Gjøvik, Torkildsen

Algoritmisk tenkning

I forslagene til de nye læreplanene gjør Kjerneelementutvalget et poeng ut av at algoritmer har en viktig plass i skolen, men at det også er viktig å forstå både bruken og hvorfor de fungerer. I denne sammenheng innføres begrepet algoritmisk tenking:

> Algoritmisk tenking er viktig i prosessen med å utvikle strategiar og framgangsmåtar for å løyse problem. Problemløysing i matematikk handlar om at elevane utviklar ein løysingsmetode på eit problem dei ikkje kjenner frå før. Det handlar òg om å analysere og forme om kjende og ukjende problem, løyse dei og vurdere om løysingane er gyldige. (Utdanningsdirektoratet, 2019)

Vi ønsker ikke å gjøre noe stort poeng ut av det, men det er behov for å dvele litt ved forskjellen mellom begrepene tenking, tenkning, tenkemåte og tankegang. En finner alle disse i litteraturen, og de blir nok brukt omtrent synonymt, selv om Språkrådet anfører at det både er forskjeller i ordbokoppføringene og i betydningene. *Tenkning* har vært oppfattet som mer generelt enn *tenking* (som betyr «det å tenke») og *tankegang* (som betyr «en bestemt tankerekke») (Språkrådet, 2007). I tillegg kan en se at begrepet *algoritmisk resonnering* blir brukt. Det finnes eksempler på at algoritmisk resonnering (Algorithmic Reasoning) brukes om det å *følge* en algoritme for å finne et svar (Lithner, 2008). Begrepsbruken vil nok problematiseres mer når en blir kjent med algoritmisk tenkning i skolen.

Algoritmisk tenkning kan høres ut som et gufs fra fortiden, fra den tiden da skolematematikken hadde mer fokus på algoritmiske løsninger av matematikkoppgaver. I dagens matematikkundervisning tones gjerne bruken av standardalgoritmer ned, og en verdsetter og bruker i større grad åpne oppgaver, modelleringsoppgaver og problemløsningsoppgaver der en ikke nødvendigvis på forhånd vet hvordan en skal komme fram til svarene. I tillegg er det fokus på å bruke forskjellige strategier for å løse oppgaver. Nils Johan Kjøsnes stilte for over tjue år siden relevante spørsmål rundt det å kunne divisjonsalgoritmen, som kanskje er det beste eksemplet på algoritmer som mange elever bare bruker uten å forstå (Kjøsnes, 1997).

Programmering og koding er nær knyttet til algoritmisk tenkning, og i høringsutkastet til de nye læreplanene (Utdanningsdirektoratet, 2019)

Øistein Gjøvik
ILU, NTNU
oistein.gjovik@ntnu.no

Hermund André Torkildsen
ILU, NTNU
hermund.a.torkildsen@ntnu.no

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten

finnes det eksempler på både programmering og koding som kompetansemål.

Allerede etter andre trinn skal elever jobbe med algoritmer, både ved å lage det vi kan kalle sorteringsalgoritmer, og ved å følge allerede eksisterende trinnvise instruksjoner. Her finnes det også forslag til kompetansemål der programmering nevnes eksplisitt. I tillegg dukker begrepet programmering opp flere ganger, både under relevans av matematikkfaget, digitale ferdigheter og andre fag.

I denne artikkelen skal vi se på hvordan algoritmisk tenkning kan ses i sammenheng med matematikkfaget og fagets forhold til algoritmer.

Algoritmisk tenkning

Det vil være synd om algoritmisk tenkning skal «skjemmes» av de samme konnotasjonene som standardalgoritmene i skolen etter hvert har fått. Det er også klart at det kan være forvirrende at det eksisterer en mengde mer eller mindre likelydende begreper som omhandler det å tenke om og med algoritmer. Vår oppfatning er at disse betyr mer eller mindre det samme, i noen tilfeller nøyaktig det samme, og at det neppe byr på store problemer å bruke dem om hverandre. Vi skal likevel prøve å rydde opp noe i begrepsbruken.

