallet 1 till N med villkoret att $a \leq N$ och $a > b > 0$. Då är det förväntade antalet steg för Euklides algoritm att slutföra beräknat med följande formel:*

$$\frac{1}{N^2} \sum_{a=2}^{N} \sum_{b=1}^{a-1} T(a, b) = \frac{12 \ln 2}{\pi^2} \ln N + 0.06$$

*Bevis för denna sats kan hittas i [2], sida **[354]**.*

Denna sats beskriver hur det genomsnittliga antalet steg som Euklides algoritm tar kan beräknas när talparen (a, b) väljs som beskrivet. I formeln representerar $T(a, b)$ antalet steg som algoritmen tar för ett specifikt par (a, b).
Notera att den högra sidan av ekvationen, $\frac{12 \ln 2}{\pi^2} \ln N + 0.06$, ger en approximation för det genomsnittliga antalet steg. Detta innebär att för de valda talparen, förväntas algoritmen i genomsnitt slutföra efter ungefär $\frac{12 \ln 2}{\pi^2} \ln N + 0.06$ steg.

För att tydliggöra hur genomsnittsformeln appliceras i praktiken kommer vi nu att utföra Euklides algoritm på tio slumpmässigt valda talpar (a, b) där $a > b > 0$ och $a, b < 100$. För varje par kommer vi att räkna ut det faktiska antalet steg som algoritmen tar och jämföra detta med det förväntade antalet steg enligt genomsnittsformeln. Resultaten presenteras i Tabell 1 nedan:

Genom att analysera resultaten i Tabell 1 kan vi se att det faktiska genomsnittliga antalet steg över de tio testfallen är 3.4. Detta är betydligt lägre än det förväntade genomsnittet på 5.88 steg enligt genomsnittsformeln.
Vi observerade också det värsta möjliga fallet. Enligt Sats 2.4 skulle det värsta fallet kräva 9 steg. Detta bekräftades när vi testade algoritmen på paret $(89, 55)$, som är ett par av Fibonacci-tal, vilket krävde 9 steg. Detta är mer än dubbelt så många steg som det faktiska genomsnittet.
Dessa observationer tyder på att Euklides algoritm ofta presterar bättre än vad den teoretiska övre gränsen och genomsnittsformeln antyder. Även om det i värsta fall kan kräva ett betydande antal steg, är det faktiska genomsnittliga antalet steg ofta lägre.

Tabell 1: Jämförelse mellan faktiskt och förväntat antal steg med genomsnittsformeln för Euklides algoritm.

Talpar (a, b)	Faktiskt antal steg	Förväntat antal steg
(97, 86)	5	6.11
(98, 48)	2	6.13
(80, 20)	1	5.86
(90, 37)	4	6.02
(96, 49)	4	6.1
(69, 38)	5	5.67
(85, 14)	2	5.94
(85, 17)	1	5.94
(87, 34)	6	5.97
(43, 32)	4	5.04
Genomsnitt	3.4	5.88

2.3 En modifierad version av algoritmen

I den traditionella Euklides algoritmen, som definieras i Definition 1.4, bygger algoritmen på principerna som beskrivs i Sats 1.1 och Sats 1.3, där vi hittar den största gemensamma delaren g mellan två heltal a och b, där $a > b > 0$. Vi introducerar nu en modifierad version av denna algoritm, som har en viktig skillnad i hur resten r och kvoten q_0 beräknas. Denna modifiering gör att den modifierade Euklides algoritmen kan ta färre steg för att hitta den största gemensamma delaren g. Notera att vi i den modifierade algoritmen styr mot mindre rester vid varje steg, vilket kan resultera i färre totala steg jämfört med den ursprungliga algoritmen. Dessutom, genom att tillåta negativa kvoter och rester, får vi en större flexibilitet i algoritmens framsteg, vilket kan leda till en mer effektiv lösning i vissa fall.

Sats 2.7. *Om två heltal a och b är givna, där $b \neq 0$, så kan man skriva $a = q_0 b + r$, där q_0 är ett heltal och $|r| \leq |b|/2$. Vi kallar q_0 för kvoten och r för den principala resten.*

Bevis. Skriv $a = mb + s$ med $0 \leq s < b$; detta är möjligt enligt Sats 1.1. Om $s \leq |b|/2$ så är vi klara, med $r = s$ och $q_0 = m$. Annars, om $s > |b|/2$, så får vi ta $r = s - b$ om $b > 0$ och $r = s + b$ om $b < 0$. För kvoten q_0, vi tar $q_0 = m + 1$ om $b > 0$ och $q_0 = m - 1$ om $b < 0$. Detta ger oss att $a = q_0 b + r$ och $-|b|/2 \leq r < 0$, vilket är vad vi ville visa. □

Definition 2.8 (Modifierad Euklides algoritm)**.** *Den modifierade Euklides algoritmen består av följande steg:*

1. *Skriv $a = q_0 b + r$ där q_0 är kvoten och r är resten, med $|r| \leq |b|/2$. Att detta är möjligt är en följd av Sats 2.7. Observera att q_0, b och r kan vara både positiva och negativa.*

2. *Om $r = 0$, avsluta processen. Talet $|b|$ är då den största gemensamma delaren g. Annars, gå tillbaka till steg 1 med (b, r) istället för (a, b). Detta steg är identiskt med det i den traditionella Euklides algoritmen, som beskrivs i Sats 1.3.*

Definition 2.9. *Vi definierar* MEUKL(a, b) *som resultatet av att tillämpa den modifierade Euklides algoritmen på talparet (a, b). Detta ger oss den största gemensamma delaren mellan a och b.*

Det enklaste sättet att förstå hur detta går till är att visa algoritmen med ett par exempel där vi tar två slumpvalda talpar (a, b).

Exempel 2.10. MEUKL$(19, 5)$

$19 = 4 \cdot 5 - 1$
$5 = (-5) \cdot (-1)$

$SGD(19, 5) = 1$

Här ser vi att den modifierade algoritmen tar 2 steg vilket är 1 steg färre än de antal steg den ursprungliga algoritmen tar att utföra.

Exempel 2.11. MEUKL$(73, 13)$

$73 = 6 \cdot 13 - 5$
$13 = (-3) \cdot (-5) - 2$
$-5 = 2 \cdot (-2) - 1$
$-2 = 2 \cdot (-1)$

$SGD(73, 13) = 1$

Här ser vi att den modifierade algoritmen tar 4 steg att utföra vilket är 2 steg färre än de antal steg den ursprungliga algoritmen tar att utföra.

