TDT4120 Algoritmer og datastrukturer

Eksamen, 11. august 2021, 15:00–19:00

Faglig kontakt	Magnus Lie Hetland
Hjelpemiddelkode	A

Løsningsforslag

Løsningsforslagene i rødt nedenfor er *eksempler* på svar som vil gi uttelling. Det vil ofte være helt akseptabelt med mange andre, beslektede svar, spesielt der det bes om en forklaring eller lignende. Om du svarte noe litt annet, betyr ikke det nødvendigvis at du svarte feil!

På grunn av koronapandemien er dette en hjemmeeksamen med alle hjelpemidler tillatt og med karakteruttrykk *bestått / ikke bestått*. Derfor er det ikke lagt vekt på å skille mellom prestasjoner i det øvre sjiktet av karakterskalaen, og utvalget av oppgavetyper er noe utenom det vanlige. Følgelig bør ikke eksamenssettet ses som representativt for ordinære eksamener.

1 Hva er traversseringstrær og hvordan kan man finne dem? Forklar kort, med egne ord.

Et traverseringstre er et tre over nodene i en (ev. rettet) graf som kan nås fra en gitt startnode, som blir roten i treet. Det vil si, der det finnes en (rettet) sti fra startnoden s til en annen node v i grafen, så finnes det også en slik sti i traverseringstreet. Et slikt tre finner man ved hjelp av en traverseringsalgoritme som for eksempel DFS eller BFS.

2 Dijkstra, Bellman-Ford og Dag-Shortest-Paths brukes alle for å finne korteste veier fra én til alle, i rettede vektede grafer. Forklar kort hvilke krav hver av algoritmene stiller for at den skal kunne finne rett svar.

Dijkstra kan brukes når vi har positive (ikke-negative) vekter.

Bellman-Ford kan brukes når vi ikke har negative sykler som kan nås fra startnoden. (Den kan også *brukes* når vi har slike negative sykler, men vil da ikke finne rett svar, men gi beskjed om at vi har en negativ sykel.)

Dag-Shortest-Paths kan brukes når grafen ikke har (rettede) sykler i det hele tatt.

3 Hvorfor er ikke kjøretiden til BUCKET-SORT lineær i verste tilfelle? Forklar kort.

Fordi man antar en uniform fordeling, men siden input er tilfeldig, kan man ende opp med svært ujevnt fordelte tall, slik at alle havner i samme bøtte, som så sorteres med en annen sorteringsalgoritme, og denne har normalt ikke lineær kjøretid i seg selv. (I pensumvarianten brukes INSERTION-SORT.)

4 Hva er memoisering, og hvorfor bruker vi det? Forklar med egne ord.

Det er en teknikk som ofte ses i sammenheng med – eller som en form for – dynamisk programmering. Mer spesifikt er det en form for mellomlagring/caching av returverdier i funksjoner, slik at om funksjonen kalles med samme argumenter flere ganger, så beregnes returverdien bare den første gangen, og hentes ut av lageret (memoen) senere. Dette kan forhindre dype rekursjonstrær hvis vi har overlappende delinstanser, og kan redusere kjøretiden fra eksponentiell til polynomisk.

5 Beskriv med egne ord de to mest sentrale måtene å implementere grafer på, som omhandlet i pensum. Diskuter kort styrker og svakheter ved begge to.

Dette er nabomatriser (*adjacency matrices*) og nabolister (*adjacency lists*). En nabomatrise er en binærmatrise A, der a_{ij} angir om i og j er naboer (ev. om det går en kant fra i til j, om vi har retning). Man kan også bruke vektmatriser W som en form for nabomatrise, der $w_{ij} < \infty$ hvis det går en kant fra i til j. Nabomatriser er gjerne egnet for tette (*dense*) grafer; for spinkle (*sparse*) grafer bruker vi ofte nabolister, der hver node assosieres med en liste av sine (ut-)naboer (ev. sammen med ekstra informasjon som kantvekter). Disse er egnet til traversering (der vi itererer over naboer), men mindre egnet til direkte oppslag for å finne en spesifikk kant (i, j), f.eks., siden man da må traversere alle ut-naboene til i; der er nabomatriser mer egnet, siden man bare kan slå opp a_{ij} direkte.

Merk at dette bare er et eksempel. Det er naturligvis mange måter å fremstille dette på. Det sentrale er at man får med, i grove trekk, hvordan de to metodene fungerer, og noe om styrker og svakheter.

6 Løs følgende rekurrens eksakt, med iterasjonsmetoden:

$$\begin{aligned} \mathrm{T}(0) &= 0 \\ \mathrm{T}(n) &= \mathrm{T}(n-1) + n - 1/2 \quad (n \geqslant 1) \end{aligned}$$

Oppgi svaret uten bruk av asymptotisk notasjon. Vis fremgangsmåten din, og forklar med egne ord hvordan den fungerer.

Løsning med iterasjonsmetoden:

$$\begin{aligned} \mathrm{T}(n) &= \mathrm{T}(n-1) + n && -1/2 \\ &= \mathrm{T}(n-2) + n + (n-1) && -2/2 \\ &= \mathrm{T}(n-3) + n + (n-1) + (n-2) && -3/2 \\ &\vdots \\ &= \mathrm{T}(n-i) + n + \cdots + (n-i+1) && -i/2 \end{aligned}$$

Vi lar $i = n$ og bruker at $\mathrm{T}(n-n) = 0$ og $1 + 2 + \cdots + n = n(n+1)/2$:

$$\mathrm{T}(n) = n(n+1)/2 - n/2 = n^2/2 + n/2 - n/2 = n^2/2$$

7 Forklar med egne ord hvordan forgjengermatrisen Π konstrueres i FLOYD-WARSHALL, og hvorfor det gir rett svar.

Vi sørger hele tiden for at π_{ij} er forgjengeren til j langs den korteste veien vi har funnet så langt. Til å begynne med er $\pi_{ij} = i$ hvis det går en kant fra i til j (dvs., hvis $w_{ij} < \infty$) og $\pi_{ij} = \text{NIL}$ ellers.