På engelsk brukes gjerne begrepet *Computational Thinking* (CT) (Bjørnevoll, E., 2016), og det er godt mulig denne internasjonale betegnelsen er bedre når det gjelder å fange essensen av konseptet. Det norske begrepet som ser ut til å ha vokst fram som det mest brukte, er *algoritmisk tankegang:*

> In Norway, algoritmisk tankegang (sic) (EN: algorithmic thinking) emerges as the most widely used umbrella term that includes common CT features. (Bocconi & Chioccariello, 2018, s. 8)

Her påstås det altså at algoritmisk tankegang er det samme som Algorithmic Thinking, og at dette er en undergruppe av Computational Thinking. Som vi ser, er det ikke fullstendig overensstemmelse mellom norske og utenlandske termer, og en vil også finne forskjellige definisjoner i forskjellig litteratur (Tedre & Denning, 2016). I og med at vi ønsker å fokusere på generelle måter å tenke og jobbe på i matematikkfaget, velger vi i denne artikkelen å holde på betegnelsen algoritmisk tenkning, jf. Språkrådets oppklaringer, og vi vil da legge det samme i det som i Computational Thinking.

Uavhengig av definisjon kan en finne en mengde elementer som går igjen. I rapporten «The nordic approach to introducing computational thinking and programming in compulsory education» står det:

> In general terms, computational thinking is regarded as a thought process entailed in designing solutions that can be executed by a computer, a human, or a combination of both. In spite of the wide variety of definitions in use, it is possible to identify a set of constituent core concepts recursively positioned under the CT umbrella, namely abstraction, algorithmic thinking, automation, decomposition and generalization. These in turn are related to a set of attitudes and skills (or practices), including creating computational artifacts, testing and debugging, collaboration and creativity, and the ability to deal with open-ended problems. (Bocconi & Chioccariello, 2018, s. 7)

Computational Thinking brukes på engelsk om Algorithmic Thinking samt elementer som abstraksjon og generalisering. Det må bety at når algoritmisk tenkning er innført på norsk som det samme som Computational Thinking, må det altså finnes en undergruppe av algoritmisk tenkning som tilsvarer Algorithmic Thinking. Forvirringen kan nok skyldes at algoritmisk tenkning slik vi bruker det på norsk, ikke er det samme som Algorithmic Thinking slik det brukes på engelsk. Vi har valgt å oversette

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten

Algorithmic Thinking med algoritmebehandling, da dette begrepet brukes om det å følge og forklare algoritmer.

Så hva skal vi fylle begrepet algoritmisk tenkning med? I de forskjellige definisjoner og diskusjoner kan en gjenkjenne en del sentrale begreper, men disse ser ut til å handle om de samme fenomenene. For eksempel er *pattern recognition* brukt som ett av fire sentrale elementer i BBC sin definisjon (BBC, 2019), men en kan godt tenke seg mønstergjenkjenning som del av det å drive generalisering. La oss se på de fem begrepene fra Bocconi & Chioccariello (2018):

Abstraksjon

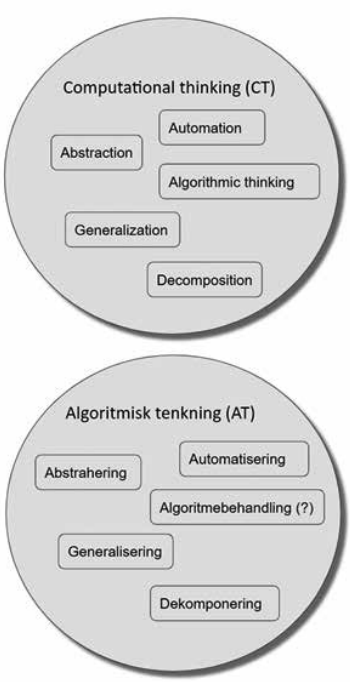
Det å kunne trekke ut (abs-tract) essensen av flere eksempler eller tilfeller. Se bort fra irrelevante opplysninger.