2.4 En unik talföljd i den modifierade Euklides algoritmen

En intressant aspekt av den ursprungliga Euklides algoritmen är att "Det värsta fallet" inträffar när de två ingångstalen är på varandra följande Fibonaccital. Så är det inte med den modifierade algoritmen. Istället, under vissa förutsättningar, verkar det som om "Det värsta fallet" inträffar när de två ingångstalen är termer i en annan talföljd. Denna talföljd, som jag har valt

att kalla för "Modifierade Euklides talföljd" kommer vi att definiera och undersöka i detta arbete.

Det är viktigt att notera att den modifierade Euklides algoritmen, liksom den ursprungliga, fortfarande garanterar att hitta SGD mellan två tal. Den potentiella fördelen med den modifierade algoritmen ligger i dess effektivitet under vissa förutsättningar, vilket kan vara av intresse i vissa tillämpningar.

För att hitta denna nya talföljd har jag gjort många tester och experiment. Jag har tittat på olika mönster i den modifierade Euklides algoritmen och hittat en speciell talföljd som verkar viktig för hur snabbt algoritmen fungerar. Detta arbete med att testa och upptäcka nya saker har varit en viktig del av mitt projekt och har hjälpt mig att förstå mer om hur algoritmen fungerar.

Definition 2.12 (Modifierade Euklides talföljd)**.**

$$A_n = \begin{cases} 1 & om\ n = 0 \\ 2 & om\ n = 1 \\ 2A_{n-1} + A_{n-2} & om\ n > 1 \end{cases}$$

Exempel 2.13. Den modifierade Euklides talföljden börjar: 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461, 80782, 195025, 470832, 1136689, 2744210, 6625109, 15994428, osv.

I likhet med Lamé's sats för den ursprungliga Euklides algoritmen, vill vi nu formulera en liknande sats för den modifierade Euklides algoritmen med vår egen talföljd A_n.

Sats 2.14. *För $N \geq 1$, låt a och b vara heltal där $|a| \geq 2|b| > 0$ sådana att* MEUKL(a, b) *kräver exakt N divisionssteg. Då är $|a| \geq A_N$ och $|b| \geq A_{N-1}$. Vidare tar* MEUKL(a, b) *precis N steg för talparet (A_N, A_{N-1}).*

För varje $N \geq 1$ definierar vi påståendet:
$P(N)$: *Om $|a| \geq 2|b| > 0$ och* MEUKL(a, b) *tar N steg , då är $|a| \geq A_N$ och $|b| \geq A_{N-1}$.*

Bevis av Sats 2.14. Vi ska visa att påståendet $P(N)$ gäller för varje $N \geq 1$.
1) Vi börjar med basfallet $N = 1$. Algoritmen tar 1 steg när b delar a utan rest. De minsta heltalen som uppfyller detta krav är $a = 2 = A_1$ och $b = 1 = A_0$, vilket överensstämmer med vårt påstående.

2) Vi antar nu att påståendet $P(N)$ gäller för $N = M$, där M är något godtyckligt tal . Då ska vi visa att det även gäller för $N = M + 1$. Vi skall alltså

visa: $P(M+1)$: Om $|a| \geq 2|b| > 0$ och MEUKL(a,b) tar $M+1$ steg , då är $|a| \geq A_{M+1}$ och $|b| \geq A_M$.

Vi tar ett talpar (a,b) där $|a| \geq 2|b| > 0$ och MEUKL(a,b) tar $M+1$ steg. Anta att första steget är: $a = q_0 b + r$, där $|r| \leq |b|/2$. Vi behöver först visa att $|q_0| \geq 2$:

Claim: $|q_0| \geq 2$.

Bevis. Om $|q_0| = 0$ då vore $|a| = |r| \leq |b|$ vilket är en motsägelse eftersom vi har antagit att $|a| \geq 2|b|$. Om $|q_0| = 1$ då vore $|a| = |q_0 b + r| \leq |q_0 b| + |r| = |b| + |r| < 2|b|$, vilket också är en motsägelse mot vår antagande att $|a| \geq 2|b|$. Därför måste $|q_0| \geq 2$ gälla. □

Nu med vetskap om att $|q_0| \geq 2$ gäller kan vi fortsätta.
De återstående M-stegen följer MEUKL(b,r) och eftersom $|b| \geq 2|r|$ då gäller: $|b| \geq A_M$ och $|r| \geq A_{M-1}$ tack vare vårt påstående $P(M)$. Vidare får vi: $|a| = |q_0 b + r|$. Eftersom vi har visat att $|q_0| \geq 2$, kan vi säga att $|q_0 b| \geq 2|b|$. Därför är $|a| = |q_0 b + r| \geq 2|b| + |r| = 2A_M + A_{M-1} = A_{M+1}$ vilket är vad vi söker.

Vi kan nu visa att MEUKL(A_{N+1}, A_N) tar precis N steg genom att bryta ner processen i steg:

$$\begin{aligned} \text{Steg 1:} \quad & A_{N+1} = q_1 A_N + A_{N-1}, \\ \text{Steg 2:} \quad & A_N = q_2 A_{N-1} + A_{N-2}, \\ & \vdots \\ \text{Steg N:} \quad & A_3 = q_N A_2 + A_1. \end{aligned}$$

Eftersom vi startade på steg $N+1$ i talföljden, kommer vi att ha tagit N steg när vi når slutet av talföljden. Därför tar MEUKL(A_{N+1}, A_N) precis N steg, vilket är vad vi ville visa. □

När vi försöker att förstå tillväxten av vår modifierade Euklides talföljd A_n hittar vi en formel som liknar vår formel för Fibonaccisekvensen F_n. Vi använder oss av den linjära rekurrensrelationen, vilket är ett verktyg för att lösa problem som involverar sekvenser som definieras rekursivt [3]. Vi använder oss av vår talföljd A_n och hittar denna ekvation och använder oss sedan av metoden för karakteristiska ekvationer för att lösa den. Den karakteristiska ekvationen för denna rekurrensrelation är $x^2 - 2x - 1 = 0$, där lösningarna är $x = 1 - \sqrt{2}$ och $x = 1 + \sqrt{2}$.

