Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Filip Strömbäck och Magnus Nielsen

27 oktober 2023

TDDI16
Datastrukturer och algoritmer
Datortentamen (DAT1)
2023-10-27, 14–18

Examinator:	Filip Strömbäck
Jour:	Filip Strömbäck (telefon 013-28 27 52)
Antal uppgifter:	6
Max poäng:	40 poäng
Preliminära gränser:	Betyg 5 = 35p, 4 = 27p, 3 = 20p.
Hjälpmedel:	Inga hjälpmedel tillåtna!

VÄNLIGEN IAKTTAG FÖLJANDE

- Observera att betygsgränserna kan komma att justeras, i samtliga kurser.
- Vid frågor om tidskomplexitet, svara alltid på den form som är mest relevant.
- Skriv dina lösningar i valfri textredigerare, exempelvis LibreOffice, Emacs, eller liknande. Strukturera din text så att det enkelt går att se vad som svarar på vilka deluppgifter.
- Lösningar till olika problem skall skrivas i en egen fil. Skriv inte två lösningar i samma fil. Delproblem får dela fil.
- Om inte annat framgår ska indexering av arrayer/listor börja från 0.
- Skicka in som lösning till rätt problem i tentaklienten, och döp filen till ett passande namn (exempelvis uppg1.txt/uppg1.odt).
- **MOTIVERA DINA SVAR ORDENTLIGT:** avsaknad av, eller otillräckliga, förklaringar resulterar i poängavdrag. **Även felaktiga svar kan ge poäng** om de är korrekt motiverade.
- Om ett problem medger flera olika lösningar, t.ex. algoritmer med olika tidskomplexitet, ger endast optimala lösningar maximalt antal poäng.
- SE TILL ATT DINA LÖSNINGAR/SVAR ÄR LÄSLIGA. Oläsliga lösningar beaktas ej.

Lycka till!

1. **Sorteringsalgoritmer** (5 p)

Svaren behöver ej motiveras.

Efter **ett fåtal** iterationer av några olika sorteringsalgoritmer på den osorterade arrayen *Original* i tabellen nedan (iteration i bemärkelse fullständig körning av inre loop, alternativt rekursivt anrop) har vi resultaten i *1, 2, 3, 4* och *5*.

Original:	96	55	77	34	89	20	53	38	49	48	74	98	50	9	52
1:	9	20	34	38	49	50	48	52	53	77	74	98	55	96	89
2:	9	20	34	96	55	77	38	89	48	53	49	50	52	74	98
3:	77	74	53	49	55	52	50	38	34	48	9	20	89	96	98
4:	9	20	34	77	89	55	53	38	49	48	74	98	50	96	52
5:	34	55	77	96	89	20	53	38	49	48	74	98	50	9	52
Sorterad:	9	20	34	38	48	49	50	52	53	55	74	77	89	96	98

Matcha fyra av de delvis sorterade arrayerna mot algoritmerna nedan. Felaktig matchning ger minuspoäng, dock kan uppgiften ej ge total minuspoäng. Utelämnat svar ger ej minuspoäng. För totalt 4 poäng krävs alltså 4 korrekta svar.

(a) Quicksort, elementet längst till höger i partitionen används som pivot (1)

(b) Bubble sort, element bubblas från höger till vänster (1)

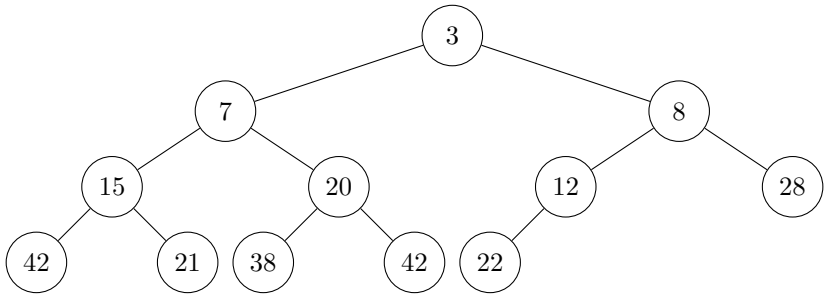
(c) Insertionsort (1)

