SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Permutationsgrupper

av

Marcus Ibrahim

2025 - No L13

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET, 106 91 STOCKHOLM

Permutationsgrupper

Marcus Ibrahim

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Håkan Granath

2025

Sammanfattning

Syftet med detta arbete är att ge en grundlig introduktion till permutationsgrupper, som är centralt inom gruppteori, genom att successivt bygga upp teorin för grupper och permutationer med begrepp som gruppverkan, banor, stabilisatorer, block och primitivitet. Arbetet kombinerar teori med praktiska algoritmer som implementeras i Python som läsaren har tillgång till. Det möjliggör interaktivt lärande och utforskandet av ämnet.

Abstract

The purpose of this project is to provide a thorough introduction to permutation groups, which are central to group theory, by gradually building up the theory of groups and permutations using concepts such as group action, orbits, stabilizers, blocks, and primitivity. The project combines theory with practical algorithms implemented in Python, which the reader has access to. This enables interactive learning and further exploration of the subject.

Innehåll

1 Inledning

Syftet med detta arbete är att ge en grundläggande genomgång av permutationsgrupper, vilket är centralt inom gruppteori. För detta introducerar vi successivt teori som permutationsgrupper bygger på. Meningen är att arbetet ska vara pedagogiskt och lättförståeligt för någon med grundläggande kunskaper inom gruppteori. Detta ska uppnås genom en kombination av teori och praktiska algoritmer som läsaren kan använda för att få en bättre förståelse. Med den givna koden kan användaren konstruera permutationer, se egenskaper och göra analyser som direkt är kopplade till teorin. Målet är att ge en tydlig presentation och ett eget utforskande av ämnet.

Arbetet inleds med grundläggande begrepp inom gruppteori, där vi kommer gå igenom vad en grupp är och tillhörande definitioner, satser och exempel som rör området. Detta kommer hjälpa oss att bygga på teorin för att gå vidare.

Här går vi vidare med begreppet gruppverkan, som beskriver hur en grupp kan operera på en mängd. Med det kommer vi gå in på banor, stabilisatorer, block och primitivitet. Detta är viktigt för att ge en förståelse om hur strukturer inom grupper kan se ut.

Vi går vidare till permutationer, där vi diskuterar definitioner, sätt representera permutationer, cykelstrukturer och ordning. I avsnittet kommer det visas hur permutationer uppfyller samma villkor som grupper under sammansättningar vilket gör att mängden av alla permutationer, symmetrigruppen S_n, är en grupp i sig.

Det föregående avsnittet leder oss naturligt in i nästa, permutationsgrupper, som är delgrupper av symmetrigruppen och är centralt för gruppteori. Här drar vi kopplingar till tidigare teori om grupper och gruppverkan men i en kontext av permutationer för att verkligen visa hur de begrepp vi tidigare gått igenom fungerar i sammanhang där elementen är permutationen. Detta gör vi genom att se hur permutationer fungerar under sammansättning och se på olika egenskaper som abelskhet, normalitet och primitivitet samt fördjupa oss i strukturer som finns.

När vi har grunden till all teori går vidare till algoritmer som går hand i hand med teorin och som visar hur aspekterna av teorin implementeras praktiskt. Det kommer att visas hur man genererar en delgrupp och hittar block med bevis grundat i teorin.

Till slut visar vi hur koden kan implementeras, där algoritmerna översatts till Python. Med koden kommer användaren själv kunna skapa permutationer, undersöka, experimentera och göra analyser på permutationsgrupper och deras egenskaper. Vi kommer gå vidare till att visa enklare tillämpningar som successivt leder till mer avancerade analyser. Avslutningsvis kommer läsaren kunna ta del av koden

och ladda ner den för att själv fortsätta utforska vidare med arbetet som verktyg.

2 Grupper

Begreppet grupper är grunden för mycket av teorin som kommer behandlas i arbetet. En grupp är en algebraisk struktur och vi kommer formellt beskriva vad en grupp är och introducera grundläggande definitioner, exempel och satser.

2.1 Introduktion

En grupp är en struktur där en mängd med en operation uppfyller vissa villkor. Här kommer vi gå igenom vad en grupp är, definitioner, exempel och satser som är grunden till idén om grupper.

Definition 2.1. [2] En grupp består av en mängd G, som tillsammans med en binär operator $*$ definierad för G ska uppfylla följande axiom:

1. (**Sluten**). För alla $x, y \in G$ är

$$x * y \in G.$$

2. (**Associativ**). För alla $x, y, z \in G$ är

$$(x * y) * z = x * (y * z).$$

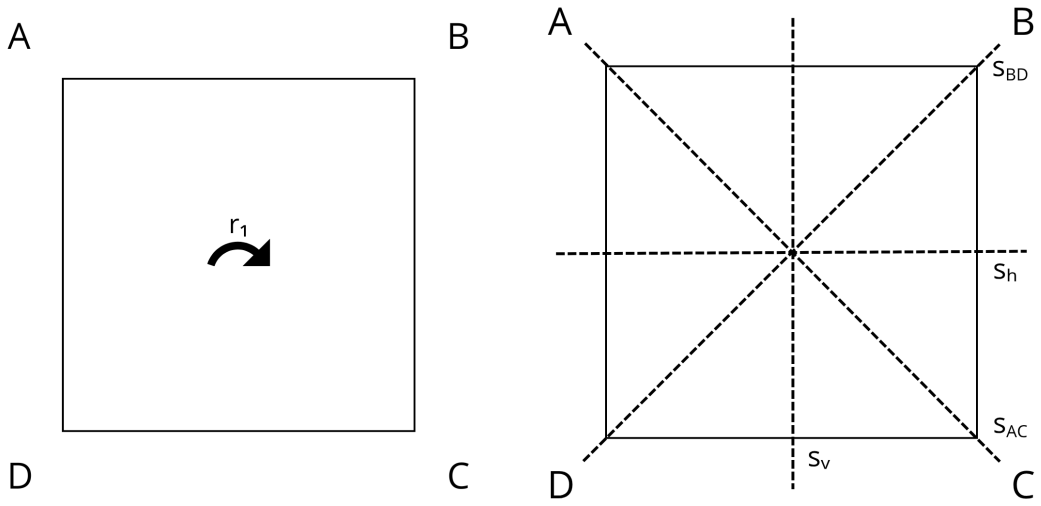
3. (**Identitetselement**). Det finns ett element $e \in G$ sådan att

$$e * x = x = x * e \text{ för alla } x \in G.$$

4. (**Inverselement**). För alla $x \in G$ finns det x^{-1} sådan att

$$x * x^{-1} = e = x^{-1} * x.$$

Exempel 2.2. [2] Ett exempel på en grupp är symmetrierna av en kvadrat D_4. Vi kan tänka på den som ett platt kort, med hörnen märkta A, B, C, D i medsols ordning. Det finns åtta olika transformationer av D_4 som behåller formen av figuren. Dessa transformationer kallas symmetrier. Symmetrierna består av den triviala symmetrin e (identitetselementet), rotationer och speglingar. Rotationerna görs medurs och betecknas $r_1(90°)$, $r_2(180°)$ och $r_3(270°)$. Speglingar över axlar betecknas s_v (vertikal), s_h (horisontell), s_{AC} (diagonal från A till C) och s_{BD} (diagonal från B till D).



Definition 2.3. [2] En delmängd H av en grupp G kallas en delgrupp om elementen i H uppfyller egenskaper för en grupp med samma binära operator som G.

Exempel 2.4. Om vi har gruppen D_4, symmetrier av en kvadrat, från exempel 2.2 så är rotationerna

$$H = \{e, r_1, r_2, r_3\}$$

en delgrupp av gruppen.

För ett element i gruppen $x \in G$ med binär operator $*$, är potensen x^k en upprepning av operatorn $k \in \mathbb{Z}$ antal gånger:

$$x^k = \underbrace{x * x * \cdots * x}_{k \text{ gånger}}, \text{ för } k > 0$$

med $x^0 = id$. För negativa exponenter $k < 0$ har vi:

$$x^k = (x^{-1})^{-k}.$$

Vi utför alltså upprepade operationer med elementets invers.

Definition 2.5. [7] Antalet element i en viss mängd M kallas för mängdens kardinalitet och vi betecknar det $|M|$.

Exempel 2.6. Om vi har mängden $\{1, 2, 3\}$ är dess kardinalitet 3. Vi skriver $|\{1, 2, 3\}| = 3$.

Definition 2.7. [2] Om G är en grupp och $|G|$ är ändligt, så är $|G|$ ordningen av gruppen. En grupp med oändligt antal element har oändlig ordning.

Exempel 2.8. Gruppen $(\mathbb{Z}_4, +)$ består av heltalen $\{0, 1, 2, 3\}$ med addition modulo 4 som binär operator. Gruppens kardinalitet $|G| = 4$ och har därför ordning 4.

Exempel 2.9. Gruppen $(\mathbb{Z}, +)$ består av alla heltal med addition som binär operator. Eftersom gruppen består av oändligt många element så har gruppen oändlig ordning.

Definition 2.10. [2] Om x är ett element i en en ändlig grupp G, så är det minsta positiva heltalet m, sådan att $x^m = e$, lika med ordningen av x i G. Om G är en oändlig grupp, definieras ordningen av x på samma sätt, förutsatt att ett heltal m finns, annars är x ett element med oändlig ordning.