Därmed kan vi skriva A_n som en kombination av potenser i följande sats:

Sats 2.15. *Det n:te talet i den modifierade Euklides talföljden, A_n, kan ges av formeln*

$$A_n = (2 - \sqrt{2})(1 - \sqrt{2})^n + (2 + \sqrt{2})(1 + \sqrt{2})^n,$$

där $(1+\sqrt{2})^n$ *är den dominerande termen som växer exponentiellt med* n*.*

Denna sats visar att A_n växer exponentiellt med n. Detta är anledningen till att antalet divisionssteg i den modifierade Euklides algoritmen är begränsat av en logaritmisk funktion av T.

Som en konsekvens av Sats 2.14 som säger att vår talföljd A_n ger oss det högsta antalet divisionssteg och med hjälp av vår Sats 2.15 där vi kan hitta vår dominerande term $(1+\sqrt{2})^n$, får vi även en viktig följdsats som lyder:

Följdsats 2.16. *Om* $0 \leq a, b < T$*, då är antalet divisionssteg som krävs när den modifierade Euklides algoritmen tillämpas på* a *och* b *högst* $\lceil \log_{1+\sqrt{2}} T \rceil$*.*

Bevis. Med Sats 2.14 ser vi att det högsta antalet divisionssteg, N, uppstår när $|a| = A_N$ och $|b| = A_{N-1}$, där N är så stort som möjligt med $A_N < T$.

Vi vill hitta det största N så att A_N är mindre än T. Vi vet att A_N växer exponentiellt med N, så vi kan använda logaritmer för att lösa detta. Vi får:

$$(1+\sqrt{2})^N < T$$

Vi tar logaritmen med bas $(1+\sqrt{2})$ på båda sidorna:

$$N < \log_{1+\sqrt{2}} T$$

Eftersom N måste vara ett heltal, får vi att N är högst $\lceil \log_{1+\sqrt{2}} T \rceil$. Detta är det vi ville visa. □

Alltså det följdsatsen säger är att det finns ett tak för de högst antal steg som den modifierade algoritmen tar att utföra som kan räknas ut med $\lceil \log_{1+\sqrt{2}} T \rceil$.

För att illustrera tillväxten av de olika talföljderna jämför vi A_n och F_{n+2} numeriskt i tabellen nedan, vilket ger oss en visuell representation av hur snabbt de olika talföljderna växer i relation till varandra.

Tabellen ger oss en tydlig bild av hur tillväxten av de olika talföljderna beter sig. Vi kan se att A_n växer betydligt snabbare än F_{n+2}.

2.5 Jämförelse mellan EUKL och MEUKL

I denna sektion kommer vi att jämföra den traditionella Euklides algoritmen (EUKL) med den modifierade Euklides algoritmen (MEUKL) och deras respektive talföljder. Dessa två algoritmer, trots att de är baserade på samma grundläggande princip, har olika egenskaper och beteenden beroende på de specifika krav de uppfyller. För att göra en rättvis jämförelse så kommer vi att

n	F_{n+2}	A_n
1	2	1
2	3	2
3	5	5
4	8	12
5	13	29
6	21	70
7	34	169
8	55	408
9	89	985
10	144	2378

Tabell 2: Jämförelse av tillväxten av F_{n+2} och A_n

anpassa EUKL så att den uppfyller samma krav som MEUKL, det vill säga $|a| \geq 2|b|$. Detta innebär att vi kommer att välja talpar från Fibonacci-sekvensen som uppfyller detta krav, snarare än att bara ta på varandra följande tal. Genom att göra detta kan vi undersöka hur de två algoritmerna presterar under samma förutsättningar och dra slutsatser om deras effektivitet och beteende. Vi kommer att illustrera detta med flera exempel och sedan bevisa att MEUKL och EUKL tar lika många steg för ett talpar från den modifierade Euklides talföljden A_n. Dessutom kommer vi att visa att MEUKL och EUKL tar exakt samma antal steg för talparen (A_N, A_{N-1}).
Låt oss utforska detta genom att titta på några exempel. Vi kommer först att överväga talpar som består av Fibonacci-tal, och därefter kommer vi att överväga talpar som kommer från vår modifierade Euklides talföljd:

$(377, 144)$**:**

EUKL*:*

$$377 = 2 \cdot 144 + 89$$
$$144 = 1 \cdot 89 + 55$$
$$89 = 1 \cdot 55 + 34$$
$$55 = 1 \cdot 34 + 21$$
$$34 = 1 \cdot 21 + 13$$
$$21 = 1 \cdot 13 + 8$$
$$13 = 1 \cdot 8 + 5$$
$$8 = 1 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$
$$2 = 2 \cdot 1$$

MEUKL*:*

$$377 = 3 \cdot 144 - 55$$
$$144 = (-3) \cdot (-55) - 21$$
$$-55 = 3 \cdot (-21) + 8$$
$$-21 = (-3) \cdot 8 + 3$$
$$8 = 3 \cdot 3 - 1$$
$$3 = (-3) \cdot (-1)$$

SGD(377,144) = 1

$(55, 12)$ **:**

EUKL*:*

$$55 = 4 \cdot 12 + 7$$
$$12 = 1 \cdot 7 + 5$$
$$7 = 1 \cdot 5 + 2$$
$$5 = 2 \cdot 2 + 1$$
$$2 = 2 \cdot 1$$

MEUKL*:*

$$55 = 5 \cdot 12 - 5$$
$$12 = (-2) \cdot (-5) + 2$$
$$-5 = (-2) \cdot 2 - 1$$
$$2 = (-2) \cdot (-1)$$

SGD(55,12) = 1

$(144, 55)$**:**

EUKL*:*

$$144 = 2 \cdot 55 + 34$$
$$55 = 1 \cdot 34 + 21$$
$$34 = 1 \cdot 21 + 13$$
$$21 = 1 \cdot 13 + 8$$
$$13 = 1 \cdot 8 + 5$$
$$8 = 1 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$
$$2 = 2 \cdot 1$$

MEUKL*:*

$$144 = 3 \cdot 55 - 21$$
$$55 = (-3) \cdot (-21) - 8$$
$$-21 = 3 \cdot (-8) + 3$$
$$-8 = (-3) \cdot 3 + 1$$
$$3 = 3 \cdot 1$$

$SGD(144{,}55) = 1$

Vi tar dessa exempel och jämför antal steg de tar att utföra i en tabell nedan:

Talpar	EUKL (steg)	MEUKL (steg)
(377,144)	11	6
(144,55)	9	5
(55,21)	5	4

Tabell 3: Jämförelse av antal steg för EUKL och MEUKL

I tabellen ovan ser vi att den modifierade algoritmen konsekvent tar färre steg än den traditionella algoritmen för samma talpar.