(d) Selectionsort (1)

För den femte och sista poängen i frågan (här ger felaktigt svar inte avdrag):

- Vilken algoritm användes i den array som inte matchades ovan? Ge en kort motivering. (1)

2. **Träd** (4 p)



(a) Är trädet ovan ett binärt sökträd? Motivera ditt svar. (1)

(b) Är trädet ovan komplett? Motivera ditt svar. (1)

(c) Är trädet ovan en min-heap? Motivera ditt svar. (1)

(d) Gör en levelorder-traversering av trädet. I vilken ordning traverseras noderna? (1)

3. **It's a good day to DALG hard** (10 p)

Du har beslutat dig för att vinna programmerings-VM. Du måste därför hitta alla medstudenter som är bättre än dig för att kunna utmana dem på veckodueller i IMPA. Planen är att hitta svårare och svårare motståndare så att du successivt bygger upp din programmeringsförmåga.

För att göra detta behöver du självfallet hitta och ordna dina medstudenter efter programmeringsförmåga. Du vill därför implementera en datastruktur för detta. Den ska ha följande operationer:

`create()` Skapa en ny, tom datastruktur.

`insert(student, points)` Sätt in en ny student vid namn *student*, som fick *points* poäng på senaste DALG-tentan. Om studenten redan fanns i datastrukturen ska den inte sättas in igen.

`find_next()` Hitta nästa motståndare att utmana. Detta ska vara den student med den lägsta poängen i datastrukturen.

`defeated()` Anropas när den student med lägst poäng har besegrats i en programmeringsduell. Studenten ska därmed tas bort från datastrukturen.

Efter lite funderande så kommer du fram till följande möjliga implementationer av datastrukturen:

1: Lagra studenter som par, $(points, student)$ i ett balanserat binärt sökträd, ordnat efter poäng. När en student ska sättas in söker vi helt enkelt igenom sökträdet efter studentens namn för att se om den redan finns med eller inte. Om den inte fanns, så sätts den in som vanligt. Att hitta nästa motståndare görs genom att helt enkelt hitta det minsta elementet i sökträdet. Det kan sedan tas bort eller returneras beroende på vilken av `find_next` och `defeated` som anropades.

2: Lagra studenter som par, $(points, student)$, i en min-heap ordnad efter poäng. Utöver det finns också en hashtabell som lagrar namnen på de studenter som finns i heapen. Insättning görs då genom att först kontrollera om studenten finns i hashtabellen. Om den inte finns, så sätts den in både i hashtabellen och i heapen. Att hitta nästa motståndare görs då genom att hitta minsta elementet i heapen. Detta kan sedan tas bort eller returneras beroende på vilken av `find_next` och `defeated` som anropades.

(fortsättning på nästa sida)

Svara på följande frågor med avseende på de två alternativen ovan:

(a) Vilken tidskomplexitet har vart och ett av alternativen 1–2 ovan för att sätta in en student i datastrukturen (dvs. `insert`)? Uttryck tidskomplexiteten i termer av antalet element i datastrukturen (n). Beskriv kort ditt resonemang. (2)

(b) Vilken tidskomplexitet har vart och ett av alternativen 1–2 ovan för att hitta nästa motståndare i datastrukturen (dvs. `find_next`)? Uttryck tidskomplexiteten i termer av antalet element i datastrukturen (n). Beskriv kort ditt resonemang. (2)

(c) Vilken tidskomplexitet har vart och ett av alternativen 1–2 ovan för att ta bort den nuvarande motståndaren i datastrukturen (dvs. `defeated`)? Uttryck tidskomplexiteten i termer av antalet element i datastrukturen (n). Beskriv kort ditt resonemang. (2)

(d) Vilket av alternativen 1–2 är bäst? Fundera bland annat på hur lång tid det tar att sätta in n studenter i en tom datastruktur, och minnesanvändningen av de två alternativen. Motivera ditt svar. (2)

(e) Du vill modifiera beteendet hos `insert` när studenten redan finns i datastrukturen. I stället för att ignorera det nya resultatet (ex. från en omtenta), så vill du uppdatera resultatet i datastrukturen, så att det högsta resultatet finns med. Alltså, om studenten fick bättre resultat på en omtenta, ska poängen höjas. Beskriv hur du kan göra detta effektivt för alternativ 2. Förklara på 5–10 meningar hur detta kan ådstakommas, och vilka ändringar som måste göras i datastrukturen. (2)

4. **Algoritmer och tidskomplexitet** (6 p)

Visa kort hur du kommer fram till tidskomplexiteten på följande uppgifter.