Eftersom G är en grupp så är x^m ett element i gruppen för varje m. Om G även är en ändlig grupp så måste vissa potenser vara lika eftersom vi har ändligt många element men oändligt många exponenter. Låt oss säga att vi har element i gruppen, $x^a = x^b$, där $a > b$. Om vi multiplicerar båda sidorna med x^{-b} så får vi $x^{a-b} = e$. Så vi kan säga att det finns ett m sådan att $x^m = e$.

Exempel 2.11. Vi har gruppen $(\mathbb{Z}_6, +)$ som består av heltalen $\{0, 1, 2, 3, 4, 5\}$ med addition modulo 6 som operator och undersöker vilken ordning 2 har. Vi vill kolla hur många gånger vi behöver upprepa operationen för att få identitetselementet:

$$2 + 2 + 2 = 6 \equiv 0 \pmod 6.$$

Vi ser att elementet 2 har ordning 3.

I allmänhet kommer vi i fortsättningen fokusera på att arbeta med ändliga grupper. Detta görs för att förenkla analysen och fokusera på egenskaperna hos ändliga grupper. Det kommer underlätta förståelsen av grupperna utan att ta hänsyn till komplexiteten med oändliga grupper.

Definition 2.12. Delgruppen H som genereras av elementen $g_1, g_2, \ldots, g_k \in G$ definieras som den minsta delgruppen av G som innehåller dessa element. Vi skriver:

$$H = \langle g_1, g_2, \ldots, g_k \rangle.$$

Om vi har två delgrupper H_1, H_2 av en grupp G, så är även deras snitt $H_1 \cap H_2$ en delgrupp. Om vi tar snittet av alla delgrupper i G som innehåller elementen $g_1, g_2 \ldots g_k$ så har vi en delgrupp med dessa element. Eftersom snittet alltid är mindre än eller lika med alla delgrupper, så är det den minsta möjliga delgruppen som innehåller $g_1, g_2, \ldots, g_k$.

Definition 2.13. [2] En grupp G kallas cyklisk om det finns ett element $x \in G$ så att varje element i G är en potens av x. Vi säger att elementet x genererar G och vi skriver:

$$G = \langle x \rangle$$

Exempel 2.14. Ett exempel på en cyklisk grupp är $(\mathbb{Z}_5, \times)$, alltså mängden $\{1, 2, 3, 4\}$ med multiplikation modulo 5 som den binära operationen. En av gruppens generatorer är 2, eftersom

$$\begin{aligned} 2 &= 2 \\ 2^2 &= 4 \\ 2^3 &\equiv 3 \pmod 5 \\ 2^4 &\equiv 1 \pmod 5. \end{aligned}$$

Vi kan se att vi behöver upprepa operationen fyra gånger för att få tillbaka identitetselementet 1. Detta innebär att ordningen av 2 är lika med 4 enligt definition 2.10.

Sats 2.15. *[2] Låt G vara en grupp, och anta att H är en icke-tom delmängd av G. Då är H en delgrupp av G om det uppfyller följande:*

S1. $x, y \in H \Rightarrow xy \in H$

S2. $x \in H \Rightarrow x^{-1} \in H$.

Om G är ändlig så räcker det med S1 för att visa att H är en delgrupp.

Bevis. [2] Vi behöver nu visa att med dessa villkor att associativitet och identitetselementet e följer. Associativitet följer från att H är en delmängd och har därför samma element som i G, och de elementen är associativa. För e kan vi kolla på $S2$, som säger att varje element x i H har en invers x^{-1} som också är i H. Vi vet från $S1$ att xx^{-1} är i H, vi vet också att $xx^{-1} = e$ och därför finns identiteten e i H.

Nu behöver vi visa att $S2$ följer av $S1$ om G är ändlig. Om H bara består av e så följer alla egenskaper av en delgrupp, annars tar vi ett godtyckligt element x, som inte är e, i H och som har ordning m. Vi har då att $x^m = e$, låt oss multiplicera ekvationen med x^{-1}, så att vi får $x^{m-1} = x^{-1}$. Eftersom m är ett positivt heltal större än 1 så är x^{-1} lika med en positiv potens av x. Med $S1$ har vi att alla positiva potenser av x finns i H, därmed finns även x^{-1} i H. □

Vi ska nu visa en följdsats som beskriver strukturen hos en grupp som genereras av ett antal element:

Sats 2.16. *Om vi har en ändlig grupp G med elementen $g_1, \ldots, g_k \in G$ och en delgrupp $H = \langle g_1, \ldots, g_k \rangle$ så kan varje element i H skrivas som en produkt av generatorerna $g_1, \ldots, g_k$.*

Bevis. Låt M vara en mängden av alla möjliga produkter av elementen $g_1, \ldots, g_k$, vi kallar dem för ord.

Det är klart att $M \subseteq H$ eftersom H är definierad som den minsta gruppen som innehåller alla $g_1, \ldots, g_k$ och måste därför innehålla alla deras produkter. Det är också klart M är sluten under multiplikation eftersom produkten av två ord helt enkelt blir ett nytt längre ord som fortfarande består av generatorerna. Vi har då att M är en delgrupp till G enligt sats 2.15.

Eftersom M är en delgrupp som innehåller generatorerna och H är den minsta delgruppen med dessa generatorer så måste det betyda att $H \subseteq M$. Samtidigt är $M \subseteq H$ vilket ger att $H = M$.

Alltså kan Varje element i H skrivas som en produkt av generatorerna. □

Lagranges sats är centralt och väldigt intressant inom gruppteori. Eftersom vi inte kommer fördjupa oss i teorin bakom beviset skriver vi bara ner satsen. Beviset till satsen hittas i [2].

Sats 2.17 (Lagranges Sats)**.** *[2] Låt G vara en ändlig grupp med ordning n och H vara en delgrupp med ordning m, då är m en delare till n.*

2.2 Gruppverkan

En gruppverkan beskriver hur en grupp verkar på en mängd. Det innebär att varje element i gruppen opererar på ett element i mängden och ger ett nytt element i mängden. Vi definierar exakt vad vi menar med detta:

Definition 2.18. [3] Låt G vara en grupp och X en mängd. En gruppverkan är en avbildning av varje ordnat par (g, x) för $g \in G$ och $x \in X$ till ett nytt element $gx \in X$. Det vill säga, en funktion

$$\begin{aligned} G \times X &\to X \\ (g, x) &\mapsto gx, \end{aligned}$$

som har följande egenskaper:

1. **Identitetselement** $ex = x$ för alla $x \in X$
2. **Associativitet** $(gh)x = g(hx)$ för alla $g, h \in G$ och alla $x \in X$

Definition 2.19. [2] Låt G vara en grupp som verkar på en mängd X. En bana av G på ett element $x \in X$ är mängden element i X som fås genom att något $g \in G$ verkar på x, och beskrivs formellt som:

$$\text{Orb}(x) = \{gx \in X \mid g \in G\}.$$

En stabilisator av ett element $x \in X$ är mängden av alla $g \in G$ som verkar på x och lämnar x oförändrat. Mängden noteras:

$$\text{Stab}(x) = \{g \in G \mid gx = x\}.$$

Sats 2.20. *Stab(x) är en delgrupp av G.*

Bevis. För att stabilisatorn ska vara en delgrupp av G behöver den vara en icke-tom, sluten under gruppoperationen och innehålla inverser, enligt sats 2.15.

Att Stab(x) är icke-tom följer av att identitetselement $e \in G$ uppfyller $ex = x$ för alla $x \in X$ och därmed gäller $e \in \text{Stab}(x)$. Alltså är Stab(x) icke-tom.

Låt $g_1, g_2 \in \text{Stab}(x)$. Vi behöver visa slutenhet, det vill säga $g_1 g_2 \in \text{Stab}(x)$. Vi vet att $g_1(x) = x$ och $g_2(x) = x$, sammansättningen av de ger

$$(g_1 g_2)(x) = g_1(g_2(x)) = g_1(x) = x.$$

Sammansättningen lämnar alltså x oförändrat vilket innebär att $g_1 g_2 \in \text{Stab}(x)$ och stabilisatorn är sluten.

Låt $g \in \text{Stab}(x)$. Vi vill visa att $g^{-1} \in G$. Vi vet att $g^{-1}g = e$, så vi har $(g^{-1}g)(x) = ex = x$ och vi vet att $g(x) = x$, så vi har

$$x = (g^{-1}g)(x) = g^{-1}(g(x)) = g^{-1}(x).$$

Därmed $g^{-1} \in \text{Stab}(x)$.

Genom att ha uppfyllt kraven är Stab(x) en delgrupp av G enligt sats 2.15. □

Exempel 2.21. Låt G vara den gruppen med symmetrier för en kvadrat D_4 och $X = \{A, B, C, D\}$ vara mängden av kvadratens hörn som G verkar på.