Nu ska vi utföra liknande jämförelser, men denna gång med tal från vår modifierade Euklides talföljd, A_n. Eftersom vi redan vet att denna talföljd uppfyller kraven för den modifierade talföljden, kan vi välja tal som följer direkt efter varandra. Låt oss ta några exempel och illustrera detta nedan:

$(408, 169)$**:**

EUKL*:*
$408 = 2 \cdot 169 + 70$
$169 = 2 \cdot 70 + 29$
$70 = 2 \cdot 29 + 12$
$29 = 2 \cdot 12 + 5$
$12 = 2 \cdot 5 + 2$
$5 = 2 \cdot 2 + 1$
$2 = 2 \cdot 1$

MEUKL*:*
$408 = 2 \cdot 169 + 70$
$169 = 2 \cdot 70 + 29$
$70 = 2 \cdot 29 + 12$
$29 = 2 \cdot 12 + 5$
$12 = 2 \cdot 5 + 2$
$5 = 2 \cdot 2 + 1$
$2 = 2 \cdot 1$

$\mathrm{SGD}(408, 169) = 1$

$(70, 29)$**:**

EUKL*:*
$70 = 2 \cdot 29 + 12$
$29 = 2 \cdot 12 + 5$
$12 = 2 \cdot 5 + 2$
$5 = 2 \cdot 2 + 1$
$2 = 2 \cdot 1$

MEUKL*:*
$70 = 2 \cdot 29 + 12$
$29 = 2 \cdot 12 + 5$
$12 = 2 \cdot 5 + 2$
$5 = 2 \cdot 2 + 1$
$2 = 2 \cdot 1$

$\mathrm{SGD}(70, 29) = 1$

Från våra exempel ser vi att den traditionella Euklides algoritmen tar N steg att utföra för talparen (A_N, A_{N-1}). Nu ska vi visa detta mer formellt genom att skriva ner det som en sats och bevisa den:

Sats 2.17. *För alla $N \geq 1$, tar den traditionella Euklides algoritmen N steg när den tillämpas på talparen (A_N, A_{N-1}), där A_N och A_{N-1} är på varandra följande tal i talföljden A_n.*

Vi definierar påståendet $P(N)$, för varje heltal $N \geq 1$ som lyder:
$P(N)$: $\mathrm{EUKL}(A_N, A_{N-1}) = N$.

Bevis av Sats 2.17. Vi visar att påståendet $P(N)$ gäller för varje $N \geq 1$
1) Vi börjar med basfallet $N = 1$. Påståendet $P(1)$ säger att $\mathrm{EUKL}(A_1, A_0) = 1$, vilket innebär att Euklides algoritm tar 1 steg när den tillämpas på talparen (A_1, A_0). Detta stämmer överens med vårt påstående, så $P(1)$ är sant.

2) Vi ska visa att om påståendet gäller för $N = M$, där M är något godtyckligt tal, då gäller det även för $N = M + 1$. Vi ska alltså visa:

$P(M+1)$: EUKL$(A_{M+1}, A_M) = M+1$.

Antag att påståendet $P(M)$ gäller, det vill säga att EUKL$(A_M, A_{M-1}) = M$. Detta innebär att vi har utfört division med rest-operationen M gånger för att få resten noll när vi tillämpar EUKL på talparen (A_M, A_{M-1}).

Om vi nu betraktar talparen (A_{M+1}, A_M), kommer vi att utföra samma operation en gång till eftersom $A_{M+1} > A_M$.

Varje steg i processen representerar en iteration av Euklides algoritm, där vi använder definitionen av talföljden A_n för att bryta ner talparen. Vi kan bryta ner processen i följande steg:

$$\begin{aligned}
\text{Steg 1:} \quad & A_{M+1} = 2A_M + A_{M-1}, \\
\text{Steg 2:} \quad & A_M = 2A_{M-1} + A_{M-2}, \\
& \vdots \\
\text{Steg M:} \quad & A_3 = 2A_2 + A_1, \\
\text{Steg M+1:} \quad & A_2 = 2A_1 + A_0.
\end{aligned}$$

Vi fortsätter processen tills vi når slutet av talföljden. Eftersom vi startade på steg $M+1$ i talföljden, kommer vi att ha tagit $M+1$ steg när vi når slutet av talföljden.

Därför tar EUKL(A_{M+1}, A_M) precis $M+1$ steg, vilket är vad vi ville visa. □

Med utgångspunkt från Sats 2.17, där vi visade att den traditionella Euklides algoritmen tar N steg för talparen (A_N, A_{N-1}), kan vi nu dra en viktig slutsats om effektiviteten hos den modifierade Euklides algoritmen jämfört med den traditionella. Vi gör det i följdsatsen nedan:

Följdsats 2.18. *För alla $N \geq 1$, tar både den traditionella Euklides algoritmen och den modifierade Euklides algoritmen exakt N steg när de tillämpas på talparen (A_N, A_{N-1}), där A_N och A_{N-1} är på varandra följande tal i talföljden A_n.*

Bevis. Från Sats 2.17 vet vi att den traditionella Euklides algoritmen tar exakt N steg när den tillämpas på talparen (A_N, A_{N-1}). Vi vet också att den modifierade Euklides algoritmen tar exakt N steg när den tillämpas på talparen (A_N, A_{N-1}), enligt Sats 2.14 av talföljden A_n.

Därmed har vi visat att båda algoritmerna, den traditionella och den modifierade Euklides algoritmen, tar exakt N steg när de tillämpas på talparen (A_N, A_{N-1}). Detta fullbordar beviset.

□

3 Euklides algoritm, primtalsfaktorisering och polynomringar

3.1 Aritmetikens fundamentalsats och primtalsfaktorisering

Avslutningsvis i detta arbete kommer vi i detta avsnitt att utforska djupare betydelser av Euklides algoritm inom talteorin. Utöver att vara en effektiv metod för att hitta den största gemensamma delaren av två tal, spelar Euklides algoritm en central roll i flera fundamentala koncept, såsom aritmetikens fundamentalsats och primtalsfaktorisering. Vi kommer även att se hur dessa idéer kan generaliseras till polynomringar.

