Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Filip Strömbäck och Magnus Nielsen

31 oktober 2024

TDDI16
Datastrukturer och algoritmer
Datortentamen (DAT1)
2024-10-31, 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:	68	27	63	77	87	69	62	33	59	67	53	97	83	72	66
1:	27	63	68	77	87	69	62	33	59	67	53	97	83	72	66
2:	77	67	72	66	63	69	62	33	59	27	53	68	83	87	97
3:	27	33	53	77	87	69	62	68	59	67	63	97	83	72	66
4:	53	27	33	59	62	63	66	68	67	69	87	97	83	72	77
5:	27	33	53	68	59	63	77	87	69	62	66	67	72	83	97
Sorterad:	27	33	53	59	62	63	66	67	68	69	72	77	83	87	97

Matcha 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 5 poäng krävs alltså 5 korrekta svar.

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

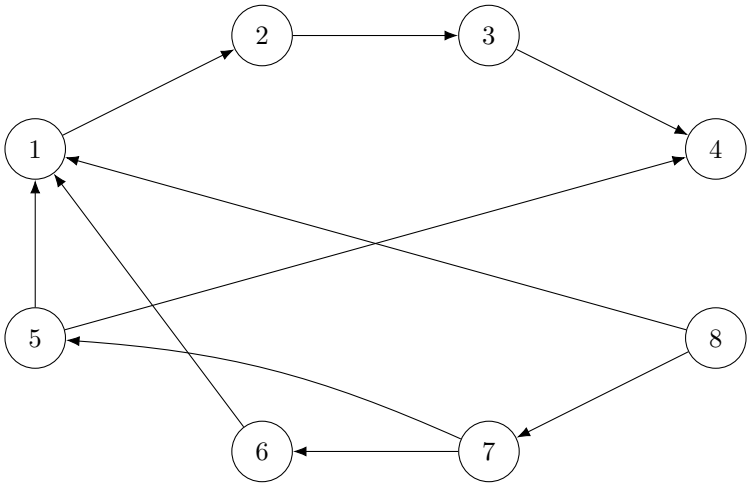
(b) Bubble sort, element bubblas från höger till vänster (1)

(c) Insertionsort (1)

(d) Selectionsort (1)

(e) Heapsort (1)

2. **Grafer** (4 p)



Svara på följande frågor om grafen ovan. Antag att bågar från en nod traverseras i stigande ordning efter numret i till-noden. Exempelvis traverseras bågen $8 \to 1$ innan $8 \to 7$.

(a) I vilken ordning besöks noderna om grafen traverseras med DFS från nod 8? (1)

(b) Vilken väg hittar DFS-sökningen i (a) till nod 4? (1)

(c) I vilken ordning besöks noderna om grafen traverseras med BFS från nod 8? (1)

(d) Vilken väg hittar BFS-sökningen i (c) till nod 4? (1)

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

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

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

```
int f(int n) {
  int result = 0;
  for (int i = 1; i < n / 2; i += 2) {
    result += 1;
  }
  return result;
}
```

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

```
int fn(int n) {
  if (n > 0) {
    return fn(n / 2) + 1;
  } else {
    return 0;
  }
}
```

(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 = 1; i < n; i++) {
    sum += f(10);
    sum += fn(m);
  }
  return sum;
}
```

(d) Beräkna tidskomplexiteten på `insertSorted` med avseende på storleken av `arr` ($= n$). (2)
Tänk på bästa och värsta fall in din analys.

```
void insertSorted(vector<int> &arr, int value) {
  int position = findPosition(arr, value);
  // Sätter in elementet på plats 'position'
  arr.insert(arr.begin() + position, value);
}