Algoritmebehandling

Å følge og forklare trinnvise instruksjoner. Her er det verdt å nevne (forhåpentligvis uten å lage fullstendig kaos) begrepet algoritmisk resonnering (AR) (Lithner, 2008), som blir brukt på en helt annen måte i litteraturen. AR inngår i definisjonene av kreativ og imitativ resonnering. Imitativ resonnering går ut på å resonnere enten ved løsninger eller løsningsmetoder en kjenner fra før. Dette omhandler både algoritmer som kan knyttes til det opprinnelige problemet, og algoritmer som aktiveres kun av eksternt påtrykk. Slik kan vi si at algoritmisk resonnering er beslektet med (en undergruppe av) algoritmebehandling. Vi har altså valgt å kalle dette algoritmebehandling.

Generalisering

Å gjenkjenne mønstre og sammenhenger og lage allmenne regler og metoder som fungerer på en hel klasse av eksempler.



Figur 1: Computational thinking og algoritmisk tenkning.

Automatisering

Det å kunne implementere løsningen av problemer i programmeringsspråk eller å gjøre det menneskelige bidraget minimalt.

Dekomponering

Det å kunne bryte opp et problem i mindre bestanddeler og håndtere hovedproblemet i mindre biter.

Denne definisjonen skiller seg altså fra BBC sin definisjon, som inneholder Decomposition, Abstraction, Pattern Recognition og Algorithms – selv om mange av bestanddelene inneholder det samme.

Framgangsmåten til en som skal løse et problem ved hjelp av programmering, er påfallende lik de framgangsmåter som brukes for å løse problemer i matematikk. Det er ikke overras-

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten

kende at begrepene i figur 1 passer godt i både programmering og matematikk. I forbindelse med fagfornyelsen i matematikk har programmering stått sentralt i diskusjonene, der både teknologipessimister og teknologioptimister har opptrådt med markante stemmer. De vedtatte kjerneelementene er

- utforsking og problemløsing
- modellering og anvendelser
- resonnering og argumentasjon
- representasjon og kommunikasjon
- abstraksjon og generalisering

På alle disse fem områdene (det sjette er «matematiske kunnskapsområder») kan programmering være en framtidsrettet og relevant måte å arbeide med matematikken på. Vi gjenkjenner også flere av grunnelementene i algoritmisk tenkning i kjerneelementene.

Programmering, koding og algoritmisk tenkning

Et naturlig miljø for å implementere algoritmisk tenkning er via programmering. Da kan det være nyttig å rydde av veien forskjellen på programmering og koding. Disse to begrepene har ofte blitt brukt som om de hadde omtrent samme begrepsinnhold. Likevel finnes det forskjellige tradisjoner og måter å bruke disse begrepene på. En vanlig distinksjon er å bruke programmering om alt det som dreier seg om å løse et problem ved hjelp av en algoritme eller et program. Det vil si forarbeid, planlegge, skrive, tegne, bearbeide, undersøke om problemet er løst, eksperimentere, revurdere, argumentere for at algoritmen alltid virker, generalisere algoritmen til å løse en hel klasse av problemer, sjekke effektivitet osv. Programmering kan sies å være å kommunisere med logikk. Det kan vi også si er en del av matematikken. Koding, på den andre siden, er selve aktiviteten med å reformulere dette inn i et bestemt programmeringsspråk. Koding er altså mer snevert enn programmering.