En grundläggande tillämpning av Euklides algoritm är i beviset för Bézouts identitet, en viktig sats inom talteorin. Denna identitet är central för lösningen av diofantiska ekvationer och är en nödvändig komponent i beviset för aritmetikens fundamentalsats.

Sats 3.1 (Bézouts identitet)**.** *Låt a och b vara heltal, inte båda noll. Då finns det heltal x och y sådana att $ax + by = sgd(a, b)$.*

Bevis. Euklides algoritm kan användas för att hitta den största gemensamma delaren av två tal. Genom att lösa varje steg i algoritmen för resten, kan vi uttrycka varje rest som en linjär kombination av a och b. När vi når slutet av algoritmen, kan vi uttrycka $sgd(a, b)$ som en linjär kombination av a och b [4]. □

Efter att ha bevisat Bézouts identitet, introducerar vi Euklides lemma. Detta lemma spelar en central roll i beviset för aritmetikens fundamentalsats. Men först, låt oss definiera några viktiga begrepp.

Definition 3.2 (Primtal)**.** *Ett heltal $p > 1$ är ett primtal om dess enda positiva delare är 1 och p själv.*

Lemma 3.3 (Euklides lemma)**.** *Låt p vara ett primtal och a och b vara heltal. Om p delar produkten ab, då måste p dela a eller b.*

Bevis. Vi ska visa att om p delar produkten ab, då måste p dela antingen a eller b. Vi överväger två fall:

Fall 1: p delar a. I detta fall är vi klara eftersom vi har visat att p delar antingen a eller b.

Fall 2: p delar inte a. I detta fall innebär det att p och a inte har några gemensamma faktorer förutom 1, så $sgd(a, p) = 1$.

Enligt Bézouts identitet, om $sgd(a, p) = 1$, då finns det heltal x och y sådana att $1 = ax + py$. Detta betyder att vi kan skriva 1 som en linjär kombination av a och p.

Om vi multiplicerar denna ekvation med b, får vi $b = abx + pby$. Eftersom p delar både termen abx (eftersom den innehåller ab som en faktor) och termen pby (eftersom den innehåller p som en faktor), måste p också dela b.
Så, om p delar produkten ab men inte delar a, då måste p dela b. Detta visar att om p delar produkten ab, då måste p dela a eller b, vilket avslutar beviset [4]. □

Följdsats 3.4 (Euklides lemma för produkter)**.** *Låt p vara ett primtal och $a_1, a_2, \ldots, a_n$ vara heltal. Om p delar produkten $a_1 a_2 \cdots a_n$, då måste p dela minst ett av talen a_i.*

Med Bézouts identitet och Euklides lemma, är vi nu redo att ta oss an aritmetikens fundamentalsats. Denna sats är grundläggande för vår förståelse av strukturen hos heltalen och spelar en central roll i primtalsfaktorisering.

Sats 3.5 (Aritmetikens fundamentalsats)**.** *Varje heltal $n \geq 2$ kan primtalsfaktoriseras på ett och endast ett sätt, upp till ordningen av faktorerna.*

Bevis. Antag att det finns ett tal n som kan skrivas som en produkt av primtal på två olika sätt. Vi skriver dessa två sätt som:

$$p_1 p_2 \cdots p_r = n = q_1 q_2 \cdots q_s.$$

Vi antar att n är det minsta talet med två olika primtalsfaktoriseringar. Eftersom primtalet p_1 delar vänster sida, måste det också dela höger sida. Enligt Följdsats 3.4 betyder det att p_1 måste dela minst ett av talen $q_1, q_2, \ldots, q_s$. Vi kan anta att p_1 delar q_1. Eftersom både p_1 och q_1 är primtal, måste de vara samma tal, alltså $p_1 = q_1$.
Om vi delar båda sidor med p_1 (eller q_1, eftersom de är samma), får vi:

$$p_2 \cdots p_r = q_2 \cdots q_s.$$

Vi fortsätter på detta sätt, med att stryka lika faktorer från båda sidor, tills det inte går längre. Om det finns några p-tal kvar eller q-tal kvar, så kommer vi att få en produkt av primtal på ena sidan och 1 på den andra sidan. Men 1 kan inte skrivas som en produkt av primtal, så detta skulle strida mot vårt antagande att n var det minsta talet med två olika primtalsfaktoriseringar. Därför måste alla tal kunna skrivas som en produkt av primtal på bara ett sätt.

Slutligen, för att visa att varje tal $n \geq 2$ kan primtalsfaktoriseras, antar vi motsatsen. Låt n vara det minsta talet som inte kan primtalsfaktoriseras. Eftersom n inte är ett primtal (eftersom det inte kan primtalsfaktoriseras), kan n skrivas som $n = ab$ där $2 \leq a, b < n$. Eftersom a och b är mindre än n, och n var det minsta talet som inte kunde primtalsfaktoriseras, måste både a och b kunna primtalsfaktoriseras. Men då kan också $n = ab$ primtalsfaktoriseras, vilket är en motsägelse. Därför måste varje tal $n \geq 2$ kunna primtalsfaktoriseras. □

3.2 Primtalsfaktorisering i polynomringar och Euklides algoritm

I detta avsnitt kommer vi att utforska hur idéer från talteorin, såsom primtalsfaktorisering, kan generaliseras till polynomringar. Denna generalisering är inte bara en intressant matematisk övning, utan den har också viktiga tillämpningar inom områden som kryptografi och kodningsteori. För att göra detta behöver vi först definiera några grundläggande koncept inom abstrakt algebra: ringar och kroppar.

En *ring* är en mängd element där vi kan utföra både addition och multiplikation på ett sätt som liknar hur vi utför dessa operationer på heltal. En *kropp* är en speciell typ av ring där varje element (förutom noll) har ett multiplikativt invers, vilket innebär att vi kan utföra division. Exempel på kroppar inkluderar de rationella talen, de reella talen och de komplexa talen, men i detta avsnitt kommer vi att vara mest intresserade av kroppar av polynom.

Nu ska vi formellt definiera dessa koncept och utforska några av deras viktiga egenskaper. Vi kommer också att se hur dessa koncept tillåter oss att generalisera idéer från talteorin, såsom Euklides algoritm och primtalsfaktorisering, till polynomringar.

