Grafteori	Niveau 3	LÆRER

Den korteste vej, effektivt

Det genererende spørgsmål

For en udbyder af en 'korteste-vej' app er Dijkstras algoritme ikke blot et grafteoretisk metode, men også underlagt praktiske overvejelser. Kan vi love 'kunderne' at Dijkstras algoritme virker? Og er den effektiv?

Dette giver anledning til spørgsmålene:

- Hvordan garanteres, at Dijkstras algoritmen finder de korteste afstande fra en knude i en graf?
- Kan den korteste vej findes på rimelig computertid?

Forudsætninger

Fortrolighed med Dijkstras algoritme, svarende til Niveau 2, herunder fx Maple-pakken 'Graph Theory' ().

Tilstræbte læringsmål

At redegøre detaljeret for en konkret algoritme (Dijkstras). At redegøre for, at resultatet af en algoritmesløjfe kan eftervises ved induktion i konkret tilfælde. Tilegnelse af læsningsstrategi af matematiktekster. Vurdering af størrelsesordener (v. andengradspolynomium).

Forslag til hjælpemidler

Papir og blyant til etiketter mm. i rollespil. Kridt og tavle. Software fx Maple-pakken 'Graph Theory' ().

Materiale til eleverne

På ppt-slides samt algoritmen, rutediagram og graf til rollespil som pdf-filer. Desuden udleveres en pdf-fil med beviset for Dijkstras algoritme.

Fase 1: Beskrivelser af vægtede grafer

For at kunne omtale trinene i Dijkstras algoritme præcist er det nødvendigt med en række betegnelser, notationer og definitioner. Det er altid en afvejning, hvor eksplicit og omfattende man vil være. Overdreven notation gør det meget tungt at følge argumenter. På den anden side skal man være helt sikker på, hvad man taler om. Et særkende for matematisk argumentation er, at den undgår at blive afsporet af en diskussion, "hvad snakker vi om?". Vi laver en definition!

I fase 1 skal eleverne gennemlæse algoritmen og nedskrive, hvilke definitioner, notationer m.m. der er behov for. Dette vil indbefatte vægtet graf (G, w), kanter (xy), knuder $V(G)$, sti mellem to knuder, længde af sti, afstand mellem to knuder $(\mathrm{d}(x, y))$, kantforbundet. (I parenteserne står noget notation, der kan komme på tale. Men det vil være en god diskussion med eleverne at forholde sig til 'god notation'). De fastlægges præcist efterhånden, som behovet opstår. Det kan fx være at 'kantforbundet' er klart for alle og ikke behøver at fastlægges i en formel definition, mens afstand mellem to knuder kræver en præcisering i retningen af: Afstanden fra knuden u til knuden z er den mindste længde af en sti, som forbinder u og z. Vi bruger notationen $\mathrm{d}(u, z)$.

Efterhånden som man arbejder sig gennem projektet, kan der blive behov for at vende tilbage til denne fase.

Eleverne arbejder i grupper og læreren holder regnskab med den fælles beskrivelse (så der er enighed om, hvad tingene hedder, og hvordan de benævnes).

Fase 2: Rollespil med algoritmen

For at opnå fortrolighed med algoritmen gennemføres den som rollespil på en konkret graf (slide, uddelt pdf-ark).

Roller: Opdatering af de foreløbige afstande for 5 punkter. Opdatering af knudemængden S. Tildeling af endelige afstande. Bestemmelse af mindste værdi af $t(z)$. Overordnet algoritmestyring. … Og måske nogle

flere. Det hører med til opnåelse af det tilsigtede læringsmål, indsigt i algoritmen, at man prøver sig lidt frem. Det er ligeledes en pointe, at rolleindehaverne forbliver i deres rolle, dvs. blot mekanisk udfører, hvad rollen indebærer. Herved skulle det blive klart, at algoritmen nedbryder den overordnede opgave 'find korteste vej' til simple instruktioner (sammenligninger, tildelinger, additioner osv.), som kan udføres ufejlbarligt, men som tilsammen løser den komplicerede opgave.