Vi kan finne mye litteratur og forskning om læring av matematikk gjennom programmering. Programmeringsspråket LOGO var et radikalt innslag i matematikkundervisningen da det ble lansert for ca. 40 år siden, og et viktig poeng med LOGO var at kreativitet, utforsking, eksperimentering og åpnere tilgang til matematikken fikk større plass i skolematematikken. Det finnes mengder av forskning som viser positive resultater av å bruke LOGO for å lære matematikk, en oversikt kan vi finne i «What the research says about ICT in maths» (BECTA, 2003). LOGO lever fortsatt i flere varianter. For eksempel finnes LOGO i Bee-Bots (en skilpaddevariant for barnehagebarn som har fokus på å utvikle romforståelse og problemløsningsstrategier), LEGO mindstorms og i Scratch (Massachusetts institute of technology, 2019).

Det kan virke merkelig både at programmering forsvant fra matematikkfaget, og at det er mye motstand mot å innføre det på nytt til tross for gode resultater fra forskningen. En kan nok se for seg mange problemer som kan oppstå når en skal introdusere programmering og algoritmisk tenkning i matematikkundervisningen. Mange av disse har blitt målbåret i debatten omkring innføringen. Hvordan vil det gå med de elevene som ikke får til abstrahering, de som ikke behersker symbolspråket? Noen mener de vil bli motivert av å få en mening med det å bruke et notasjonssystem, andre mener det vil bli vanskeligere for disse å sette seg inn i enda et notasjonssystem, med bare til dels like egenskaper (f.eks. kan likhetstegnet i programmering bety «blir», mens det i matematikken betyr «er likt med»). Den vanlige misoppfatningen mange elever har om likhetstegnet, kan vi kanskje nå oppleve blir enda verre. Men en trenger likevel ikke tenke seg at programmering nødvendigvis skal skje på en datamaskin med et nytt språk eller notasjonssystem. En kan jobbe med programmering og algoritmisk tenkning på mange nivåer og klassetrinn. Vi skal se på noen eksempler uten datamaskin og hvordan

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten

disse forholder seg til algoritmisk tenkning og matematikkfaget.

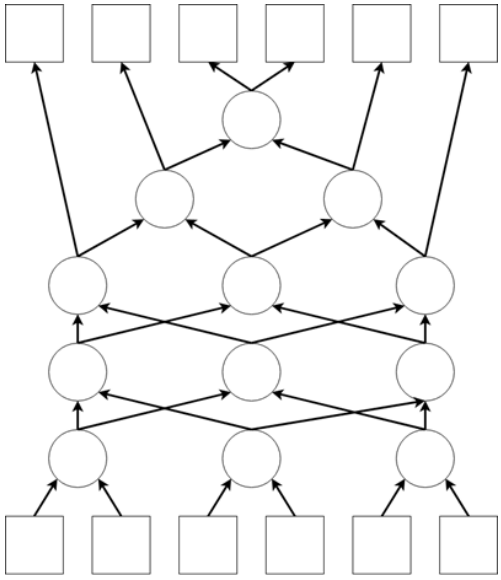
Eksempel: Sorteringsalgoritmer

I matematikk stilles det krav til presisjon, og det er viktig å være eksakt og presis også i arbeidet med å utforme algoritmer for løsning av problemer. Det kan sammenlignes med en kjent aktivitet fra barnetrinnet, geometristafetter (Botten, 2003). Her skal elever tegne en geometrisk figur, for så å beskrive denne til en annen elev kun ved hjelp av muntlige instruksjoner. På sett og vis er også dette kommunikasjon med logikk, altså godt plassert i både matematikk og programmering. For å understreke at algoritmisk tenkning ikke nødvendigvis forutsetter utstrakt bruk av digitale verktøy, kan en se på nettsidene til prosjektet CS unplugged (UC Computer science education, Google, & Microsoft, 2019). Her finnes det en stor mengde ressurser for å jobbe med sentrale aspekter innen problemløsning – aspekter som en senere kan behandle med digitale verktøy og programmering, men som elever helt ned i førskolealder kan jobbe med. Her er et eksempel som handler om sortering.