Bana: Vi väljer elementet $x = B$ i mängden. Banan av D_4 på B är alla hörn som B kan avbildas till när något $g \in G$ verkar på B. Vi får

$$\begin{aligned} &e(B) = B, \quad r_1(B) = C, \quad r_2(B) = D, \quad r_3(B) = A, \\ &s_v(B) = A, \quad s_h(B) = C, \quad s_{AC}(B) = D, \quad s_{BD}(B) = B. \end{aligned}$$

Vi har fått alla element i mängden X när något element g verkar på B. Så banan är:

$$\text{Orb}(B) = \{A, B, C, D\}.$$

Stabilisator: Stabilisator för B är mängden element $g \in G$ som lämnar B oförändrat. Det ska alltså gälla att g(B) = B, vi ser på vår undersökning ovan för vilka g det gäller. Det följer att stabilisatorerna är:

$$\text{Stab}(B) = \{e, s_{BD}\}.$$

Observera att banans kardinalitet multiplicerat med stabilisatorns är lika med gruppens kardinalitet,

$$|\text{Orb}(B)| \times |\text{Stab}(B)| = 4 \cdot 2 = 8 = |G|.$$

Detta visar sig gälla allmänt:

Sats 2.22. *Låt G vara en grupp som verkar på en mängd X, och låt $x \in X$. Då gäller*

$$|Orb(x)| \times |Stab(x)| = |G|$$

Beviset till denna sats finns i [2, Theorem 21.3].

Definition 2.23. [3] En gruppverkan av en grupp G på en mängd X kallas transitiv om den bara har en bana. Alltså om

$$\text{Orb}(x) = X, \text{ för något } x \in X.$$

Ekvivalent kan vi säga att verkan är transitiv om för varje par $x, y \in X$ existerar ett $g \in G$ sådan att $gx = y$.

Exempel 2.24. Vi kan titta på exempel 2.21, där gruppen D_4 verkar på mängden av kvadratens hörn $X = \{A, B, C, D\}$. Vi såg att banan för elementet B är hela mängden X, det vill säga

$$\text{Orb}(B) = \{A, B, C, D\} = X.$$

Därför verkar D_4 transitivt på X.

Sats 2.25. *Om G verkar transitivt på X så gäller*

$$Orb(x) = X \text{ för alla } x \in X$$

Bevis. Anta att G verkar transitivt på X, alltså att det finns ett element $z \in X$ med hela mängden X som bana: $\text{Orb}(z) = X$.

Vi vill visa att banan för ett godtyckligt element är $x \in X$ är hela mängden, alltså $\text{Orb}(x) = X$.

Vi tar ett godtyckligt element $y \in X$ och eftersom $\mathrm{Orb}(z) = X$ finns ett element $g_1 \in G$ sådan att $g_1(z) = y$.

Vi tar det godtyckliga elementet $x \in X$, och likt tidigare finns ett element $g_2(z) = x$. Då gäller

$$g_1 g_2^{-1}(x) = g_1(g_2^{-1}(x)) = g_1(z) = y.$$

Eftersom G är sluten så är $g_1 g_2^{-1}$ ett element i G .Det finns alltså ett element i G som verkar på ett godtyckligt element $x \in X$ för att få ett annat godtyckligt element $y \in X$. Därför gäller att $\mathrm{Orb}(x) = X$ för alla $x \in X$. □

Definition 2.26. [2] Om ett par element i en grupp uppfyller $xy = yx$ säger vi att de kommuterar. En grupp där varje element kommuterar med varandra kallas kommutativ eller abelsk.

Exempel 2.27. Vi har gruppen $(\mathbb{Z}, +)$, som är heltal under addition. Det gäller att

$$a + b = b + a, \text{ för något } a, b \in \mathbb{Z}$$

eftersom addition av heltal är kommutativ. Så är $(\mathbb{Z}, +)$ en abelsk grupp.

Definition 2.28. Låt G vara en grupp. Två element $a, b \in G$ är konjugerade om det finns ett element $g \in G$ sådant att

$$gag^{-1} = b.$$

I detta fall sägs b vara ett konjugat av a.

Exempel 2.29. Låt oss undersöka konjugation i D_4 och se om två element är konjugerade med varandra. Vi väljer $a = r_1$ och $g = s_v$ för att hitta det element b som är konjugat till r_1. Eftersom s_v är sin egna invers, gäller $s_v^{-1} = s_v$. Vi beräknar $s_v r_1 s_v$ genom att följa varje hörn:

$$\begin{array}{lll} s_v(A) = B & r_1(B) = C & s_v(C) = D \\ s_v(B) = A & r_1(A) = B & s_v(B) = A \\ s_v(C) = D & r_1(D) = A & s_v(A) = B \\ s_v(D) = C & r_1(C) = D & s_v(D) = C. \end{array}$$

Resultatet av sammansättningen blir

$$\begin{aligned} A &\mapsto D \\ B &\mapsto A \\ C &\mapsto B \\ D &\mapsto C. \end{aligned}$$

Detta är exakt effekten av r_3 och vi har att $s_v r_1 s_v^{-1} = r_3 = b$. Alltså är r_1 och r_3 är konjugerade i D_4.

Definition 2.30. [9] Låt G vara en grupp och H en delgrupp. Delgruppen H sägs vara en normal delgrupp om för varje $h \in H$ och varje $g \in G$ gäller

$$ghg^{-1} \in H.$$

Som tidigare nämnt består en abelsk grupp av element som kommuterar, alltså att $xy = yx$ för alla x, y i gruppen. Om vi har en abelsk grupp G och en delgrupp H så är H en normal delgrupp. Låt $g \in G$ och $H \in H$, då gäller

$$ghg^{-1} = hgg^{-1} = he = h.$$

Vi ser att $ghg^{-1} \in H$ och H är därför en normal delgrupp.

Exempel 2.31. Låt D_4 vara gruppen och $H = \{e, s_v\}$ vara delgruppen. För att undersöka om H är en normal delgrupp väljer vi något element i delgruppen och något i gruppen. Vi börjar med att räkna konjugatet för $h = s_v$ och $g = r_1$ och vi får:

$$r_1 s_v r_1^{-1} = s_h.$$

Men vi vet att $s_h \notin H$ vilket innebär att H därmed inte är en normal delgrupp.

2.3 Block och primitiv gruppverkan

Tidigare har det beskrivits hur en grupp kan verka på en mängd. Den kan även verka på olika delmängder, vilket är hur vi kan hitta olika block i grupperna. Avsnittet kommer visa vad ett block är, hur de uppstår och vilka egenskaper de medför.

Definition 2.32. [6] Låt $G \times X \to X$ vara en gruppverkan av en grupp G på en mängd X. Ett block är en delmängd $B \subseteq X$ där för varje $g \in G$ gäller följande:

1. $gB = B$ (g bevarar B), eller
2. $gB \cap B = \emptyset$ (gB och B är disjunkta).

Exempel 2.33. Låt D_4 verka på mängden $X = \{A, B, C, D\}$ och delmängden, hörnen diagonalt från varandra, $B = \{A, C\}$. Vi undersöker om delmängden är ett block:

$$\begin{array}{llll} eb = \{A, C\}, & r_1 b = \{B, D\}, & r_2 b = \{C, A\}, & r_3 b = \{D, B\}, \\ s_v b = \{B, D\}, & s_h b = \{D, B\}, & b = \{A, C\}, & s_{BD} b = \{C, A\}. \end{array}$$

Eftersom varje gruppoperation på delmängden antingen bevarar B eller avbildas till en mängd disjunkt till B så är det ett block. Vi ser att mängden som är disjunkt från B, mängden $\{B, D\}$, även är ett block eftersom det antingen bevaras eller är disjunkt under operationer.

Definition 2.34. En grupp G som verkar transitivt på en mängd X utan att utgöra icke-triviala block verkar primitivt på mängden. Att ett block är trivialt innebär att det antingen är tomt, $B = \emptyset$, består av ett element, $B = \{x\}$ för något $x \in X$, eller om det är hela mängden, $B = X$.

Exempel 2.35. Vi kan se i exempel 2.33 att delmängderna som vi kan associera med diagonalerna $\{A, C\}$ och $\{B, D\}$ utgör block i D_4. Dessa block är icke-triviala eftersom de innehåller fler än ett element men består inte av hela mängden. Enligt definitionen är en gruppverkan inte primitiv om det finns något icke-trivialt block, och därför är verkan av D_4 på hörnen inte primitiv.

Exempel 2.36. Låt symmetrierna för en triangel D_3 verka på mängden $X = \{A, B, C\}$, som representerar triangelns hörn. Vi har att $|X| = 3$, vilket innebär att ett icke-trivialt block måste ha storlek exakt 2. Detta gäller eftersom ett block är trivialt om det har storlek 0,1 eller är hela mängden, alltså storlek 3 och för att undvika det behöver vi ha ett block med exakt två element. I detta fall är ett sådant block inte möjligt eftersom en mängd med två element kan inte bli disjunkt med en annan sådan mängd när hela mängden bara innehåller tre element. Det finns helt enkelt inte plats för två olika mängder med två element vardera utan att de överlappar. Verkan av D_3 på X är alltså primitiv.

3 Permutationer

I detta avsnitt kommer vi introducera vad en permutation är och dess grundläggande egenskaper. Vi kommer gå igenom vad symmetrigruppen S_n är och se hur det är en grupp.

3.1 Introduktion

En permutation är en omordning av element i en mängd och kan beskrivas med hjälp av en bijektiv avbildning. Mer formellt definieras en permutation på följande sätt:

Definition 3.1. En permutation av en icke-tom, mängd X är en bijektion från X till X. Det vill säga en avbildning $\alpha : X \to X$ som både är injektiv och surjektiv.