Hver gang vi finner en kortere vei via k, og oppdaterer d_{ij} til $d_{ik} + d_{kj}$, må vi sette π_{ij} til forgjengeren langs den siste biten av stien, fra k til j. Med andre ord lar vi $\pi_{ij} = \pi_{kj}$.

Om man tar med iterasjonsnummeret, har vi altså

$$\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)} \text{ hvis } d_{ik}^{(k-1)} + d_{kj}^{(k-1)} < d_{ij}^{(k-1)}.$$

Ellers lar vi $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$.

8 Forenkle uttrykket $\Omega(n^2) \cdot \mathrm{O}(\lg n) + \Theta(n)$. Uttrykk svaret med asymptotisk notasjon. Forklar og diskuter kort.

Uttrykkes kan forenkles til $\Omega(n)$.

Her er det viktig å se på oppførselen til hvert ledd og hver faktor. $\Omega(n^2)$ representerer en funksjon som vokser kvadratisk eller raskere, mens $\mathrm{O}(\lg n)$ er maksimalt logaritmisk, men kan være mindre enn det. Så det første leddet i summen kan f.eks. være $n^2 \cdot n^{-2} = 1$ – men det kan også være $2^n \cdot \lg n$, eller noe enda verre. Vi kan i det hele tatt ikke gi noen øvre eller nedre grenser for dette leddet, så summen kan bli vilkårlig stor, og vi kan ikke gi noen øvre grense for uttrykket som helhet.

Det andre leddet $\Theta(n)$ er uansett lineært, samme hvor lite det første leddet er, så summen kan ikke bli mindre enn det. Med andre ord er summen $\Omega(n)$.

9 Din venn Lurvik synes reduksjoner er litt forvirrende. Slik han forstår det, kan man redusere fra et problem B til et annet problem A, slik at man bare trenger løse A og så er man ferdig. Dersom man alt har en løsning på A, så er man altså i mål. Han mener det da virker som om A er enklere enn B, siden det ellers ikke er noe poeng i å redusere til A. Det vil si, det er ikke noe poeng i å redusere til noe som er *vanskeligere* enn det man startet med. Og siden A ikke er vanskeligere enn B, må det være slik at om A er vanskelig, så er B også vanskelig.

Lurvik er ikke helt sikker på om resonnementet hans stemmer, og kunne gjerne tenke seg en presisering enten av hvorfor det er riktig eller hvorfor det er galt.

Diskuter Lurviks forståelse av situasjonen. Forklar kort.

Her har nok Lurvik delvis misforstått. Det er slik at om man reduserer fra B til A (og reduksjonen ikke utfører signifikant arbeid), så vil en løsning på A gi oss en løsning på B. Det er sant enten A er et enkelt eller vanskelig problem. Poenget er at om vi vet at B er vanskelig, så kan vi ikke snike oss unna det ved å løse A; dermed må A også være vanskelig (siden vi ellers hadde en løsning på B).

10 Du skal fordele m gjenstander (f.eks. frukt) på n personer. Hver gjenstand tilhører nøyaktig én av et sett med *kategorier*, og hver kategori har en maks-grense for hvor mange gjenstander én person kan få (f.eks. at ingen kan få mer enn to sitrusfrukter).

For person i angir v_{ij} hvor stor verdi gjenstand j har for henne. Vi antar at v_{ij} er 0 eller 1, så verdien bare angir om hun ønsker seg gjenstanden eller ikke.

Du ønsker helst en *proporsjonal* fordeling, der hver person får sin rettferdige andel av totalverdien, slik hun selv ser det. Det vil si, hver person i skal få gjenstander som tilsammen har en verdi på minst $(v_{i1} + v_{i2} + \cdots + v_{im})/n$. Merk at dette ikke alltid er mulig (f.eks. hvis vi har for få gjenstander).

Beskriv en effektiv algoritme som finner en proporsjonal fordeling dersom det er mulig, og ellers svarer at det ikke er mulig.

Dette kan løses som et flytproblem (f.eks. med Edmonds-Karp).

- Innfør én node per gjenstand med en kant fra kilden, som har kapasitet 1.
- Innfør en «terskelnode» for hver kombinasjon av kategori og person.
- Legg inn en kant med kapasitet 1 fra en gjenstand til en terskelnode hvis gjenstanden tilhører kategorien og personen gir den verdi 1. Det er fordi det ikke er noe poeng i å gi gjenstanden til noen som gir den verdi 0; på denne måten garanterer vi at hver gjenstand bidrar til den faktiske totalverdien for en person.

- Legg inn en kant fra terskelnoden til personen med kapasitet lik terskelverdien. Vi hindrer da at personen mottar mer enn det lovlige antallet fra kategorien.
- Legg inn en kant fra hver person i til sluket med kapasitet $(v_{i1} + \cdots + v_{im})/n$, rundet opp til nærmeste heltall. Hun må få minst så mange gjenstander inn, og ved å runde opp, sikrer vi at heltallsteoremet gjelder (siden vi kun har heltallskapasiteter). Ellers kunne vi risikere at noen gjenstander ble delt i biter og fordelt utover personene.

Hvis maksimal flyt fyller alle kanter fra personer til sluk, er det mulig å gi alle sin rettferdige andel. Gjenstander som ikke er fordelt i flyt-løsningen kan fordeles vilkårlig (ev. med lotteri til dem som ønsker seg dem, e.l.).