Definition 3.6 (Ring)**.** *En ring är en mängd R tillsammans med två operationer, addition $(+)$ och multiplikation $(\cdot)$, som uppfyller följande axiomer:*

1. *(Additiv associativitet) För alla $a, b, c \in R$, gäller att $(a+b)+c = a+(b+c)$.*
2. *(Additivt identitetselement) Det finns ett element $0 \in R$ sådant att för alla $a \in R$, gäller att $a + 0 = a$ och $0 + a = a$.*
3. *(Additivt invers) För varje $a \in R$, finns det ett element $-a \in R$ sådant att $a + (-a) = 0$ och $(-a) + a = 0$.*
4. *(Additiv kommutativitet) För alla $a, b \in R$, gäller att $a + b = b + a$.*
5. *(Multiplikativ associativitet) För alla $a, b, c \in R$, gäller att $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
6. *(Multiplikativt identitetselement) Det finns ett element $1 \in R$ sådant att $x \cdot 1 = 1 \cdot x = x$ för alla $x \in R$.*
7. *(Distributivitet) För alla $a, b, c \in R$, gäller att $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ och $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$ [5].*

Definition 3.7 (Kropp)**.** *Låt K vara en mängd. Vi säger att K är en kropp om den är en ring och för varje element $a \in K$ (förutom noll), finns det ett annat element $b \in K$ så att $a \cdot b = 1$. Detta andra element b kallas det multiplikativa inverset till a [5].*

Med dessa definitioner på plats, kan vi nu börja utforska hur vi kan generalisera idéer från talteorin till polynomringar. Precis som vi kan dela heltal och hitta deras största gemensamma delare, kan vi också dela polynom och hitta deras största gemensamma delare. Detta leder oss till en generalisering av Euklides algoritm, som vi kan använda för att hitta den största gemensamma delaren av två polynom.
Dessutom, precis som vi kan dela upp heltal i primtal, kan vi dela upp polynom i vad vi kallar irreducibla polynom, vilka spelar en roll i polynomringar som är analog med rollen som primtal i heltalen.

Innan vi går vidare, låt oss klargöra några termer. Vi använder K för att representera en kropp, vilket är en samling av element där vi kan lägga till, dra ifrån, multiplicera och dela (så länge vi inte delar med noll). Du kan tänka på K som alla reella tal eller alla komplexa tal, men det finns också andra exempel.

Definition 3.8. *Med $K[x]$ menar vi ringen av alla polynom vars koefficienter kommer från K. Ett polynom i $K[x]$ kan se ut som $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, där varje a_i är ett element i K.*

I det följande kommer vi att presentera en serie satser och definitioner som är analoger till de vi tidigare har sett för heltal, men nu för polynom. Detta kommer att visa oss hur djupt dessa koncept är kopplade, och hur idéer från talteorin kan generaliseras till andra matematiska strukturer.

Sats 3.9. *Om två polynom $f(x), g(x) \in K[x]$ är givna, där $\deg(f) \geq \deg(g)$ och $g \neq 0$, så kan man skriva $f(x) = q(x)g(x) + r(x)$, där $q(x), r(x) \in K[x]$ och antingen $r(x) = 0$ eller $\deg(r) < \deg(g)$. Vi kallar $q(x)$ för kvoten och $r(x)$ för resten.*

Definition 3.10. *Ett polynom $d(x)$ sägs dela ett annat polynom $f(x)$ om det finns ett polynom $q(x)$ sådant att $f(x) = d(x)q(x)$.*

Definition 3.11. *Den största gemensamma delaren till två polynom $f(x)$ och $g(x)$, betecknad $d(x)$, är det moniska polynom av högsta möjliga grad som delar både $f(x)$ och $g(x)$. Ett polynom är moniskt om koefficienten av dess högsta gradsterm är 1.*

Sats 3.12. *En gemensam delare till $f(x)$ och $g(x)$ måste även dela $r(x) = f(x) - q(x)g(x)$. Dessutom, om $d(x)$ är den största gemensamma delaren till $f(x)$ och $g(x)$, då är $d(x)$ också den största gemensamma delaren till $g(x)$ och $r(x)$.*

Bevis. Låt oss anta att $d(x)$ är den största gemensamma delaren till $f(x)$ och $g(x)$. Eftersom $d(x)$ delar både $f(x)$ och $g(x)$, måste den också dela $r(x) = f(x) - q(x)g(x)$.
För att visa att $d(x)$ är den största gemensamma delaren till $g(x)$ och $r(x)$, observera att varje gemensam delare till $g(x)$ och $r(x)$ också måste dela $f(x) = q(x)g(x) + r(x)$. Därför kan ingen gemensam delare till $g(x)$ och $r(x)$ vara större

än $d(x)$, vilket innebär att $d(x)$ är den största gemensamma delaren till $g(x)$ och $r(x)$.
Detta bevis följer samma logik som beviset för Sats 1.5, som visar att den sista nollskilda resten i Euklides algoritm på talparet (a, b) är lika med $\text{sgd}(a, b)$. Vi hänvisar till det beviset för mer detaljer. □

När vi arbetar med polynom, är det ofta användbart att bryta ner dem i mindre delar, på samma sätt som vi kan bryta ner heltal i primfaktorer. Men vissa polynom kan inte brytas ner på detta sätt - de är 'grundläggande' på ett sätt som är analogt med primtal. Dessa polynom kallas irreducibla, och de spelar en central roll i många områden av algebra, inklusive polynomfaktorisering och lösning av polynomekvationer. Låt oss definiera detta mer formellt:

Definition 3.13 (Irreducibla polynom)**.** *Ett polynom $f(x) \in K[x]$ är irreducibelt över kroppen K om det inte är möjligt att skriva det som en produkt av två polynom i $K[x]$ vars grader är mindre än graden av $f(x)$.*

Vi har tidigare diskuterat konceptet delbarhet i samband med heltal. Precis som vi kan säga att ett heltal delar ett annat om det multipliceras med ett tredje heltal för att få det andra, kan vi säga att ett polynom delar ett annat om det kan multipliceras med ett tredje polynom för att få det andra. Denna idé om delbarhet sträcker sig naturligt till polynom och är lika grundläggande inom algebra. Låt oss nu formalisera detta koncept i kontexten av polynom:

Definition 3.14 (Delbarhet i polynomringar)**.** *Låt $f(x), g(x) \in K[x]$ vara två polynom. Vi säger att $f(x)$ delar $g(x)$, skrivet $f(x)|g(x)$, om det finns ett polynom $h(x) \in K[x]$ sådant att $g(x) = f(x)h(x)$.*

Med dessa definitioner och satser i åtanke, kan vi nu diskutera mer komplexa koncept som Euklides algoritm för polynom och hur den kan användas för att hitta den största gemensamma delaren av två polynom.