I denna uppsats kommer vi bara arbeta med ändliga mängder, där vi oftast kommer ha en mängd $X = \{1, 2, \ldots, n\}$ för positiva heltal n.

Exempel 3.2. Låt $X = \{1, 2, 3, 4\}$ och $\alpha : X \to X$ en permutation, definierad av ekvationerna:

$$\alpha(1) = 2, \quad \alpha(2) = 4, \quad \alpha(3) = 1, \quad \alpha(4) = 3.$$

Permutationen ovan kan beskrivas på olika sätt, ett är med tvåradsnotation:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix}$$

där den övre raden visar de ursprungliga elementen och den nedre raden visar vad de avbildas på.

I fortsättningen kommer permutationer beskrivas i cykelnotation. För samma permutation α skriver vi

$$(1243)\,.$$

Det visar att elementet avbildas till elementet höger och kommer sen tillbaka till första elementet, i detta fall är avbildningarna:

$$1 \mapsto 2 \mapsto 4 \mapsto 3 \mapsto 1.$$

I allmänhet kan en permutation skrivas som en eller flera cykler:

Exempel 3.3. Låt β vara en permutation av mängden $\{1, 2, 3, 4, 5, 6\}$ med avbildningarna

$$\beta(1) = 3, \quad \beta(2) = 6, \quad \beta(3) = 4, \quad \beta(4) = 1, \quad \beta(5) = 5, \quad \beta(6) = 2.$$

Vi kan då skriva permutationen i cykelnotation som:

$$\beta = (1\,3\,4)(2\,6)(5).$$

När man skriver cykelnotation i praktiken utelämnar man ofta cykler av längd 1 eftersom de är triviala, man skriver:

$$\beta = (1\,3\,4)(2\,6).$$

Vi kommer i fortsättningen skriva cykelnotationer på detta sätt.

Definition 3.4. En permutatation av en mängd $X = \{1, 2, \ldots, n\}$, som lämnar varje element oförändrat kallas för identitetspermutationen, det noteras id_X. Formellt beskrivs det

$$id_X(i) = i \text{ för varje } i \in \{1, 2, \ldots, n\}.$$

I fortsättningen kommer vi använda notationen id då mängden kommer vara given.

Som tidigare visat skriver vi inte ut de triviala cyklerna. Så identitetspermutationen id som bara består av triviala cykler kommer skrivas som ().

Definition 3.5. En transposition är en permutation som ändrar exakt två element och lämnar alla andra oförändrade. Ekvivalent till det, en permutation med en cykel av längd två och alla övriga cykler av längd ett.

Exempel 3.6. Om vi har en permutation α på mängden $X = \{1, 2, 3, 4\}$ och har avbildningarna $\alpha(1) = 3,\ \alpha(2) = 2,\ \alpha(3) = 1$ och $\alpha(4) = 4$. Då är denna permutation en transposition, och vi skriver

$$(13).$$

Sats 3.7. *Det finns $n!$ stycken permutationer för mängden $\{1, 2, \ldots, n\}$.*

Bevis. Låt mängden $X = \{1, 2, \ldots, n\}$ och låt oss hitta alla olika permutationer till den mängden. Varje element ska avbildas exakt en gång. Låt oss titta hur många sätt vi kan arrangera det. För det första elementet har vi n val. Efter att ett element blivit valt kan vi välja något av resterande, så vi har $n - 1$ val. Nu är 2 element valda och vi har $n - 2$ alternativ och detta fortsätter tills det bara är

ett alternativ kvar. För att se hur många sätt vi kan göra detta på kan vi använda oss av produktregeln:

$$n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots \cdot 1 = n!.$$

Detta visar att vi har $n!$ olika permutationer för mängden X. □

Mängden av alla permutationer av en mängd $\{1, 2, \ldots, n\}$ har notationen S_n och kallas för symmetrigruppen, på n element.

Exempel 3.8. S_3 består av följande $3! = 6$ permutationer:

$$(), (23), (12), (123), (132), (13).$$

Sats 3.9. *Symmetrigruppen S_n är en grupp.*

Detta innebär att gruppens egenskaper gäller för alla permutationer i S_n under sammansättning. Alltså är S_n sluten, associativ, innehåller identitetselementet och ett inverselement för varje element.

Bevis. Vi vet att enligt definition 3.1 att permutationer är bijektioner från en ändlig mängd till sig själv. Vi kan kontrollera att egenskaperna för en grupp följer:

Sammansättningen av två bijektiva funktioner är också bijektiv vilket visar slutenhet. Alltså är sammansättningen av två permutaioner i S_n även en permutation i S_n

För associativitet, har vi att sammansättningar av funktioner alltid är associativa. För permutationer $\alpha, \beta, \gamma \in S_n$ gäller då $\alpha(\beta\gamma) = (\alpha\beta)\gamma$.

Att ha en bijektion av en mängd till sig självt innebär att det alltid finns bijektioner som avbildar ett element på samma element. Vi har att $id(\alpha) = \alpha$

En invers är en bijektion från en mängd till sig själv vilket innebär att det existerar även för permutationer.

Symmetrigruppen S_n är alltså en grupp under sammansättning. □

Exempel 3.10. Låt $\alpha = (134)(25)$ och $\beta = (1425)$ vara permutationer. En sammansättning $\alpha\beta$ skulle se ut på följande sätt, för första elementet:

$$\begin{aligned}\alpha(\beta(1)) &= \alpha(4) = 1\\ \alpha(\beta(2)) &= \alpha(5) = 2\\ \alpha(\beta(3)) &= \alpha(3) = 4\\ &\vdots\end{aligned}$$

och så vidare för de andra elementen. Sammansättningen blir till slut $\alpha\beta = (345)$.

3.2 Ordning

Eftersom symmetrigruppen är en grupp så gäller definition 2.7 för gruppens ordning även för permutationsgrupper. Ordningen bestäms alltså av gruppens kardinalitet.

Exempel 3.11. Om vi har symmetrigruppen S_5 så vet vi att den innehåller $5! = 120$ element. Alltså har S_5 har ordning 120.

På samma sätt följer definition 2.10, ordningen av ett element. Så ordningen av en permutation α bestäms av det minsta heltal k så att $\alpha^k = id$. Om en permutation har en enda cykel så är ordningen lika med dess längd eftersom varje element återgår till sin ursprungliga plats efter exakt så många tillämpningar.

Exempel 3.12. Låt $\alpha = (1423)$ vara en permutation Vi ser att det är en cykel med längd fyra, så vi räknar α^4 för det första elementet:

$$\alpha(1) = 4,\ \alpha^2(1) = 2,\ \alpha^3(1) = 3,\ \alpha^4(1) = 1$$

Vi ser att elementet återvänder till sin ursprungliga plats efter 4 tillämpningar och det samma gäller även för resterande element. Så $\alpha^4 = id$ och permutationen har ordning 4.

Sats 3.13. *För en permutation som är en produkt av flera disjunkta cykler är ordningen lika med cykellängdernas minsta gemensamma multipel (MGM).*

Bevis. Låt, för enkelhetens skull, $\alpha = (a_1a_2 \dots a_l)(b_1b_2 \dots b_m)$ vara en permutation med två disjunkta cykler. Vi behöver hitta det minsta heltal k så att $\alpha^k = id$. Vi har

$$\alpha^k = ((a_1a_2 \dots a_l)(b_1b_2 \dots b_m))^k = (a_1a_2 \dots a_l)^k(b_1b_2 \dots b_m)^k.$$

Vi behöver att båda cyklerna återgår till identiteten:

- För att $(a_1 a_2 \dots a_l)$ ska återgå till id så måste k vara en multipel av l.
- För att $(b_1 b_2 \dots b_m)$ ska återgå till id så måste k vara en multipel av m.

Så för att båda cykler ska gälla måste k vara en multipel av dess längd l och m. Ordningen bestäms av det minsta heltal så att $\alpha^k = id$ gäller, och därför är ordningen den minsta gemensanma multipeln (MGM) av cykellängderna, $MGM(l, m)$ i detta fall. Detta fungerar på samma sätt för permutationer med fler än två cykler. □

Exempel 3.14. Låt $\beta = (13)(245)$ vara en permutation med två disjunkta cykler av längd 2 och 3. Eftersom cyklerna är disjunkta, bestäms ordningen enligt sats 3.13 av minsta gemensamma multipeln (MGM) av deras längder, 2 och 3, vilket är 6. Med beräkning kan man kontrollera att $\beta^6 = id$ gäller.

Ett sätt att beskriva en permutations cykler och dess längder är genom cykeltyper.

Definition 3.15. Låt $\sigma \in S_n$. Cykeltypen av σ är partitionen $[1^{\alpha_1}, 2^{\alpha_2}, \cdots, n^{\alpha_i}]$ av n där α_i är antalet i-cykler av σ.

Exempel 3.16. Låt $\sigma = (351)(62)(4)(7)$. Då har σ cykeltypen $[1^2, 2^1, 3^1]$.

4 Permutationsgrupper

En permutationsgrupp är en grupp av permutationer av en ändlig mängd. Vi kommer dra kopplingar mellan tidigare diskuterade egenskaper hos grupper och hur det kan fungera för permutationsgrupper. Vi kommer även undersöka intressanta observationer som följer av dem.

