TDT4120 Algoritmer og datastrukturer

Eksamen, 24. november 2020, 09:00–13:00

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 Forklar Huffmans algoritme (HUFFMAN) med egne ord. Det holder med en beskrivelse av hvilket problem den løser og hvilke trinn den utfører; du trenger ikke forklare hvorfor den gir korrekt svar eller diskutere kjøretid.

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	F.eks. at det handler om koding av tekst til binærstrenger og at algoritmen er grådig.
Akseptabelt	Forklarer grunnideen: Symboler vektet med frekvens. Lag én node per symbol, og slå gjentatte ganger sammen de to billigste nodene, med vektsummen i rota. Kanter til høyre og venstre barn merkes med 1 og 0.
Godt besvart	Mer om f.eks. hvordan Huffmantrær brukes til koding og dekoding eller at de gir en minimal forventet tegnlengde.

2 Flere algoritmer for å finne minimale spenntrær baserer seg på samme prinsipp for å velge én og én kant. Forklar hvilket krav de stiller til kantene som velges og gi en overordnet forklaring på hvorfor dette er trygt.

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	F.eks. at de velger grådig, eller at de velger den letteste kanten som ikke danner en sykel.
Akseptabelt	De velger den letteste kanten (ev. en av de letteste kantene) over et snitt i grafen, der et snitt er en inndeling av nodene i to mengder.
Godt besvart	F.eks. kort forklart hvordan det å velge en tyngre kant gir en selvmotsigelse, siden den kunne byttes ut (uten å danne en sykel) og gi et bedre resultat. (Trenger ikke gå i detalj på hvorfor den kan byttes ut.)

3 Diskuter ulike kjøretidsfunksjoner (f.eks. i beste og verste tilfelle og ulike typer gjennomsnitt) og forholdet mellom disse og ulike typer asymptotisk notasjon.

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	Begge følgende punkter: • Enkel forklaring av beste, verste og gjennomsnittlig kjøretid, f.eks., kjøretiden man får for beste og verste input, og gjennomsnittet over alle inputs (ev. forventning over instanser, med uniform fordeling). • Enkel beskrivelse av O, Ω og Θ, f.eks. at O og Ω gir øvre og nedre grense, og at Θ gir begge.
Akseptabelt	Begge følgende punkter, i tillegg til det foregående: • Alle kjøretidsfunksjoner kan beskrives av alle operatorene (f.eks. *best-case* med O og *worst-case* med Ω). • Om man ikke vet hvilket tilfelle man beskriver (beste, verste eller noe annet), så må man ta høyde for både beste og verste tilfelle. (Dvs., man må ta høyde for verste tilfelle om man bruker O og for beste tilfelle om man bruker Ω. Man ikke kan bruke Θ for kjøretiden generelt, dersom beste og verste tilfelle er forskjellige.)
Godt besvart	I tillegg til det foregående, minst ett av følgende punkter om andre typer gjennomsnittlig kjøretid: • Forventet kjøretid (*expected running time*), der forventningen er over tilfeldige valg som gjøres (et gjennomsnitt over disse). • Amortisert kjøretid (*amortized running time*), som er et gjennomsnitt av kjøretiden til en serie med operasjoner, f.eks. på én og samme datastruktur.

4 Din venn Smartnes har kommet over algoritmen SELECTION-SORT, som hun forklarer som følger:

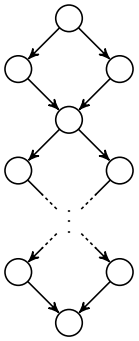
> Du får inn en tabell $A[1..n]$. For $i = n$ ned til 1, bytt plass på $A[i]$ og det største elementet i $A[1..i]$.

Hun tenker å forbedre algoritmen ved å bruke et binært søketre til å finne de største elementene. Diskuter fordeler og ulemper ved denne fremgangsmåten, og hva det ev. kunne være naturlig å gjøre i stedet.

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	Det vil være mulig å gjøre som hun tenker. Operasjonene vil ta logaritmisk tid i gjennomsnitt, som gir gjennomsnittlig kjøretid på $\Theta(n \lg n)$.
Akseptabelt	F.eks. minst to av følgende, i tillegg til det foregående: • Det vil være mer naturlig å bruke en haug (*heap*). • Man vil også kunne få $\Theta(n \lg n)$ i verste tilfelle, om man bruker et tre med balansering. • Noe diskusjon av relevante tre-operasjoner (TREE-MAXIMUM, TREE-INSERT, TREE-DELETE), og at disse tar logaritmisk tid i gjennomsnitt. • Det vil være en del overhead med et slikt tre, med noder og pekere rundt i minnet, med balansering, etc.
Godt besvart	F.eks., i tillegg til det foregående: Om man bruker en haug, er dette ekvivalent med haugsortering (HEAPSORT).