La oss si vi ønsker å sortere en liste av tall i stigende rekkefølge. En typisk, og naiv, måte å løse problemet på er å søke gjennom listen etter det minste tallet og flytte det til starten av listen. Deretter fortsetter en å gjøre det samme helt til listen er sortert, altså til en ikke trenger å gjøre flere flyttinger.

En lignende, men forbedret metode er det vi kan kalle innstikksortering (insertion sort). Dette er en måte som mange bruker når de skal sortere en kortstokk. En søker gjennom listen helt til en finner et element som ikke er på rett plass. Da tar en elementet ut av listen, starter på begynnelsen av listen og finner plassen som det skal være.

Boblesortering (bubble sort) er en litt annerledes metode, der en starter på begynnelsen av listen og sammenligner to og to elementer som er ved siden av hverandre. Hvis det første elementet er større enn det andre, bytter de helt enkelt plass. Når en har gått gjennom listen en gang, starter en på begynnelsen igjen og fortsetter slik til alle elementene er i riktig rekkefølge.



Figur 2: Sorteringsnettverk

Det finnes mange ulike sorteringsalgoritmer, med varierende grad av kompleksitet og effektivitet. Boblesortering er en enkel, men lite effektiv algoritme, og den brukes sjelden av programmerere. Tenk på hvor mange flyttinger en må gjøre hvis listen som skal sorteres, er i synkende rekkefølge. Likevel – hvis listen som skal sorteres, allerede er nesten sortert, er den faktisk ganske effektiv.

Elever kan selv utvikle slike metoder for å sortere. De kan argumentere for at metoden fungerer, generalisere metoden til å fungere på en hel klasse av eksempler og vurdere effektivitet.

En annen måte å arbeide med sorteringsalgoritmer på er det vi kan kalle sorteringsnettverk. Figur 2 viser et slikt nettverk. Nederst er listen av elementer som skal sorteres. Hver sirkel (komparator, fra det engelske ordet *compare*) i figuren har to inn-verdier og to ut-verdier. Når to elementer kommer inn i komparatoren, så

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten

sammenlignes de. Det minste elementet følger pila til venstre, og det største elementet går til høyre (gitt at vi skal sortere i stigende rekkefølge). Når tallene har gått gjennom hele nettverket, resulterer dette i en sortert liste.

En kan lage flere ulike sorteringsnettverk, og mange vil ikke alltid resultere i en sortert liste. Elever kan lage egne nettverk og prøve å argumentere for at nettverket alltid vil sortere riktig. En kan også snu pilene i nettverket og finne ut av om nettverket da vil sortere korrekt. En kan finne eksempler på at det ikke alltid går.

Denne kan en også fint jobbe med i papirformat og med konkretiseringshjelpemidler som ikke er digitale. Allerede etter andre trinn finner vi i forslagene til nye læreplaner at elevene skal kunne sortere størrelser og diskutere om det kan gjøres på flere måter, samt lage og følge regler og trinnvise instruksjoner. CS Unplugged har flere videoer av elever som arbeider med sorteringsnettverk, der blant annet de yngste fysisk går gjennom nettverket.

I en slik «analog programmeringsøkt» vil en bruke mange av komponentene fra algoritmisk tenkning.

Abstrahering – En lager en abstrakt representasjon av noe fysisk, det å sortere. Sorteringsnettverk kan brukes på alle slags data som kan sammenlignes (ikke bare tall). En trenger ikke vite hva slags data det er, bare at elementene kan sammenlignes og hvordan.

Algoritmebehandling – Her kan en både lage og sammenligne algoritmer, også om den ene er mer effektiv enn den andre.

Generalisering – Elever kan se at en slik sorteringsalgoritme kan brukes på datamengder av en vilkårlig størrelse, og uansett hvordan datamengden er usortert til å begynne med.

Automatisering – Om en vil lage en oppskrift som kan følges av en datamaskin, er ikke veien så lang til å jobbe med sortering i for eksempel Scratch.

Dekomponering – I stedet for å sortere en hel liste er problemet dekomponert i små deloppgaver (sammenligne to og to).