(a) Beräkna tidskomplexiten med avseende på parametern n för följande funktion: (1)

```
int f(int n) {
  int result = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
      result += j;
    }
  }
  return result;
}
```

(b) Beräkna tidskomplexiten med avseende på parametern n för följande funktion: (1)

```
int fn(int n) {
  int result = 0;
  for (int i = n; i > 0; i = i / 2) {
    result += i;
  }
  result += f(n);
  return result;
}
```

(c) Beräkna tidskomplexiten med avseende på parametrarna n och m för följande funktion: (2)

```
int fun(int n, int m) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += f(m);
  }
  return sum;
}
```

(d) Beräkna tidskomplexiteten av följande funktion. Antag att n är storleken av arrayen `into`. Tänk på bästa- och värstafall i din analys. Beskriv också kortfattat hur du kom fram till ditt svar. (2)

```
void insert_if_missing(vector<Animal> &into, const Animal &insert) {
  for (int i = 0; i < into.size(); i++) {
    if (into[i] == insert) {
      return;
    }
  }
  into.push_back(insert);
}
```

5. **Roliga djur** (5 p)

Programmet VärldensRoligasteDjurvideor™ (VRD™) är ett program som visar och rankar femsekundersklipp av djur som gör roliga saker. Rankningen går ut på att programmet har en livepublik™, och undersöker hur mycket publiken skrattar åt varje klipp.

Historiskt sett har programmet anställt en stor mängd studenter för att räkna alla skratt. För varje person i publiken har de anställt en student vars uppgift är att hålla koll på vilka klipp som individen skrattade åt.

I och med ökad publikstorlek och ökade personalkostnader vill de nu automatisera räknandet. De har därför ersatt studenterna med mikrofoner. Det finns en mikrofon framför varje person i publiken. Mikrofonerna är kopplade till en AI, som känner igen när personerna skrattar. Däremot kan den inte räkna ordentligt, utan den används bara för att känna igen, och spara när publiken skrattar.

Den data som AIn producerar är i formen av en array. Varje element i arrayen motsvarar data insamlad för ett videoklipp. Varje element innehåller då i sin tur en array. Denna arrayen innehåller platsnumret av alla som har skrattat under klippet. Data lagras alltså som `vector<vector<int>>`.

Exempelvis har följande data samlats in. Varje rad i tabellen är ett element i den yttre arrayen, och innehållet i kolumnen *Skratt ifrån publiken* är innehållet i den inre arrayen.

Klipp	**Skratt ifrån publiken**
0	1, 3, 8, 10, 4, 3, 9
1	3, 8, 1, 7, 3, 2, 11, 6
2	3, 7, 5, 3, 2, 11
...	...

Din kollega har fått i uppgift att beräkna rankningen av varje video. Din uppgift är i stället att beräkna vilka i publiken som hade mest roligt under inspelningen. För varje person vill du alltså räkna hur många klipp personen skrattade åt. Om samma person skrattade flera gånger under samma klipp vill vi bara räkna detta en gång. Utdata ska vara en array av heltal, där varje element representerar en plats. Du vet att det finns p platser, numrerade från 0 till $p - 1$.

(a) Beskriv hur du effektivt kan beräkna antalet skratt för varje plats enligt ovan. (3)

Beskriv din algoritm, gärna i punktform eller med pseudokod. Tänk på att ha med hur du lagrar data och vilka datastrukturer din algoritm använder. Du behöver inte detaljbeskriva sådant som finns i standardbiblioteket.