Rollelisten skrives på tavlen.

Opstår der tvivl undervejs i rollespillet, noteres tvivlen, så man kan vende tilbage. Men det er vigtigt, at rollespillets 'flow' fastholdes.

Rollespillet evalueres: Blev resultatet korrekt? Manglede der roller? osv.

Efterfølgende problem:

Den viste version af Dijkstras algoritme giver kun afstande, men ikke selve de korteste stier. Hvilke yderligere instruktioner (sammenligninger, tildelinger, additioner osv.) skal tilføjes for, at algoritmen også kan angive en korteste sti?

Svar: Hvis etiketten $t(z)$ ændres i en opdatering, noteres også det sidst tilføjede punkt. Da etiketten $t(z)$ kun opdateres, indtil z tilføjes, vil der på etiketten til sidst stå forgængeren til z. De korteste stier kan nu findes ved successivt at opsøge forgængere. Dette kan evt. prøves ved et nyt rollespil.

Fase 3: Bevis for algoritmen

Bevis for algoritmen vil være en relativt avanceret læseøvelse for eleverne. Projektets fokus bliver læsestrategier og træning i at komme videre, når man går i stå. Læringsmålet for læseøvelsen er, at eleverne kan redegøre for, hvor i teksten de kan følge argumentet fuldt – og vigtigst, hvad mangler der (hvis noget) for en fuld tilegnelse af sætningen og dens bevis. En god læsestrategi er 'stop and go'. Man læser ord for ord, og spørger sig selv: 'Er jeg med hertil?' indtil første gang 'kæden hopper af'. Hvis problemet kan løses (fx man finder en begrundelse for det pågældende trin) noteres løsningen, og man fortsætter til næste stop. Hvis ikke, noterer man, at der er en uafklarethed, og hvori den består, hvorefter man fortsætter til næste stop. Ofte vil teksten give mening, hvis man tager det uafklarede til efterretning [1]. Når teksten er læst til ende første gang, gentages dette, men med nogle overskrifter: Giver sætningens udsagn mening? Hvad er den overordnede struktur? Hvad er idéen i beviset? …? Nogle afgørende punkter i dette er:

Sætningens scenario: Utvetydig beskrivelse af de objekter og definitioner, som fastlægger, hvad sætningen handler om.

Forudsætninger, som vi antager om scenariet.

Den **ønskede konklusion**, hvis antagelserne er opfyldte.

Bevis: **Logisk deduktion** *forudsætninger* ⇒ *ønsket konklusion*

De punkter hvor 'kæden hopper af' kan antage forskellige varianter: 'Er det noget, som jeg blot skal genopfriske?', 'Hvor kan jeg finde den indsigt, som jeg mangler?' ('spørg læreren' er jo oplagt, men læreren skal være påholdende – og kender måske ikke svaret!), 'Er der tale om et 'trick' – en snedighed?'. Der er i sagens natur ikke en færdig klassifikation af, hvilke forhindringer man kan støde på. Det vigtige er den skærpede opmærksomhed om 'matematikteksten', så eleven ikke ender i en ureflekteret afvisning.

Således fortsættes, indtil man med 'god samvittighed' kan sige, at man ikke kan komme videre og kan redegøre for, hvad man mangler (hvis noget). Det er vigtigt at pointere over for eleverne, at denne afklaringsproces er betydningsfuld i sig selv. At kunne forklare, at sætningen '*Dijkstras algoritme giver korteste veje fra et knude, u i en vægtet graf til grafens øvrige knuder*' kan bevises ved det anførte induktionsbevis, er vigtigt – også selvom man ikke er med på alle detaljerne i beviset. Læsestrategien

[1] Fx: Jeg kan ikke lige se hvorfor løsningen til ligningen bliver $x = \cdots$, men accepterer det og kan godt læse meningsfuldt videre.

handler om *'ikke at løbe panden mod muren'.* Det ligner i øvrigt den strategi, som en forsker vil anvende for at sætte sig ind i ny matematik.

