Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Filip Strömbäck och Magnus Nielsen

25 augusti 2020

TDDE22 & 725G97
Datastrukturer och algoritmer
Datortentamen (DAT1)
2020-08-25, 08–12

Examinator:	Erik Nilsson
Jour:	Magnus Nielsen (Zoom: 662 2309 8004, telefon 013-28 58 86)
Antal uppgifter:	3
Max poäng:	40 poäng
Preliminära gränser TDDE22:	Betyg 5 = 35p, 4 = 27p, 3 = 20p.
Preliminära gränser 725G97:	Betyg G = 20p, VG = 30p.
Hjälpmedel:	Inga hjälpmedel tillåtna!

REGLER

- Tentan genomförs individuellt.
- Tentan ska göras i en lugn miljö utan andra personer i samma rum.
- All kommunikation är förbjuden, undantaget frågor till kurspersonal.
- Alla källor du tar inspiration av ska redovisas.
- Dina lösningar skickas in via Lisam enligt nedan och kommer att köras genom Urkund.
- Var beredd på att i efterhand kunna redogöra för dina svar.
- Markera tydligt med rubriker eller dylikt vilken uppgift och deluppgift du svarar på (vi vill snabbt kunna hitta svaret på exempelvis uppgift 2 d).
- Observera att betygsgränserna kan komma att justeras, i samtliga kurser.
- Vid frågor om tidskomplexitet, svara alltid på den form som är mest relevant.
- Om inte annat framgår ska indexering av arrayer/listor börja från 0.
- Om ett problem medger flera olika lösningar, t.ex. algoritmer med olika tidskomplexitet, ger endast optimala lösningar maximalt antal poäng.

INLÄMNING

Lämna in dina svar som ett dokument med tydliga rubriker i det Lisam-rum som du har fått information om via e-post tidigare. Använd en ordbehandlare som låter dig formatera text med rubriker och dylikt. Sikta på att lämna in 2-4 sidor text. Skicka sedan in antingen som *pdf*, *doc*, *docx*, *odt*, *org* eller *txt*.

FRÅGOR

Har du frågor om någon uppgift, eller om någonting går snett med din utrustning, hör av dig till kursledare antingen via Zoom (662 2309 8004) eller telefon (013-28 58 86).

Lycka till!

1. **Regler** (0 p)

Läs tentans regler ovan. Skriv sedan följande text i ditt dokument för att bekräfta att du har läst och förstått dem. Accepterar du inte reglerna så rättar vi inte tentan.

Jag har läst och förstått tentans regler, och jag lovar att jag har följt dem under tentans gång.

2. **Storskalig tentahantering** (20 p)

Snålköpings universitet har nyss insett att de kan spara väldigt mycket pengar genom att effektivisera tentamenshanteringen. De tänker att med hjälp av ett bra datasystem för att hantera alla tentamina som skrivs varje tentaperiod så kan de minska personalbehovet med minst 120% (indikativt för universitetskompetensen?). I och med att det är billigt att leva i Snålköping, och att universitetet har låga intagningspoäng, går det ca 100 000 studenter där. Varje tentaperiod går det tentamina för 1 000 kurser, och varje student skriver ungefär 3 tentamina, så tentamensenheten behöver hantera ungefär 300 000 tentor per tentamensperiod.

Tentamensenheten vill att systemet ska kunna göra följande för att möta deras behov:

- Direkt efter att en tentamen är slut, så tar tentamensvakten alla inlämnade tentamina och scannar in dem, så att de kan lagras digitalt i systemet (digitalisering är bra). Tentamensvakten matar också in kurskod och student-id för varje tentamen. För att minska risken till fusk sitter studenterna mer eller mindre slumpmässigt utspridda i de många salarna som finns på campus. Alla tentamina matas således inte in i någon ordning. Vi antar också att en tentamen för en kurs bara går en gång per tentamensperiod.
- Examinator för varje kurs kommer någon dag efter tentamenstillfället och vill hämta ut alla tentamina för en viss kurskod.
- I vissa fall kan också disciplinnämnden behöva hämta ut en eller ett fåtal tentamina. I så fall vet de kurskod och student-id.