Om vi har en permutationsgrupp $G = \langle g_1, \ldots, g_k \rangle$ så är alla permutationer i G sammansättningar av elementen g_i för $i \in \{1 \ldots k\}$. Dessa element kallas för generatorerna för permutationsgruppen.

Definition 4.1. Låt G vara en mängd permutationer av en ändlig mängd X. Om G är en grupp så säger vi att G är en permutationsgrupp av X.

Detta innebär, enligt definition 2.1, att G är sluten och associativ under permutationssammansättning, innehåller ett identitetselement och att varje permutation i G har en invers som också ligger i G.

Har vi en delmängd av S_n som uppfyller kraven för en grupp så har vi, enligt definition 2.3, en delgrupp av S_n. Alla delgrupper vi har i S_n är alltså permutationsgrupper.

Exempel 4.2. Delgrupperna i S_3 är:

$$H_1 = \{()\}, \qquad H_2 = \{(), (12)\}, \qquad H_3 = \{(), (13)\},$$
$$H_4 = \{(), (23)\}, \qquad H_5 = \{(), (123), (132)\}, \qquad H_6 = S_3$$

Vi har i tidigare avsnitt introducerat gruppverkan, som beskriver hur en grupp verkar på en mängd. När det gäller permutationsgrupper, så fungerar det på samma sätt och det följer helt enkelt från definitionen 2.18.

Exempel 4.3. Låt G vara symmetrigruppen S_3 och $X = \{1, 2, 3\}$ vara mängden som G verkar på.

Bana: Vi väljer ett element $x = 2$ i mängden. Banan av S_3 på 2 är alla värden som 2 kan avbildas till i S_3. Vi får

$$\mathrm{Orb}(2) = \{g(2) \mid g \in S_3\} = \{1, 2, 3\}.$$

Detta innebär att från 2 kan vi nå alla andra värden i X genom permutationer i S_3.

Stabilisator: Vi har elementet 2 och vill hitta dess stabilisator, betecknat Stab(2). Stabilisatorn är mängden permutationer som lämnar 2 oförändrat. Vi har:

$$\mathrm{Stab}(2) = \{g \in G \mid g(2) = 2\} = \{(), (13)\}.$$

Dessa två permutationer, identitetspermutationen och permutationen (13), utgör stabilisatorn för elementet 2.

Vi har tidigare tittat på definitionen för transitiv och primitiv gruppverkan. När det gäller permutationer handlar det om en grupp som verkar på sin egna mängd så vi säger att själva permutationsgruppen är transitiv eller primitiv.

Enligt definition 2.23 är en transitiv permutationsgrupp är en grupp med bara en bana och det gäller att:

$$\mathrm{Orb}(x) = X, \text{ för något } x \in X.$$

Transitiva permutationsgrupper är viktiga inom gruppteori, det kan till exempel vara väldigt användbart i exempelvis galoisteori [10].

En primitiv permutationsgrupp är en grupp där inga icke-triviala block finns.

Vissa egenskaper hos permutationsgrupper kan följa av generatorernas egenskaper, till exempel om generatorerna kommuterar följer det att hela gruppen är abelsk.

Sats 4.4. *Om generatorerna för en grupp kommuterar så är hela gruppen abelsk.*

Bevis. Låt $G = \langle g_1, g_2, \cdots, g_n \rangle$ vara en grupp där generatorerna kommuterar, det vill säga $g_i g_j = g_j g_i$ för alla i, j. Vi vill visa att $xy = yx$ för varje $x, y \in G$. Eftersom G genereras av $\{g_1, \cdots, g_n\}$ kan alla element skrivas som produkter av generatorerna enligt sats 2.16. Vi har

$$x = g_{i_1} g_{i_2} \cdots g_{i_m},\ y = g_{j_1} g_{j_2} \cdots g_{j_r}.$$

Eftersom generatorerna är kommutativa kan vi byta plats på dem så att alla faktorer för y hamnar till vänster om faktorerna för x:

$$xy = (g_{i_1} g_{i_2} \cdots g_{i_m})(g_{j_1} g_{j_2} \cdots g_{j_r}) = \cdots = (g_{j_1} g_{j_2} \cdots g_{j_r})(g_{i_1} g_{i_2} \cdots g_{i_m}) = yx$$

Alltså kommuterar alla element $x, y \in G$ så gruppen G är abelsk. □

Att egenskaper för generatorerna medför egenskaper till hela gruppen gäller även för normalitet och konjugering. Det räcker alltså att kontrollera om generatorerna är normala för att säga att hela gruppen är normal, på samma sätt för att kontrollera om två grupper är konjugerade. Detta bygger på samma princip som i beviset ovan.

I boken [3] finns övningen 1.2.14 där man ska ta reda på vilka villkor som finns för att generera hela S_n med en permutation av n-cykel, $(12 \ldots n)$ och en transposition (ij). Vi visar att så är fallet om $(ij) = (12)$:

Sats 4.5. *Permutationerna* (12) *och* $(12\dots n)$ *genererar hela* S_n*.*

Bevis. Vi vill visa att $S_n = \langle (12), (12\dots n) \rangle$.

Låt $\gamma = (12\dots n)$ och $\tau = (12)$ vara permutationer. Vi undersöker vad som händer när vi utför konjugeringen $\gamma^k \tau \gamma^{-k}$ för positiva heltal $k \pmod{n}$:

$$\begin{aligned} \gamma\tau\gamma^{-1} &= (23) \\ \gamma^2\tau\gamma^{-2} &= (34) \\ \gamma^3\tau\gamma^{-3} &= (45) \\ &\vdots \end{aligned}$$

Vi ser att konjugationerna ger transpositionen (ab) där $b \equiv a+1 \pmod{n}$. Med hjälp av dessa element kan kan vi vidare göra konjugationerna

$$\begin{aligned} (23)(12)(23)^{-1} &= (13) \\ (34)(13)(34)^{-1} &= (14) \\ (45)(14)(45)^{-1} &= (15) \\ &\vdots \end{aligned}$$

Om man fortsätter på detta sätt ser att vi får alla transpositioner på formen $(1\,a)$. Genom konjugation av elementen vi har kan vi få resterande transpositioner:

$$(1\ a)(1\ b)(1\ a)^{-1} = (a\ b)$$

Vi har nu alla transpositioner. Enligt sats i [2] kan alla permutationer av en ändlig mängd uttryckas som en produkt av transpositioner. Vi har alla transpositioner och därmed alla permutationer i S_n. □

5 Algoritmer

I detta avsnitt presenteras två algoritmer, en egen algoritm för att generera alla element i en grupp samt en algoritm från litteraturen för att hitta block för en permutationsgrupp. Båda algoritmerna bygger på teori som vi tidigare gått igenom. För den första algoritmen ges ett bevis som grundas i arbetets teori och visar att algoritmen faktiskt genererar alla element i delgruppen. Beviset för den andra algoritmen är finns i litteraturen. Genom implementering i Python kan teorin testas i praktiken och läsaren har möjligheten att själv undersöka dessa.

5.1 Generera en delgrupp

Från 2.12 har vi: Delgruppen H som genereras av en mängd element $\{g_1, g_2, \ldots, g_k\}$ i en grupp G definieras som den minsta delgruppen av G som innehåller dessa element. Vi skriver:

$$H = \langle g_1, g_2, \ldots, g_k \rangle.$$

Algoritm 1: algoritmen för att generera alla element i en delgrupp

Algoritmen genererar en mängd M enligt följande:

1. Till en början är M en tom mängd. Vi definierar en lista Q med ett element som är identitetselementet id.
2. Ett element C tas bort från Q och används som det aktuella elementet.
3. Elementet C läggs till i mängden M.
4. Vi itererar över våra generatorer g_i och beräknar produkt $p = C \cdot g_i$ för varje generator. Om p inte redan finns i M, läggs p till i Q. Annars går vi vidare till nästa generator.
5. Om Q inte är tom till steg 2.
6. Mängden M returneras.

Sats 5.1. *Algoritm 1 genererar alla element i delgruppen.*

Bevis. Eftersom M bildas av produkter av generatorerna g_i och id kan vi säga att alla generatorer ligger i M och att M är en delmängd till $H = \langle g_1, g_2, \ldots, g_k \rangle$. Eftersom S_n är ändlig behöver vi bara visa att M är sluten under multiplikation för att M ska vara en delgrupp till H enligt sats 2.15. Vi ska alltså visa egenskapen:

$$x, y \in M \Rightarrow xy \in M$$

Vi kan från algoritmen säga att:

$$Mg_i = M \text{ för alla } i.$$

Detta innebär att om vi tar elementet $x \in M$ och multiplicerar det med en generator så kommer produkten ligga i M. Å andra sidan kan vi återigen, utifrån algoritmen, skriva $y \in M$ som produkt av generatorer:

$$y = g_{i_1} \cdot g_{i_2} \cdot \ldots \cdot g_{i_s}.$$

Nu har vi

$$x \cdot y = x \cdot g_{i_1} \cdot g_{i_2} \cdot \ldots \cdot g_{i_s}.$$

Vi kan från egenskapen $Mg_i = M$ säga att $x \cdot g_{i_1}$ ligger i M, vi kallar produkten x_1. Vidare har vi:

$$x \cdot y = x_1 \cdot g_{i_2} \cdot \ldots \cdot g_{i_s}.$$

På samma sätt gäller $x_1 \cdot g_{i_2} \in M$. Denna process kan vi fortsätta med tills vi kommer till att $xy \in M$. Vi har då att M är sluten under multiplikation. Eftersom M även innehåller identiteten och är ändlig ger det att M är en delgrupp. Då M innehåller generatorerna så gäller det att $M = H$ och att algoritmen genererar alla element i delgruppen. □

5.2 Hitta block

Algoritm 2: Algoritmen för att hitta block i en permutationsgrupp

Att skapa en algoritm som hittar alla block givet en permutationsgrupp är relativt avancerat. För att vara säker på att den fungerar rätt har jag använt algoritmen i artikeln Atkinson [1]. Algoritmen hittar alla block som innehåller element 1 och angivet ett element $\omega \in \Omega$ och $\omega \neq 1$, där $\Omega = \{1, 2 \ldots n\}$ är mängden som permuteras.