Euklides algoritm för polynom fungerar på ett liknande sätt som den gör för heltal. Givet två polynom, $f(x)$ och $g(x)$, där vi vill hitta deras största gemensamma delare, börjar vi med att dela det större polynomet (i termer av grad) med det mindre polynomet, och tar sedan resten. Vi upprepar denna process, varje gång vi byter ut det större polynomet med det mindre polynomet, och det mindre polynomet med resten, tills vi får en rest av noll. Vid denna punkt är det sista icke-noll polynomet den största gemensamma delaren av $f(x)$ och $g(x)$.

Definition 3.15 (Euklides algoritm)**.** *Euklides algoritmen är en process för att hitta den största gemensamma delaren $d(x)$ mellan två polynom $f(x), g(x) \in K[x]$ där $\deg(f) \geq \deg(g)$ och $g \neq 0$. Algoritmen består av följande steg:*

1. *Skriv $f(x) = q(x)g(x) + r(x)$ där $q(x), r(x) \in K[x]$ och antingen $r(x) = 0$ eller $\deg(r) < \deg(g)$.*

2. *Om $r(x) = 0$, avsluta processen. Polynomet $g(x)$ är då den största gemensamma delaren $d(x)$. Annars, gå tillbaka till steg 1 med (g, r) istället för (f, g).*

Låt oss nu se ett exempel på hur Euklides algoritm kan användas för att hitta den största gemensamma delaren av två polynom. Vi kommer att använda polynomen $f(x) = x^3 - 2x^2 - x + 2$ och $g(x) = x^2 - x - 1$ över kroppen av reella tal $\mathbb{R}$.

1. Vi börjar med att dela $f(x)$ med $g(x)$, vilket ger oss kvoten $q_1(x) = x - 1$ och resten $r_1(x) = -x + 1$.

2. Vi byter nu ut $f(x)$ med $g(x)$ och $g(x)$ med $r_1(x)$, och upprepar processen. Vi delar $g(x)$ med $r_1(x)$, vilket ger oss kvoten $q_2(x) = x + 1$ och resten $r_2(x) = -1$.

3. Vi byter nu ut $g(x)$ med $r_1(x)$ och $r_1(x)$ med $r_2(x)$, och upprepar processen. Vi delar $r_1(x)$ med $r_2(x)$, vilket ger oss kvoten $q_3(x) = -x + 1$ och resten $r_3(x) = 0$.

4. Vid denna punkt har vi fått en rest av noll, så vi avslutar processen. Det sista icke-noll polynomet, $r_2(x) = -1$, är en största gemensamma delare av $f(x)$ och $g(x)$. Om vi vill uttrycka den största gemensamma delaren som ett moniskt polynom, blir den största gemensamma delaren 1.

Observera att den största gemensamma delaren är ett konstant polynom, vilket innebär att $f(x)$ och $g(x)$ är relativt prima, dvs. de har ingen gemensam faktor förutom konstanta faktorer.

3.3 Bézouts identitet, Gauss lemma och fundamentalsatsen för polynom

I detta avsnitt utforskar vi några av de mest grundläggande och kraftfulla resultaten inom algebra: Bézouts identitet för polynom, Gauss lemma för polynom och fundamentalsatsen för polynom. Dessa resultat, som vi tidigare har diskuterat i kontexten av heltal, tar en ny betydelse när de tillämpas på polynom. De illustrerar hur djupt koncept som delbarhet, primtal och unik faktorisering sträcker sig in i matematikens struktur och ger oss viktiga verktyg för att arbeta med och förstå polynom. Genom att undersöka dessa satser i kontexten av polynom, kan vi se hur dessa grundläggande koncept sträcker sig över olika områden av matematiken och skapar djupa och oväntade kopplingar.

Vi börjar med Bézouts identitet för polynom. Precis som Bézouts identitet för heltal ger oss ett sätt att uttrycka den största gemensamma delaren av två heltal som en linjär kombination av de två talen, ger Bézouts identitet för polynom oss ett sätt att uttrycka den största gemensamma delaren av två polynom som en

polynomkombinationäv de två polynomen. Detta är ett kraftfullt verktyg med många tillämpningar inom algebra.

Sats 3.16 (Bézouts identitet för polynom)**.** *Låt $f(x)$ och $g(x)$ vara polynom i $K[x]$, inte båda noll. Då finns det polynom $u(x)$ och $v(x)$ i $K[x]$ sådana att*

$$f(x)u(x) + g(x)v(x) = gcd(f(x), g(x)).$$

Beviset fungerar på exakt samma sätt som beviset av Bézouts identitet för heltal (Sats 3.1), med Euklides algoritm applicerad på polynom istället för heltal.

Vi fortsätter med Gauss lemma, som är en analog till Euklides lemma för heltal. Det ger oss ett viktigt kriterium för delbarhet i polynomringar. Specifikt säger det att om ett irreducibelt polynom delar en produkt av två polynom, då måste det dela minst ett av polynomen. Detta är en grundläggande egenskap som irreducibla polynom delar med primtal, och det är avgörande för att bevisa fundamentalsatsen för polynom.

Lemma 3.17 (Gauss lemma)**.** *Låt $p(x)$ vara ett irreducibelt polynom i $K[x]$ och $f(x)$ och $g(x)$ vara polynom i $K[x]$. Om $p(x)$ delar produkten $f(x)g(x)$, då måste $p(x)$ dela $f(x)$ eller $g(x)$ [6]*

Beviset fungerar på exakt samma sätt som beviset av Euklides lemma för heltal (Lemma 3.3), med den skillnaden att vi nu använder irreducibla polynom istället för primtal.

Slutligen kommer vi till fundamentalsatsen för polynom. Precis som fundamentalsatsen för heltal säger att varje heltal kan faktoriseras unikt som en produkt av primtal, säger fundamentalsatsen för polynom att varje icke-konstant polynom kan faktoriseras unikt som en produkt av irreducibla polynom. Denna sats är en av de mest grundläggande resultaten inom algebra, och den visar hur djupt konceptet med primtalsfaktorisering går in i matematikens struktur.