Det företag som Snålköpings universitet har anlitat för att implementera systemet har nu kommit fram till tre möjliga implementationer av de centrala datastrukturerna för att hantera alla tentamina. Det är nu din uppgift som oberoende expert att granska förslagen och komma fram till vad som är bäst (universitetet valde såklart det billigaste företaget...). Förslagen är som följer:

1. Lagra alla tentamina i en enkellänkad lista, sorterad efter kurskod, och sedan student-id.
2. Lagra alla tentamina i en array, sorterad efter kurskod, och sedan student-id.
3. Lagra alla tentamina i ett balanserat sökträd, sorterad efter kurskod, och sedan student-id.
4. Lagra alla tentamina för en viss kurs i en array, och sedan lagra alla dessa arrayer i en hashtabell med kurskod som nyckel. Alltså: `HashMap<String, ArrayList<Exam>>`

Frågor:

(a) För vart och ett av alternativen 1-4 ovan, beskriv hur systemet hittar **en** tentamen givet kurskod och student-id. Ange även tidskomplexiteten för uppslagningen uttryckt i n (antalet tentamina i datastrukturen) i varje fall. För full poäng krävs optimal lösning. (4)

(b) För vart och ett av alternativen 1-4 ovan, beskriv hur systemet hittar **alla** tentamina för en viss kurskod. Ange även tidskomplexiteten för uppslagningen uttryckt i n (antalet tentamina i datastrukturen) i varje fall. För full poäng krävs optimal lösning. (4)

(c) För vart och ett av alternativen 1-4 ovan, beskriv hur systemet sätter in en tentamen i datastrukturen. Ange även tidskomplexiteten för uppslagningen uttryckt i n (antalet tentamina i datastrukturen) i varje fall. För full poäng krävs optimal lösning. (4)

(d) För vart och ett av alternativen 1-4 ovan, beräkna den totala tidskomplexiteten för att sätta in n tentamina och sedan hitta dem. Det vill säga, allt arbete med datastrukturen som sker under en tentamensperiod. Redovisa dina beräkningar/resonemang. (2)

(e) Givet dina svar ovan, vilken av alternativen 1-4 skulle du använda i systemet? Beskriv för- och nackdelar med var och en av lösningarna. Glöm inte att resonera kring tidskomplexitet och minnesanvändning. (2)

(f) Kan du förbättra alternativ nummer 4 ovan, så att det går snabbare att hitta en tentamen för en specifik student? Ange i så fall en tidskomplexitet för den optimerade lösningen. Motivera ditt svar. (2)

(g) Antag att tentamensenheten visste att ingen examinator vill hämta sina tentamina förrän alla kurser är klara och införda i systemet. Du vet alltså att alla insättningar i datastrukturen sker innan någon vill leta efter något i datastrukturen. Kan du i detta fallet optimera den totala tiden som behövs för alternativ nummer 2 som du beräknade i deluppgift (d)? Ange i så fall en tidskomplexitet för den optimerade lösningen. Motivera ditt svar. (2)

3. **Vägvisande spelutveckling** (20 p)

Din kompis har just blivit meddragen som enda utvecklare i en startup (de andra tre är säljare) som gör datorspel. Spelet är ett traditionellt Pay to Win-spel där man ska ta sig igenom ett antal nivåer samtidigt som man blir attackerad av systemadministratörer som inte fått sitt kaffe. Spelvärlden är uppbyggd i form av ett rutnät på ungefär 100x100 rutor, där vissa rutor är passerbara och vissa rutor är väggar och därmed opasserbara. Rutorna kan också innehålla fiender eller spelaren själv. Spelet är ett mobilspel, så man ska kunna trycka på skärmen för att förflytta spelaren. Trycker man någonstans ska spelaren ta den närmsta passerbara vägen till den rutan man tryckte på, om en sådan finns. För att spelet ska fungera på så många telefoner som möjligt ska spelet använda så lite minne som möjligt (absolut maximalt 100 MB).