1. Initialt är C en tom lista och för alla $\alpha \in \Omega$ är $f(\alpha) = \alpha$.
2. Lägg till ω till C och sätt $f(\omega) = 1$.
3. Ta bort ett element β från C och beräkna $\alpha = f(\beta)$.
4. Sätt ett heltal j till 0.
5. Öka j med 1 och beräkna $\gamma = \alpha g_j$ och $\delta = \beta g_j$.
6. Om $f(\gamma)$ och $f(\delta)$ är lika, gå till 9.

7. Se till att $f(\delta) < f(\gamma)$ genom att byta plats på γ och δ om nödvändigt.
8. Definiera de värden $f(\epsilon)$ av f som är lika med $f(\gamma)$ för att ge dem det nya värdet $f(\delta)$ och lägg till det gamla värdet av $f(\gamma)$ till C.
9. Om $j < m$, gå till 5.
10. Om C inte är tom, gå till 3.

När detta är klart har vi fått funktionen f färdig, där varje element i Ω har ett värde som representerar vilket block det tillhör. Alla element inom samma block har tilldelats samma värde. För att konstruera blocken samlas alla element med samma värde i en mängd, och de mängderna samlas i sin tur i en lista. Resultatet är en lista med mängder, där varje mängd representerar ett block.

Att bevisa att algoritmen fungerar är komplicerat men beviset finns i [1].

Denna algoritm har jag översatt till fungerande kod i Python för att följa algoritmens steg exakt.

6 Implementering av permutationsgrupper

För att tydligt koppla teorin med praktiska exempel har vi skrivit Pythonkod att arbeta med permutationsgrupper. Syftet koden är att vara enkel att använda, och spegla teorierna som vi tidigare diskuterat. Den gör det möjligt för läsaren att själv testa olika strukturer och teorier, exempelvis konstruera permutationer, generera grupper och hitta block. Koden är ett praktiskt och interaktivt verktyg som läsaren kan använda sig av för att förstå varje del av teorin inom permutationsgrupper vi diskuterat. Med detta kan man direkt se hur teorin fungerar i praktiken med konkreta och egna exempel.

Kodens fokus ligger på enkelhet framför snabbhet. Den är menad till att vara lättläst och pedagogisk samt ha tydliga kopplingar till textens formulering. Koden är inte menad att ha en så bra prestanda som möjlig. Algoritmerna genererar exempelvis alla element i permutationsgrupper, speciellt hela symmetrigruppen S_n vilket lägger stor belastning på minnet vid större n. Detta begränsar vår kod så att vi bara kan jobba i mindre grader, $n \leq 9$.

Jag har själv implementerat de matematiska algoritmerna, och handledaren har hjälpt till med vissa tekniska aspekter av Pythonklasser. Särskilt det som går utanför kursen Programmeringsteknik, exempelvis för att göra instanser av klassen SymmetricGroup unika, som är viktigt för prestandan.

6.1 Grundläggande tillämpning

I detta kapitel kommer några exempel på hur man kan använda koden att visas. Den består av tre klasser och importeras på följande sätt

```
from permutation import Permutation, SymmetricGroup, PermutationGroup
```

För att hantera permutationer och utföra operationer på dem används klassen Permutation. Denna klass är grunden till resterande klasser så att vi kan arbeta med mer utvecklade beräkningar.

För att skapa en permutation matar vi in en lista med andra raden i tvåradsnotationen. På följande sätt konstruerar vi permutationen (134)(25):

```
a = Permutation([3, 5, 4, 1, 2]); a
```

```
(134)(25)
```

Vi kan undersöka permutationens egenskaper, exempelvis hur enskilda element avbildas, cykelstrukturen, ordningen, inversen och potenser:

```
print(a(3), a.cycles, a.order(), ~a, a**6)
```

```
4 [(1, 3, 4), (2, 5)] 6 (143)(25) ()
```

Vi skapar en annan permutation för att se sammansättningar och egenskaperna mellan dem:

```
s = Permutation([4, 3, 2, 5, 1]) # (145)(23)
print(s * a, a * s, a * s == s * a)
```

```
(12)(35) (24)(35) False
```

Vi kan från detta exempelvis se att permutationerna a och s inte kommuterar.

Vi går vidare till klassen SymmetricGroup där vi kan konstruera hela symmetrigruppen S_n för alla $n \leq 9$, som tidigare nämnts.

Vi konstruerar symmetrigruppen S_3 där vi kan se gruppens alla element i gruppen:

```
S3 = SymmetricGroup(3)
print(S3.elements)
```

```
[(), (23), (12), (123), (132), (13)]
```

Med denna klass kan vi konstruera permutationer genom att mata in deras cykler på följande sätt:

```
S7 = SymmetricGroup(7)
g1 = S7([(1, 2, 3)])
g2 = S7([(1, 2), (3, 4)])
print(g1, g2)
```

```
(123) (12)(34)
```

Vi ser att de cyklerna vi matade in har nu konstruerats till permutationer.

Slutligen har vi nu klassen PermutationGroup som är en klass som gör det möjligt att konstruera permutationsgrupper genom att mata in generatorer. Med valda permutationerna genereras hela gruppen automatiskt för att sen möjliggöra analyser och se egenskaper så som ordning, banor och block.

Låt oss ta permutationerna vi precis skapade, $g1, g2$ och generera en permutationsgrupp av dem:

```
G = PermutationGroup(g1, g2)
print(G)
sorterade_element = sorted(G, key=lambda x: x.order())
print('Element:', sorterade_element)
print('Elementens ordning', sorted(g.order() for g in G))
```

```
A permutation group of order 12 in SymmetricGroup(7)
Element: [(), (12)(34), (14)(23), (13)(24), (123), (124), (142),
↪  (132), (234), (143), (243), (134)]
Elementens ordning [1, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3]
```

Som tidigare visats har permutationsgrupper flera olika egenskaper, och det kan vi undersöka med koden:

```
print(G.orbits())
print('Transitiv:', G.is_transitive(),
      'Abelsk:', G.is_abelian(),
      'Normal:', G.is_normal())
```

```
[{1, 2, 3, 4}, {5}, {6}, {7}]
Transitiv: False Abelsk: False Normal: False
```

Vi skapar en ny permutationsgrupp i S_6 för att undersöka gruppens block och primitivitet:

```
S6 = SymmetricGroup(6)
g1 = S6([(1, 4), (2, 5), (3, 6)])
g2 = S6([(1, 2, 3), (4, 5, 6)])
G = PermutationGroup(g1, g2)
blocks = G.minimal_blocks()
print(G)
print('Transitiv:', G.is_transitive())
print('Block:', blocks)
```

```
A permutation group of order 6 in SymmetricGroup(6)
Transitiv: True
Block: [{1, 4}, {2, 5}, {3, 6}]
```

Vi ser att vi får tre olika block och med det har vi att gruppen inte är primitiv.

6.2 Mer avancerat

Vi kan undersöka en permutationsgrupps struktur genom att generera en grupp-tabell. För att skapa en snygg tabell importerar vi Dataframe från pandas. Vi multiplicerar varje element med varje annat i gruppen och visar resultatet med hjälp av Dataframe. Vi kollar grupp-tabellen för S_3:

```
from pandas import DataFrame
els = sorted(S3, key=lambda g: g.order())
table = [[x * y for y in els] for x in els]
df = DataFrame(table, index=els, columns=els); df
```

	()	(23)	(12)	(13)	(123)	(132)
()	()	(23)	(12)	(13)	(123)	(132)
(23)	(23)	()	(132)	(123)	(13)	(12)
(12)	(12)	(123)	()	(132)	(23)	(13)
(13)	(13)	(132)	(123)	()	(12)	(23)
(123)	(123)	(12)	(13)	(23)	(132)	()
(132)	(132)	(13)	(23)	(12)	()	(123)

Med hjälp av tabellen ser vi alla kombinationer av alla permutationer som finns i symmetrigruppen. Den gör det enkelt att se flera egenskaper i gruppen, exempelvis om den är abelsk, hitta inverser, hitta ett elements ordning och gruppens

generatorer. Med koden kan man skapa liknande tabeller för andra symmetri- och permutationsgrupper.

Exempel 6.1. Vi tittar på den tidigare nämnda övningen i [3], där vi ska hitta villkoren för att generera S_n med en n-cykel $(12\dots n)$ och en transposition (ij).