(b) Vilken tidskomplexitet har din implementation uttryckt i det totala antalet platser (p), och det totala antalet skratt (s)? (1)

(c) Hur mycket minne, utöver indatan, använder din implementation? Uttryck ditt svar i p och s som ovan. (1)

6. **En kulinarisk promenad** (10 p)

Efter att ha klarat DALG-tentan har du beslutat dig för att ta en välförtjänt DALG-relaterad semester i Eulers hemstad, Köningsberg. Första kvällen efter att ha kommit dit vill du testa på att äta en distribuerad middag. Du vill alltså kombinera rätter från alla restauranger för att skapa en perfekt n-rätters middag, med tillhörande sightseeing. Planen är alltså att du äter en rätt på en restaurang, och sedan går vidare till nästa restaurang och äter nästa rätt där (på vägen kan du också se på någon av de kända broarna i staden).

Till din hjälp har du en lista över de rätter du är intresserad av att äta (tyvärr inte alla rätter på alla restauranger, du måste ju spara några rätter till dagen därpå). Varje rätt har också en eller flera rätter som måste ätas innan rätten, för optimal upplevelse. Dessa rätter måste bara ätas någon gång innan rätten i fråga. Det är alltså okej att inte äta rätterna i direkt följd. Du kan anta att det inte finns cykliska beroenden i listan.

Listan ser ut som följer:

ID	**Restaurang**	**Rätt**	**Måste ätas efter**
0	Dorsia	Seekoeier ceviche	-
1	Kolm luud	Pivo od maslaca	0
2	Los Pollos Hermanas	Pollo Caliente	0, 1
3	Kafejo Monk	Velika Salata	0, 7, 2
4	Los Pollos Hermanas	Pollo Frio	2
	...	...	...

Utifrån detta kan vi exempelvis se att rätt nummer 2 måste ätas efter rätt 0 och 1.

(a) Beskriv hur du från listan av n rätter (enligt ovan) kan implementera en algoritm som beräknar en möjlig ordning du kan äta rätterna i så att alla beroenden är uppfyllda. Algoritmen ska producera resultat i form av en array av rätternas ID. (4)

Beskriv din algoritm, gärna i punktform eller med pseudokod. Tänk på att ha med vilka datastrukturer du använder, vilken data som lagras i dem, och hur de initieras. Du behöver inte detaljbeskriva sådant som finns i standardbiblioteket.

(b) Vilken tidskomplexitet har din implementation i (a), uttryckt i antal rätter (n), och det totala antalet beroenden (m)? (1)

(c) Dagen därpå ansluter en av dina kompisar. Din kompis har spanat in en specifik dessert som denne vill prova (id k). Likt du, är din kompis intresserad av den kulinariska upplevelsen, men har en annan approach. I stället för att se till att ha ätit alla rekommenderade rätter innan, vill din kompis se till att äta *en* av de rekommenderade rätterna *precis* innan rätten själv. Tyvärr är din kompis inte lika nyfiken på den lokala kulinariska upplevelsen som du, och vill därmed äta så få rätter som möjligt totalt. (4)

Exempelvis: för rätt nummer 3 så vill din kompis äta en av rätterna, 0, 7, eller 2 precis innan. Däremot så räcker det att *en* av dessa äts innan. För att i sin tur kunna äta exempelvis rätt 2, så måste då antingen rätt 0 eller rätt 1 ätas precis innan rätt 2, och så vidare. Om $k = 3$ så är alltså ordningen 0, 2, 3 giltig (men inte kortast).

Beskriv hur du kan implementera en algoritm som löser detta problem. Algoritmen ska, likt den förra, producera en array av IDt på de rätter som ska ätas. Den producerade arrayen ska vara så kort som möjligt, men se till så att kraven ovan uppfylls.

Beskriv din algoritm, gärna i punktform eller med pseudokod. Tänk på att ha med vilka datastrukturer du använder, vilken data som lagras i dem, och hur de initieras. Du behöver inte detaljbeskriva sådant som finns i standardbiblioteket.

(d) Vilken tidskomplexitet har din implementation i (c), uttryckt i antal rätter (n), och det (1)

totala antalet beroenden (m)?