Din kompis har tyvärr inte fått äran att läsa en DALG-kurs, och har nu kommit för att rådfråga dig angående val av datastrukturer och algoritmer i spelet.

(a) Spelet försöker att alltid ha ett visst antal fiender vid liv i spelet. När en fiende dör, så ska spelet komma ihåg detta så att det senare kan skapa en ny lika dan fiende när det är lämpligt. Den som försvann först ska skapas först. Beskriv för din kompis vilken datastruktur som är lämplig att använda för detta, och hur lång tid insättning och borttagning i denna tar. (2)

(b) Efter lite speltestning har din kompis kommit på att om man fokuserar på att besegra de enkla fienderna först så kommer man senare inte att möta så många svåra fiender i och med att det var så många enkla fiender på tur sedan tidigt i spelet. För att råda bot på detta funderar din kompis på att ge den starkaste fienden företräde när nya fiender skapas. Beskriv för din kompis vilken datastruktur som är lämplig att använda för detta, och hur lång tid insättning och borttagning i denna tar. (2)

(c) Förklara för din kompis hur denne kan implementera en algoritm som hittar kortaste vägen dit spelaren har tryckt på skärmen. Ge också tidskomplexiteten för algoritmen uttryckt i n (antalet rutor på spelplanen). För full poäng krävs optimal lösning. (4)

(d) En vecka senare kommer din kompis till dig och ser något upprörd ut. De tre säljarna har just lovat investerarna att spelet ska ha stöd för terräng. De vill alltså att spelarens karaktär ska röra sig olika fort genom olika rutor (det går exempelvis långsamt att gå igenom en sladdhärva), och inser att detta påverkar vilken den snabbaste vägen är. Förklara för din kompis hur denne kan implementera en algoritm som löser problemet (och ge tidskomplexiteten för denna), alternativt varför din lösning i (a) fungerar även här. (2)

(e) Efter att ha programmerat ytterligare en vecka hör din kompis återigen av sig. Nu har det blivit dags att få fienderna att röra sig mot spelaren. I en värld finns det maximalt 1000 fiender som alla ska röra sig mot spelaren. Din kompis har försökt att använda samma algoritm som i (b) för att hitta vägar åt alla fiander, men det är för långsamt (investerarna tolererar inte lagg). Förklara för din kompis hur denne kan implementera en algoritm som hittar kortaste vägen mellan varje fiende och spelarkaraktären. Ge också tidskomplexiteten för algoritmen uttryckt i n (antalet rutor på spelplanen). För full poäng krävs optimal lösning. (4)

(f) Efter ytterligare ett tag hör din kompis återigen av sig, och har funderat på om det går att beräkna all pathfinding i förväg då det skulle snabba upp spelet så att man hinner med fler micro-transactions. Du vet att detta går att göra i $\mathcal{O}(n^3)$ tid, och genererar en nxn-matris där varje cell innehåller den kortaste vägen mellan ruta i och j. Du funderar dock på om detta är en bra idé i det här fallet. Förklara för din kompis om du tror att det är en bra idé att beräkna alla vägar i förväg på detta sättet eller inte. Berätta om för och nackdelar med denna idé, samt vad din slutsats är. (4)

(g) Din kompis pratade också om en ny idé till spelet: spelaren kan ha ett antal engångslösenord (som man köper via micro-transactions) som går att använda för att öppna en av många låsta dörrar i spelet. Engångslösenorden förbrukas naturligtvis efter att de har använts. Beskriv för din kompis hur algoritmen i (d) kan modifieras för att hitta vägar genom dörrar och ta hänsyn till att engångslösenorden bara får användas en gång. (2)