Med hjälp av koden kan vi snabbt testa olika transpositioner för ett valt n. Detta kan hjälpa oss att hitta svaret på övningen. Vi genererar en permutationsgrupp av n-cykeln och en transposition för att kontrollera om gruppen är lika med S_n. Vi väljer S_6 och testar transpositionen (34):

```
g1 = S6([(1,2,3,4,5,6)])
g2 = S6([(3,4)])
G1 = PermutationGroup(g1,g2)
G1 == S6
```

```
True
```

Här ser vi att permutationsgruppen genererade hela S_6. Låt oss nu testa med transpositionen (13):

```
g1 = S6([(1,2,3,4,5,6)])
g2 = S6([(1, 3)])
G2 = PermutationGroup(g1,g2)
G2 == S6
```

```
False
```

Denna gången fick vi inte hela S_6 och det beror på att vi får block som skapar strukturer i mängden. Som tidigare diskuterats skapas block eftersom permutationerna gör att en mängd antingen bevaras eller blir disjunkt. Detta leder till att vi får block som bara permuteras inom blockstrukturen och aldrig utanför, vilket gör att vi inte kan nå alla element i S_n.

Vi undersöker detta med grupperna vi precis skapat:

```
print('G1 = S6:', G1 == S6)
print('G1 block:', G1.minimal_blocks())
print('G2 = S6:', G2 == S6)
print('G2 block:', G2.minimal_blocks())
```

```
G1 = S6: True
G1 block: [{1, 2, 3, 4, 5, 6}]
G2 = S6: False
G2 block: [{1, 3, 5}, {2, 4, 6}]
```

Vi vet därför att $G2$ inte är primitiv, vilket S_n är. Det kan vi kontrollera med koden:

```
print('G1 Primitiv:', G1.is_primitive())
print('G2 Primitiv:', G2.is_primitive())
```

```
G1 Primitiv: True
G2 Primitiv: False
```

Från The On-Line Encyclopedia of Integer Sequences (OEIS) kan vi se antalet transitiva [4] och primitiva [5] permutationsgrupper av grad n:

n	1	2	3	4	5	6	7	8	9	10	11	12	13
Transitiva	1	1	2	5	5	16	7	50	34	45	8	301	9
Primitiva	1	1	2	2	5	4	7	7	11	9	8	6	9

När man vet hur många transitiva och primitiva grupper av en viss ordning det finns kan man relativt enkelt hitta alla grupper, genom att pröva olika kombinationer av generatorerna. Med lite hjälp av handledaren har vi fått alla alla transitiva och primitiva permutationsgrupp upp till grad 7. Vi kan se deras generatorer och undersöka deras egenskaper. Detta är en tabell som visar alla 16 transitiva grupper för $n = 6$:

Grupp	Ordning	Normal	Abelsk	Primitiv	Block
$\langle(123456)\rangle$	6	False	True	False	{1, 4}, {2, 5}, {3, 6}
$\langle(123)(456), (14)(26)(35)\rangle$	6	False	False	False	{1, 4}, {2, 5}, {3, 6}
$\langle(123456), (26)(35)\rangle$	12	False	False	False	{1, 4}, {2, 5}, {3, 6}
$\langle(123)(456), (24)(35)\rangle$	12	False	False	False	{1, 6}, {2, 4}, {3, 5}
$\langle(123456), (12)(34)(56)\rangle$	18	False	False	False	{1, 3, 5}, {2, 4, 6}
$\langle(123456), (36)\rangle$	24	False	False	False	{1, 4}, {2, 5}, {3, 6}
$\langle(123)(456), (24)(36)\rangle$	24	False	False	False	{1, 5}, {2, 6}, {3, 4}
$\langle(123)(456), (12)(34)(56)\rangle$	24	False	False	False	{1, 5}, {2, 6}, {3, 4}
$\langle(123456), (35)(46)\rangle$	36	False	False	False	{1, 3, 5}, {2, 4, 6}
$\langle(123)(456), (14)(2536)\rangle$	36	False	False	False	{1, 2, 3}, {4, 5, 6}
$\langle(123456), (23)(56)\rangle$	48	False	False	False	{1, 4}, {2, 5}, {3, 6}
$\langle(123)(456), (34)(56)\rangle$	60	False	False	True	{1, 2, 3, 4, 5, 6}
$\langle(123456), (46)\rangle$	72	False	False	False	{1, 3, 5}, {2, 4, 6}
$\langle(123456), (34)(56)\rangle$	120	False	False	True	{1, 2, 3, 4, 5, 6}
$\langle(123)(456), (12)(3456)\rangle$	360	True	False	True	{1, 2, 3, 4, 5, 6}
$\langle(123456), (56)\rangle$	720	True	False	True	{1, 2, 3, 4, 5, 6}

Med tabellen kan vi snabbt identifiera en icke-primitiv grupp och dess icke-triviala block. Låt oss undersöka en av de grupperna lite närmare:

Exempel 6.2. Vi undersöker den icke-primitiva gruppen av ordning 48 i tabellen:

```
S6 = SymmetricGroup(6)
g1 = S6([(1, 2, 3, 4, 5, 6)])
g2 = S6([(2, 3), (5, 6)])
G = PermutationGroup(g1,g2)
blocks = G.minimal_blocks()

print(blocks)
print([g1(block) for block in blocks])
print([g2(block) for block in blocks])
```

```
[{1, 4}, {2, 5}, {3, 6}]
[{2, 5}, {3, 6}, {1, 4}]
[{1, 4}, {3, 6}, {2, 5}]
```

Vi ser att gruppen G har ett blocksystem med blocken:

$$B_1 = \{1, 4\}, \quad B_2 = \{2, 5\}, \quad B_3 = \{3, 6\}.$$

Om vi tittar på hur generatorerna verkar på blocken får vi att den första generatorn g_1 skapar avbildningarna:

$$B_1 \mapsto B_3 \mapsto B_2 \mapsto B_1$$

och andra generatorn g_2 skapar

$$B_1 \mapsto B_1, \quad B_2 \mapsto B_3 \mapsto B_2$$

Detta innebär att G skapar en permutation av blocken, vi får en grupphomomorfi:

$$\varphi : G \to S_3.$$

Vi ser alltså att gruppen är icke-primitiv och att samtidigt har vi en gruppverkan på blocken. Detta gör det möjligt att reducera större och icke-primitiva permutationsgrupper för att lättare undersöka deras struktur.

Med dessa exempel har vi visat hur teorin bakom permutationsgrupper kan användas med praktiska verktyg och vi binder den teoretiska grunden med tillämpningar för fortsatt utforskande inom ämnet.

7 Sammanfattning

Med detta arbete har vi systematiskt byggt förståelse för permutationsgrupper, från grundläggande gruppteori till mer avancerande begrepp. Genom teori, algoritmer och praktisk implementation har vi visat hur grupper kan analyseras och förstås. Kombinationen av teori och kod ger ett verktyg till det teoretiska vilket förhoppningsvis ger läsaren en ännu djupare förståelse. Detta ger en bra grund för att vidare utforska gruppteori.

Utöver detta har arbetet visat hur teoretiska begrepp inom gruppteori kan konkretiseras genom programmering. De implementerade algoritmerna visar hur definitioner kan översättas till användbara verktyg, vilket underlättar inlärning. Genom att koppla teori skapas också en möjlighet att experimentera och på egen hand verifiera satser och egenskaper hos permutationsgrupper.

Arbetet har vissa begränsningar såsom att vi bara kan hantera symmetrigrupper upp till grad 9. Med det finns det möjlighet för framtida utveckling med mer optimerade algoritmer och kodning.

8 Kod

Detta är den kompletta koden som skrivits i Python för arbetet och som använts i tillämpningarna. Nedan finns även en länk för läsaren att ladda ner koden för att själv utforska den.