'Stop and go' læsning kan gennemføres individuelt og med en 'læsemakker'. Det kan være meget udbytterigt først at starte med den individuelle læsning og derefter søge yderligere afklaring med en makker.

Detaljeret om beviset:

Beviset er et induktionsbevis efter $k = |S|$, hvor S er mængden af knuder, hvis afstand til u algoritmen allerede har givet et bud på. Vi vil vise, at algoritmens bud er korrekt. Erfaring har vist, at det er en god idé at arbejde med ***to*** induktionspåstande på én gang:

i) For $z \in S$ er $t(z) = \mathrm{d}(u, z)$
ii) For $z \notin S$ er $t(z) =$ længden af korteste sti fra u til z direkte fra S. (Hvis z ikke er kantforbundet til S (altså ingen *direkte* sti fra S), menes hermed at $t(z) = +\infty$.)

Induktionens start, $k = 1$

Vi er i situationen $S = \{u\}$. Algoritmen har sat $t(u) = 0$, så $t(u) = \mathrm{d}(u, u)$ er oplagt, dvs. i) holder for $k = 1$. Da $t(z) = w(uz)$ for $z \neq u$, er ii) ligeledes opfyldt for $k = 1$. Afstanden direkte fra S er jo lig med kantlængden, når z er kantforbundet til u og ellers $+\infty$.

Induktionsskridtet

Eleverne opfordres til at lave figurer, der illustrerer deres argumenter

Antagelsen er, at vi har gennemført algoritmesløjfen k gange. Hvis $k = |V(G)|$, er vi færdige. Der er ikke flere knuder, og i) siger netop, at vi har bestemt afstanden fra u til ethvert af grafens knuder.

Ellers vælger algoritmen jo et knude $v \notin S$, så $t(v)$ er mindst mulig. Vi sætter så $S' = S \cup \{v\}$. Induktionsantagelsen er, at i) og ii) gælder for S, og induktionsskridtet betyder, at vi skal vise at i) og ii) også gælder for S'. Først viser vi i), dvs. vi skal vise, at $t(v) = \mathrm{d}(u, v)$. Vi må bruge, at både i) og ii) antages at gælde for S. Induktionsantagelsen ii) siger, at $t(v)$ er længden af den korteste sti til v direkte gennem S. Da $\mathrm{d}(u, v)$ er længden af den korteste sti overhovedet, må der gælde at $t(v) \geq \mathrm{d}(u, v)$. Hvis vi kan vise den modsatte ulighed $t(v) \leq \mathrm{d}(u, v)$, har vi vist $t(v) = \mathrm{d}(u, v)$. Vi tager nu en korteste sti fra u til v. Per definition er længden af denne sti $\mathrm{d}(u, v)$. Endvidere gælder, at for ethvert knude undervejs er denne sti også en korteste sti til det pågældende knude (Dijkstras idé). Stien starter i u og altså i S. Den ender i v, altså uden for S. Lad v' være stiens første knude uden for S. Algoritmen har valgt v, så $t(v) \leq t(v')$. Den korteste vej overhovedet til v' er direkte gennem S. Det var sådan, vi valgte v'. Induktionsantagelsen siger hermed, at $t(v') = \mathrm{d}(u, v')$. Og der gælder, at $d(u, v') \leq d(u, v)$ for v' er et knude undervejs på den korteste sti fra u til v. Alt i alt,

$$t(v) \leq t(v') = d(u, v') \leq d(u, v)$$

som læst fra venstre til højre er den ulighed, vi var på jagt efter.