Sats 3.18 (Fundamentalsatsen för polynom)**.** *Varje icke-konstant polynom $f(x) \in K[x]$ kan faktoriseras som en produkt av irreducibla polynom i $K[x]$ på ett och endast ett sätt, upp till ordningen av faktorerna och multiplikation med icke-noll skalärer.*

Beviset fungerar på exakt samma sätt som beviset av fundamentalsatsen för heltal (Sats 3.5), med den skillnaden att vi nu använder irreducibla polynom istället för primtal.

Genom att utforska dessa satser och lemma, har vi sett hur djupt koncepten delbarhet, primtal och unik faktorisering sträcker sig in i matematikens struktur, och hur de ger oss kraftfulla verktyg för att arbeta med och förstå polynom. Dessa resultat är grundläggande för många områden inom algebra och belyser

den rika strukturen och de djupa sambanden inom matematiken. Speciellt har vi sett hur Euklides algoritm, ett grundläggande verktyg inom talteorin, har en direkt analog inom polynom, vilket ytterligare understryker de djupa kopplingarna mellan dessa områden av matematiken.

Tack

Jag skulle vilja uttrycka mitt djupaste tack till min handledare, Olof, för hans outtröttliga tålamod och vägledning under hela processen att skriva denna uppsats. Hans insikter och råd har varit ovärderliga, och hans entusiasm för ämnet har varit en ständig källa till inspiration. Jag är tacksam för den tid han har tagit att granska mitt arbete, ge feedback och hjälpa mig att förstå de komplexa koncepten inom algebra. Utan hans stöd skulle detta arbete inte ha varit möjligt.

Referenser

[1] Wikipedia. Euclidean algorithm — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Euclidean%20algorithm&oldid=1136359173, 2023. [Online; accessed 10-February-2023].

[2] Donald E. Knuth. *The art of computer programming*. Addison-Wesley Publishing Company, 2nd edition, 1981.

[3] Brilliant. Linear Recurrence Relations — brilliant - wiki. https://brilliant.org/wiki/linear-recurrence-relations/, 2023. [Online; accessed 9-June-2023].

[4] Qimh Xantcha och Håkan Granath Rikard Bøgvad. *Algebra I*. Stockholms Universitet, 10th edition, 2018. Linjär algebra.

[5] Norman L. Biggs. *Discrete mathematics*. Oxford University Press, 2nd edition, 1989.

[6] Wikipedia. Gauss's Lemma (polynomials) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Gauss%27s_lemma_(polynomials), 2023. [Online; accessed 25-July-2023].

1. **Satsens Beskrivning:** Uppdaterade satsen för att reflektera valet av $\tau(a)$ istället för tidigare formeln $Y(n)$ och förtydligade villkoren för hur a och b väljs.

2. **Genomsnittsformelns Beskrivning:** Förklarade mitt val av $\tau(a)$ som en mer skräddarsydd metod för att uppskatta det genomsnittliga antalet steg när a är fast och b är inbördes prima med a, jämfört med andra tillgängliga formler.

3. **Tabellens Beräkningar:** Ändrade vissa beräkningar i tabellen för att korrigera tidigare fel och för att anpassa dem till $\tau(a)$. Trots dessa ändringar kvarstår slutsatsen att det faktiska genomsnittliga antalet steg är något lägre än när man använder genomsnittsformeln.

1

2

2.1 Genomsnittet av antal steg

Det är klart att antalet steg som krävs för att utföra algoritmen varierar beroende på de specifika talen vi använder. Men kan vi hitta ett genomsnittligt antal steg för alla talpar (a, b) där $a > b > 0$? Efter lite forskning hittar vi att det finns tre olika formler för att beräkna detta genomsnitt.

- $T(a)$: Ger genomsnittet av stegen som krävs för att beräkna SGD av ett fast tal a och alla tal b mindre än a.

- $\tau(a)$: Tar genomsnittet av stegen som krävs för att beräkna SGD av a och alla tal b som är inbördes prima med a. Denna formel är en förfining av $T(a)$ och ger en mer konsekvent uppskattning genom att undvika talpar där SGD är större än 1.

- $Y(n)$: Ger genomsnittet av stegen som krävs när båda a och b väljs slumpmässigt från intervallet 1 till n. Denna formel bygger på de tidigare två och ger en övergripande bild av genomsnittet.

För att undvika förvirring mellan dessa formler, kommer jag i denna text att fokusera på en av formlerna: $\tau(a)$, som jag anser passar bäst med de exempel jag kommer att presentera senare. Jag valde denna formel eftersom den ger en mer skräddarsydd uppskattning av det förväntade genomsnittliga antalet steg när a är fast och b är inbördes prima med a.

Sats 2.6. *Låt a vara ett naturligt tal och b vara ett tal som är inbördes prima med a. Då är det förväntade antalet steg för Euklides algoritm att slutföra beräknat med följande formel:*

$$\tau(a) = \frac{1}{\phi(a)} \sum_{1 \leq b < a, \gcd(a,b)=1} T(a,b)$$

*Ett detaljerat bevis för denna sats finns i Donald E. Knuths verk "The Art of Computer Programming", sida **[354]**.*

I denna sektion kommer vi att utforska hur genomsnittsformeln $\tau(a)$ appliceras i praktiken. Vi kommer att utföra Euklides algoritm på tio slumpmässigt valda talpar (a, b) där $a > b > 0$ och $a, b < 100$. För varje par kommer vi att räkna ut det faktiska antalet steg som algoritmen tar och jämföra detta med det förväntade antalet steg enligt genomsnittsformeln. Resultaten presenteras i Tabell 1 nedan.

Table 1: Jämförelse mellan faktiskt och förväntat antal steg med genomsnittsformeln för Euklides algoritm.

Talpar (a, b)	Faktiskt antal steg	Förväntat antal steg
(97, 86)	5	4.31
(98, 48)	2	4.64
(80, 20)	1	4.13
(90, 37)	4	4
(96, 49)	4	4.25
(69, 38)	5	3.95
(85, 14)	2	4.13
(85, 17)	1	4.13
(87, 34)	6	4.21
(43, 32)	4	3.69
Genomsnitt	3.4	4.14

Genom att analysera resultaten i Tabell 1 kan vi se att det faktiska genomsnittliga antalet steg över de tio testfallen är nära det förväntade genomsnittet enligt genomsnittsformeln $\tau(a)$ men att det faktiska antalet steg är något lägre.