Länk till kod finns här.

```python
from math import lcm
from functools import cache
from itertools import permutations


class Permutation:
    """
    Klass för individuella permutationer på mängder av typen
    {1, 2, ..., n}.

    En permutation skapas från en permutationslista (andra
    raden i tvåradsnotationen), t ex

    Permutation([3, 2, 4, 1, 6, 5])  # skapar permutation (134)(56)
    """

    def __init__(self, perm_list):
        self.perm_list = tuple(perm_list)
        self.n = len(self.perm_list)
        self.cycles = self._find_cycles()

    @cache
    @staticmethod
    def identity(n):
        return Permutation(range(1, n + 1))

    def __eq__(self, other):
        return self.perm_list == other.perm_list

    def __hash__(self):
        return hash(self.perm_list)

    def _find_cycles(self):
        """Identifierar cykler i permutationen."""
        res = []
        visited = []
```

```
37          for i in range(self.n):
38              if i not in visited:
39                  cycle = []
40                  current = i
41                  while current not in visited:
42                      visited.append(current)
43                      cycle.append(current + 1)
44                      current = self.perm_list[current] - 1
45                  res.append(tuple(cycle))
46          return res
47
48      @cache
49      def order(self):
50          "Beräknar permutationens ordning."
51          cycle_lengths = [len(cycle) for cycle in self.cycles]
52          return lcm(*cycle_lengths)
53
54      def __call__(self, B):
55          "Permutationen verkar på B. B kan vara både heltal eller en
            ↪ mängd."
56          if isinstance(B, set):
57              return {self(k) for k in B}
58          return self.perm_list[B - 1]
59
60      def __invert__(self):
61          res = [0] * len(self.perm_list)
62          for index, value in enumerate(self.perm_list):
63              res[value - 1] = index + 1
64          return Permutation(res)
65
66      def __mul__(self, other):
67          res = [self.perm_list[i - 1] for i in other.perm_list]
68          return Permutation(res)
69
70      def __pow__(self, exp):
71          exp = exp % self.order()
72          if exp == 0:
73              return Permutation.identity(self.n)
74          res = self
75          for i in range(1, exp):
76              res = self * res
77          return res
78
```

```
79      def __repr__(self):
80          res = str(tuple([tuple(cycle) for cycle in self.cycles if
            ↪  len(cycle) > 1]))
81          res = res.replace(',', '').replace(' ', '')
82          if res == '()':
83              return res
84          return res[1:-1]
85
86
87  class SymmetricGroup:
88      """
89      Klass för symmetriska gruppen S_n, upp till n = 9.
90
91      Instanser SymmetricGroup(n) av klassen är unika för varje n.
92      """
93
94      MAX_SIZE = 9  # Prestandabegränsning
95      groups = {}
96
97      def __new__(cls, n):
98          if n not in cls.groups:
99              cls.groups[n] = super().__new__(cls)
100         return cls.groups[n]
101
102     def __init__(self, n):
103         if hasattr(self, 'n'):
104             return
105         if n > self.MAX_SIZE:
106             raise ValueError('För stor SymmetricGroup')
107         self.n = n
108         self.identity = Permutation.identity(n)
109         self.elements = self._generate_elements()
110         self.order = len(self.elements)
111         self.gens = self._standard_gens()
112
113     def __repr__(self):
114         return f'SymmetricGroup({self.n})'
115
116     def _standard_gens(self):
117         Sn_gen1 = self([tuple(range(2, self.n + 1)) + (1,)])
118         Sn_gen2 = self([(1, 2)])
119         return Sn_gen1, Sn_gen2
120
```

```
121     def __call__(self, cycles):
122         """
123         Används för att skapa permutation från deras cykler, t ex
124         S6 = SymmetricGroup(6)
125         S6([(2, 3), (5, 6)])  # returnerar (23)(56)
126         """
127         perm_lst = [i + 1 for i in range(self.n)]
128         for cycle in cycles:
129             for i in range(len(cycle) - 1):
130                 perm_lst[cycle[i] - 1] = cycle[i + 1]
131             perm_lst[cycle[-1] - 1] = cycle[0]
132         return Permutation(perm_lst)
133 
134     def _generate_elements(self):
135         perms = permutations(range(1, self.n + 1))
136         return [Permutation(perm) for perm in perms]
137 
138     def __getitem__(self, k):
139         return self.elements[k]
140 
141 
142 class PermutationGroup:
143     """
144     Klass för permutationsgrupper.
145 
146     En instans skapas genom att ange en uppsättning generatorer
147     tillhörande någon SymmetricGroup(n), t ex
148 
149     PermutationGroup(g1, g2, g3)
150     """
151 
152     def __init__(self, *gens):
153         self.n = gens[0].n
154         self.gens = gens
155         self.identity = Permutation.identity(self.n)
156         self.elements = self._generate_elements()
157         self.order = len(self.elements)
158         self.parent = SymmetricGroup(self.n)
159 
160     def _generate_elements(self):
161         "Genererar alla element i gruppen."
162         M = set()
163         queue = [self.identity]
```

```
164             while queue:
165                 current = queue.pop()
166                 M.add(current)
167                 for gen in self.gens:
168                     new_perm = current * gen
169                     if new_perm not in M:
170                         queue.append(new_perm)
171             return M
172 
173         def __repr__(self):
174             return (f'A permutation group of order {self.order} in '
175                     f'SymmetricGroup({self.n})')
176 
177         def __iter__(self):
178             return iter(self.elements)
179 
180         def __hash__(self):
181             return hash(self.gens)
182 
183         def __contains__(self, x):
184             return x in self.elements
185 
186         def __eq__(self, other):
187             "A == B om grupperna är lika."
188             return self <= other and self.order == other.order
189 
190         def __le__(self, other):
191             "A <= B om A är en delgrupp till B."
192             for gen in self.gens:
193                 if gen not in other.elements:
194                     return False
195             return True
196 
197         def __lt__(self, other):
198             "A < B om A är en äkta delgrupp till B."
199             return self <= other and self.order < other.order
200 
201         @cache
202         def orbits(self):
203             f = {x: x for x in range(1, self.n + 1)}
204 
205             for gen in self.gens:
206                 for x in range(1, self.n + 1):
```

```
                y = gen(x)
                if f[x] != f[y]:
                    old_value = f[y]
                    new_value = f[x]
                    for z in range(1, self.n + 1):
                        if f[z] == old_value:
                            f[z] = new_value
        orbits = []
        visited = set()
        for x, f_x in f.items():
            if f_x not in visited:
                orbit = {m for m, f_m in f.items() if f_m == f_x}
                orbits.append(orbit)
                visited.add(f_x)
        return orbits

    def is_transitive(self):
        return len(self.orbits()) == 1

    def is_abelian(self):
        for i in self.gens:
            for j in self.gens:
                if not i * j == j * i:
                    return False
        return True

    def is_normal(self):
        for s_gen in self.parent.gens:
            for gen in self.gens:
                if s_gen * gen * (~s_gen) not in self.elements:
                    return False
        return True

    def is_conjugate(self, other):
        if self.order != other.order:
            return False
        for g in self.parent:
            g_inv = ~g
            if all(g * h * g_inv in other for h in self.gens):
                return True
        return False

    def find_blocks(self, omega):
```

```
250         """
251         Hittar alla block som innehåller element 1 och angivet omega.
252         Returnerar en lista av block som uppfyller kriteriet.
253         Algoritm baserad på M. D. Atkinsons artikel:
254         An Algorithm for Finding the Blocks of a Permutation Group,
255         Mathematics of Computation,Vol. 29, No. 131, (1975).
256         """
257         if omega != 1:
258             C = set()  # Steg 1
259             f = {x: x for x in range(1, self.n + 1)}
260             C.add(omega)
261             f[omega] = 1  # Steg 2
262             while C:  # Steg 10
263                 beta = C.pop()
264                 alpha = f[beta]  # Steg 3
265                 for gen in self.gens:  # Steg 4, 5 & 9
266                     gamma = gen(alpha)
267                     delta = gen(beta)
268                     if f[gamma] != f[delta]:  # Steg 6
269                         if f[gamma] < f[delta]:
270                             gamma, delta = delta, gamma  # Steg 7
271                         new_value = f[delta]
272                         old_value = f[gamma]
273                         for key, value in f.items():
274                             if value == old_value:
275                                 f[key] = new_value
276                         C.add(old_value)
277         blocks = []
278         for fvalue in set(f.values()):
279             block = {x for x, value in f.items() if value == fvalue}
280             blocks.append(block)
281         return blocks
282 
283     @cache
284     def minimal_blocks(self):
285         "Hittar minimala icke-triviala block för gruppen."
286         minimal_blocks = None
287         minimal_size = self.n + 1
288         for omega in range(2, self.n + 1):
289             blocks = self.find_blocks(omega)
290             if len(blocks[0]) < minimal_size:
291                 minimal_blocks = blocks
292                 minimal_size = len(blocks[0])
```

```
293             return minimal_blocks
294
295         def is_primitive(self):
296             if not self.is_transitive():
297                 return False
298             return len(self.minimal_blocks()) == 1
```

9 Referenser

[1] Atkinson, M. D. (1975). *An Algorithm for Finding the Blocks of a Permutation Group*. Mathematics of Computation.

[2] Biggs, N. (2002). *Discrete mathematics*. Oxford University Press.

[3] Dixon, J. D., & Mortimer, B. (1996). *Permutation groups*. Springer.

[4] OEIS Foundation Inc. (2025), Number of transitive permutation groups of degree n, Entry A002106 in The On-Line Encyclopedia of Integer Sequences, `https://oeis.org/A002106`.

[5] OEIS Foundation Inc. (2025), Number of primitive permutation groups of degree n, Entry A000019 in The On-Line Encyclopedia of Integer Sequences, `https://oeis.org/A000019`

[6] Rowland, T., & Weisstein, E. W. *Group block*. MathWorld–A Wolfram Web Resource. Hämtad 28 november 2024 från `https://mathworld.wolfram.com/GroupBlock.html`

[7] Wikipedia contributors. *Kardinalitet*. Wikipedia. Hämtad 22 november 2024 från `sv.wikipedia.org/wiki/Kardinalitet`

[8] Wikipedia contributors. *Conjugacy class*. Wikipedia. Hämtad 27 november 2024 från `en.wikipedia.org/wiki/Conjugacy_class`

[9] Wikipedia contributors. *Normal delgrupp*. Wikipedia. Hämtad 28 november 2024 från `sv.wikipedia.org/wiki/Normal_delgrupp`

[10] Wikipedia contributurs. *Galoisgrupp*. Wikipedia. Hämtad 10 juni 2025 från `https://sv.wikipedia.org/wiki/Galoisgrupp`