5 Figuren til høyre viser dekomponeringen av en instans i delinstanser (også kalt delproblemer). Hver kant peker fra en instans til en delinstans.



Diskuter kjøretiden til en naiv rekursiv løsning på problemet, og hvordan den kan forbedres. Hva kalles denne forbedringen og hva er det ved problemstrukturen som gjør at forbedringen er mulig?

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	Minst ett av følgende punkter: • Kjøretiden blir eksponentiell, siden hver delinstans løses mange ganger. • Forbedringen kalles dynamisk programmering eller memoisering.
Akseptabelt	Begge foregående punkter.
Godt besvart	Det som gjør forbedringen mulig kalles overlappende delproblemer: Det at flere delinstanser er avhengige av de samme delinstansene gjør at disse vil beregnes mange ganger, med mindre delløsningene mellomlagres. Om det ikke var tilfelle, ville vi ikke se noen forbedring.

Om man nevner optimal delstruktur, så trekker det ikke ned, men det er implisitt i den typen dekomponering vi har her, og er ikke det som muliggjør forbedring av kjøretid.

6 Diskuter forskjellen mellom QUICKSORT og RANDOMIZED-QUICKSORT, og hvordan denne forskjellen skiller seg fra forskjellen mellom SELECT og RANDOMIZED-SELECT. Hvordan kan bildet endre seg om du endrer QUICKSORT til å bruke SELECT som en subrutine?

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	Minst ett av følgende punkter: • QUICKSORT har kjøretid som varierer med input; RANDOMIZED-QUICKSORT overfører variasjonen til et tilfeldig valg, så samme input (f.eks. allerede sortert) ikke alltid gir verste kjøretid. • RANDOMIZED-SELECT er også randomisert, og har varierende kjøretid; SELECT, derimot gjør et spesielt godt valg av pivot, og klarer dermed å fjerne denne variasjonen, og få god (lineær) kjøretid uansett.
Akseptabelt	Begge foregående punkter.
Godt besvart	I tillegg til det foregående: Man kunne bruke SELECT til å velge pivot i QUICKSORT. Det ville ta lineær tid, men PARTITION tar uansett lineær tid, så det vil ikke øke den asymptotiske kjøretiden. Men det *vil* gjøre at man unngår kvadratisk kjøretid i verste tilfelle, så man får da en versjon av QUICKSORT med kjøretid $\Theta(n \lg n)$, uavhengig av input.

7 Du har en database med regler av typen «Hvis A så B» for ulike logiske variable A, B, C, etc. Systemet har støtte for såkalt fremoverkjeding (*forward chaining*): Om det får vite at en variabel X er sann, så settes konsekvensene av X til å være sanne, og så konsekvensene av disse, etc., helt til alle konsekvenser er funnet.

Sjefen din ønsker en ny funksjonalitet, der du får vite *hvorfor* en variabel er sann: Hvilken annen variabel var den en konsekvens av? Og hvilken variabel var *den* en konsekvens av, etc.? Hun ønsker også at disse forklaringene skal være så enkle som mulig, dvs., unngå kjeder med veldig mange variable. Hvordan ville du angripe dette problemet?

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	Vi kan representere reglene som en graf, så kjedingen blir en form for traversering: Å sette variable til å være sanne tilsvarer å traversere dem.
Akseptabelt	Å finne en enkel forklaring tilsvarer å finne korteste vei fra X i regelgrafen.
Godt besvart	Vi kan bruke bredde-først-søk (BFS), siden lengden bare er avhengig av antall kanter (dvs., vi har en uvektet graf).

8 Din venn Lurvik har lest at Ford-Fulkerson har pseudopolynomisk kjøretid, og mener det er fordi maks-flyt egentlig er NP-hardt. Han argumenterer som følger: Maks-flyt er et spesialtilfelle av såkalt vektet hyperflyt (*min-cost hyperflow*), der kantene har vekter og hver kant kan være koblet til flere noder – og i artikkelen *A Hypergraph Network Simplex Algorithm* skriver Beckenbach (2018, oversatt):

> Spesifikt så er det NP-hardt å finne en heltallig vektet hyperflyt (f.eks. ved å redusere til 3D-Matching) [. . .].

Diskuter påstandene til Lurvik og utsagnet fra Beckenbach.

Forklar og utdyp. Knytt til relevant teori, gjerne i ulike deler av pensum.

Noe uttelling	For eksempel: Her tar Lurvik trolig feil, siden et slikt bevis ville medføre at P = NP.
Akseptabelt	Reduksjonene går feil vei. Lurvik reduserer til vektet hyperflyt, og Beckenbach reduserer til 3D-matching, men begge skulle ha redusert i motsatt retning.
Godt besvart	I tillegg til det foregående, f.eks. at Ford-Fulkerson faktisk *har* en pseudopolynomisk kjøretid, siden denne er avhengig av størrelsen på flyten (som igjen er avhengig av størrelsen på kapasitetene).