Oppsummering

Programmering er antatt å få en såpass stor rolle for kommende generasjoner at en også må regne med at det vil ta en del plass i skolen. Noen land har allerede kommet et stykke på vei i implementeringen av programmering. I England har en introdusert programmering ved ScratchMaths-prosjektet (Benton, Saunders, Kalas, Hoyles & Noss, 2018), og det har også blitt brukt for læring av matematikk (Benton, Hoyles, Kalas & Noss, 2017). En annen innfallsvinkel er micro:bit. Dette er programmering på en datamaskin i lommeformat, tatt i bruk i for eksempel Danmark og England (se oversikt over forskning rundt micro:bit i skolen så langt på https://microbit.org/research/).

Nå når programmering skal inn i matematikkfaget, er det viktig at det blir gjort på en god måte. Programmering må være noe mer enn å implementere allerede eksisterende algoritmer. Det må handle om problemløsning og å utvikle metoder/algoritmer for å løse en hel klasse av problemer. Prosessen mot en generell metode må være det viktigste, og her er kanskje matematikkfaget en naturlig arena, til og med om en ikke bruker digitale verktøy i arbeidet. Fokuset må heller ikke være å lære seg et programmeringsspråk og syntaks. Det kommer stadig nye programmer og programmeringsspråk. Inngangsterskelen må være lav, slik at teknologien ikke kommer i veien for læring. Målet bør være læring av matematikk med programmering, ikke bare å lære koding. Her kan algoritmisk tenkning hjelpe oss, men da må en nok både rydde opp i begrepene rundt algoritmisk tenkning og på nytt ta tak i de uklarhetene som oppstod i skjæringspunktet mellom læring og programmering på 1980-tallet.

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten

Referanser

BBC (2019). BBC Bitesize - KS3 Computer Science - Introduction to computational thinking. Hentet 15. februar 2019 fra https://www.bbc.com/bitesize/guides/zp92mp3/revision/1

BECTA (2003). *What the research says about ICT in maths*. Hentet fra http://www.education.gov.uk/publications/eOrderingDownload/15014MIG2799.pdf

Benton, L., Hoyles, C., Kalas, I. & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education, 3*(2), 115–138.

Benton, L., Saunders, P., Kalas, I., Hoyles, C. & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction, 16*, 68–76.

Bocconi, S. & Chioccariello, A. (2018). *The Nordic approach to introducing computational thinking and programming in compulsory education*. Hentet fra https://www.itd.cnr.it/doc/CompuThinkNordic.pdf

Enge, O. & Buan, A. B. (2004). Kryptografi i skolen. *Tangenten – tidsskrift for matematikkundervisning, 15*(3), 8–11.

Lampert, M. (1990). When the problem is not the question and the solution is not the answer: Mathematical knowing and teaching. *American Educational Research Journal, 27*(1), 29–63.

Lithner, J. (2008). A research framework for creative and imitative reasoning. *Educational Studies in Mathematics, 67*(3), 255–276.

Massachusetts institute of technology (2019). Scratch – Imagine, Program, Share. Hentet 28. mars 2019 fra https://scratch.mit.edu/

Språkrådet (2007). Spørsmål og svar. *Språkrådet, 35*(2), 32–33.

Tedre, M. & Denning, P. J. (2016). The long quest for computational thinking. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research – Koli Calling '16*, 120–129. https://doi.org/10.1145/2999541.2999542

UC Computer science education, Google, & Microsoft (2019). CS Unplugged. Hentet 24. mai 2018 fra CS Unplugged website: https://csunplugged.org/en/

Utdanningsdirektoratet (2019). *Læreplan i matematikk fellesfag 1.–10. trinn*. Hentet fra https://hoering.udir.no/Hoering/v2/343?notatId=686

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten – tidsskrift for matematikkundervisning, 30*(3). 31–37.

www.caspar.no/tangenten