Vi viser nu ii) for S'. Hvis $S' = V(G)$, er vi færdige. Ellers gælder for $z \notin S'$, at algoritmetrinnet *opdaterer* $t(z)$ til

$$t(z) = \min \begin{cases} \text{korteste afstand direkte fra } S \\ w(zv) + t(v) \end{cases}$$

Den første mulighed er værdien af $t(z)$ før algoritmetrinnet, den anden er længden af den korteste sti til z via det nye punkt v, fordi vi viste under i), at $t(v) = \mathrm{d}(u, v)$. Alt i alt: $t(z)$ opdateres til den korteste afstand til z direkte fra S'.

Konklusion: Hvis i) og ii) gælder for S, så gælder i) og ii) også for $S' = S \cup \{v\}$, så vi ender med at i) og ii) gælder for hele $V(G)$, og at Dijkstras algoritme faktisk finder de korteste stier fra u.

Den største vanskelighed med forståelse af induktionsprincippet er antageligt induktionsskridtet. Dette går ud på at vise, at bestemte *implikationer* er gyldige, $p(1) \Rightarrow p(2), p(2) \Rightarrow p(3), \cdots,$ eller kort $\forall\, n\colon p(n) \Rightarrow$

$p(n+1)$. En almindelig misforståelse er, at man skal vise $(\forall\, n\colon p(n)) \Rightarrow\ (\forall\, n\colon p(n+1))$, hvilket jo er en selvfølgelighed, så eleven synes, at induktion er en tomgangsøvelse: Vi har øjensynligt allerede antaget, at 'formlen gælder'. Men induktionstrinnet går ud på at **bevise implikationen, ikke** at $p(n)$ er sand. Dette følger af induktionsprincippet, som er en egenskab ved de naturlige tal.

En algoritmes iterationer kan være en god måde at sætte dette på plads, da successive gennemløb af løkken er et håndgribeligt induktionsskridt, men jo ingen garanti i sig selv for, at algoritmen udfører, hvad der påstås.

Fase 4: Algoritmens effektivitet

Vi må formode, at Google bruger de hurtigste computere. De hurtigste computere kan typisk udføre 10^{11} instruktioner per sekund. (Ja, det er rigtigt nok ☺)

Lad os prøve at give et groft skøn af antallet af Dijkstras instruktioner for en graf med n knuder.

Starten er n tildelinger.

Algoritmetrinnet gennemløbes $n-1$ gange. I hvert trin foretages højst:

- $n-2$ sammenligninger for at finde den mindste værdi af $t(z)$.
- 1 tildeling (tilføjelse af det nye knude til S)
- $n-1$ additioner (opdatering af $t(z)$)
- $n-1$ minimumsbestemmelser af 2 tal (opdatering af $t(z)$)

Vi slutter med n tildelinger af afstande.

Dette giver alt i alt $n + (n-1)\big(3(n-1)\big) + n = \ 3n^2 - 4n + 3$. Hvis man også skal holde regnskab med selve Dijkstra-stierne, fås at algoritmens kompleksitet er højst $5n^2 - 8n + 5$. Vi kunne godt have været mindre grove i vores skøn, men under alle omstændigheder bliver der tale om et andengradspolynomium.

I USA er der $6{,}4\ 10^6$ km offentlig vej. Lad os sige, at der er 10 afstandspunkter pr. km, dvs. vi skal udføre Dijkstras algoritme på en graf med 64 mill. knuder. Dette tager af størrelsesorden[2] $(5\frac{(64\ 10^6)^2}{10^{11}})$ sekunder eller ca. 60 timer. Men det er jo også en hel urealistisk opgave.

I Danmark er det 1,68 km offentlig vej per km^2. Lav et estimat af, hvor hurtigt Dijkstras algoritme implementeret i Googles computer beregner korteste afstande på et område af 10 km x 10 km. (Svar: størrelsesorden mikrosekunder).

[2] Førsteordensleddet og konstantleddet er ligegyldige. Indses fx ved komplettering af kvadratet: $5\left(n-\frac{4}{5}\right)^2$, altså et knude fra eller til ud af ca. 64000000 knuder.