int findPosition(const vector<int> &arr, int value) {
  int position = 0;
  int size = arr.size();
  while (size > 0) {
    size_t middle = position + (size / 2);
    if (arr[middle] < value) {
      position = middle + 1;
      size = (size - 1) / 2;
    } else {
      size = size / 2;
    }
  }
  return position;
}
```

4. **Positiv summering** (10 p)

När man arbetar med flyttal (ex.vis `float` och `double`) så finns det alltid avrundningsfel på grund av att flyttal har begränsad precision. Lägger vi exempelvis ihop talet 10^{-9} med talet 10^9 så blir resultatet 10^9. Detta innebär att vi måste vara försiktiga om vi har en stor mängd flyttal med olika magnitud som ska summeras.

Antag exempelvis att vi har en lista som innehåller talet 10^9 följt av 1 000 000 element som innehåller 10^{-9}. På grund av problematiken ovan så får vi resultatet 10^9 om vi summerar elementen från vänster till höger. För att få rätt svar så måste vi i det här fallet börja med de små talen. Då får vi det korrekta svaret, 1 000 000 000,001.

För att hantera detta på ett generellt sätt vill du bygga en funktion, `sum(x)` som summerar en lista av flyttal för att minimera avrundningsfel. Funktionen antar att alla flyttal är positiva, men inte att de är sorterade på något sätt. Du har följande idéer till implementationen:

Alternativ 1:

0. Lagra listan x i en dynamisk array.
1. Sortera arrayen x med insertion sort, i sjunkande ordning så att det minsta elementet hamnar sist.
2. Ta bort de två sista elementen i x, spara dem i a och b.
3. Lägg in summan av a och b sist i x.
4. Om x innehåller mer än 1 element, gå till steg 1. Returnera annars $x[0]$.

Alternativ 2:

0. Lagra alla element i listan x i en min-heap, h, med hjälp av *make-heap*.
1. Ta bort min-elementet i h, lagra det i a.
2. Ta bort min-elementet i h, lagra det i b.
3. Lägg in summan av a och b i h.
4. Om h innehåller mer än 1 element, gå till steg 1. Ta annars bort det sista elementet i h och returnera det.

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

(a) Vilken tidskomplexitet har alternativ 1 ovan i **bästa** och **värsta** fall? Uttryck tidskomplexiteten i termer av n, som är antalet element i listan x. Beskriv kort ditt resonemang. (4)

(b) Vilken tidskomplexitet har alternativ 2 ovan i **bästa** och **värsta** fall? Uttryck tidskomplexiteten i termer av n, som är antalet element i listan x. Beskriv kort ditt resonemang. (4)

(c) Beskriv hur du kan optimera steg 1 i alternativ 1. Dels så att steget är snabbare i *första* iterationen av loopen, och dels så att den har ett bästa fall av $\mathcal{O}(1)$ i resterande iterationer. (2)

5. **Sketchy puns på DVD** (6 p)

En av dina barndomsvänner har valt att bli komiker. Under de senaste åren har din vän producerat ganska mycket material och tänker därför samla ihop materialet i form av en samling DVD:er. Materialet består av längre sketcher (korta filmer) och en stor samling korta puns. Planen är att avsluta varje sketch med en lämplig pun för att tittaren ska ha något att se fram emot.

Problemet som kvarstår är att välja ut lämpliga puns att avsluta sketcherna med. Det viktigaste med det pun som avslutar varje sketch är att det är lagom roliga. Det måste överträffa sketchen, men inte för mycket (de roligaste måste antagligen sparas till roligare sketcher). För varje sketch vill vi alltså välja det pun som närmast överträffar sketchen i rolighet av de som ännu inte har används på nuvarande DVD:n. Som underlag till detta finns följande information om alla puns:

ID	Rolighet	Pun
1	100 %	What do you call a fake noodle? An Impasta.
2	95 %	A bicycle can't stand on its own because it's two-tired.
3	35 %	The shovel was a ground-breaking invention.
4	40 %	Sausage puns are the wurst.
5	20 %	Always trust a glue salesman. They tend to stick to their word.
6	90 %	What did the grape say when it got crushed? Nothing, it just let out a little wine!
...	...	...

Det finns även följande data om hur roliga sketcherna är:

Sketch	Rolighet
1	10 %
2	70 %
3	40 %
4	92 %
...	...

I och med att det finns mycket material behöver din vän lösa problemet många gånger. Därför har du blivit tillfrågad att skriva ett program som hittar lämpliga puns för varje segment. Notera att vi vill såklart inte att samma pun används mer än en gång på varje DVD.

(a) Beskriv hur du kan implementera ett program som hjälper din kompis att välja puns för en DVD, givet de två tabellerna ovan. Antag att tabellerna läses in från två separata filer på disk. Du kan också anta att det finns tillräckligt många bra puns så att problemet går att lösa. Ditt program ska producera en lista med ID på de puns som ska användas, i samma ordning som segmenten i indata. (4)

Beskriv vilka datastrukturer du använder, samt den algoritm du använder för att lösa problemet. Beskriv gärna algoritmen i punktform eller med pseudokod. Du behöver inte detaljerat beskriva sådant som finns i standardbiblioteket (men nämn gärna kort hur det du använder fungerar).

(b) Vilken tidskomplexitet har ditt program uttryckt i antal segment på DVD:n (n), samt antal puns i den första tabellen (p). Motivera ditt svar. (1)

(c) Hur mycket minne använder ditt program uttryckt i n och p (som i (b)). Motivera ditt svar. (1)

6. **Minnesläckor** (9 p)

När man programmerar i C++ är det lätt hänt att man glömmer att frigöra minne, vilket leder till minnesläckor. Ett sätt att identifiera vilka objekt som är en del av en minnesläcka är att undersöka vilka objekt som går att nå via pekare (antingen direkt eller indirekt) om man utgår från alla pekare som finns i lokala (dvs. inuti en funktion) och globala variabler.

Pekare i lokala och globala variabler:

Variabel	Pekar på
`a`	4
`b`	3, 8
`c`	1
`d`	4
...	...

Pekare i heapallokerade objekt:

Objekt-ID	Pekar på
1	
2	3
3	1, 5
4	4, 1
5	9
6	2, 1
...	...

Givet den datan vi ser i tabellerna ovan så är exempelvis objekt 5 inte en minnesläcka, eftersom den går att nå från variabel `b` som pekar på objekt 3. Objekt 3 innehåller i sin tur en pekare som pekar på objekt 5. Objekt 2 är dock en minnesläcka, eftersom det inte går att ta sig till objekt 2 genom att följa pekare från den vänstra tabellen för att komma till objekt 2.

(a) Beskriv hur du med hjälp av datan i de två tabellerna kan identifiera vilka objekt som är minnesläckor. (4)

Beskriv vilka datastrukturer du använder, samt den algoritm du använder för att lösa problemet. Beskriv gärna algoritmen i punktform eller med pseudokod. Du behöver inte detaljerat beskriva sådant som finns i standardbiblioteket (men nämn gärna kort hur det du använder fungerar).

(b) Vilken tidskomplexitet har din algoritm? Uttryck tidskomplexiteten i termer av antal objekt (o) och totalt antal pekarvariabler (p). Motivera ditt svar. (1)

(c) Efter att ha analyserat de objekt som var minnesläckor inser du att de obekten innehöll viktig data som programmet borde ha haft koll på. De beror alltså på en logisk bugg i programmet, snarare än att någon helt enkelt har glömt att anropa `delete`. (4)

Ett enkelt sätt att undvika att objekten blir minnesläckor skulle exempelvis vara att lägga till en global `vector` som pekar på alla dessa objekt. Detta är dock inte en bra idé i allmänhet eftersom det bara löser symptomen, och inte det egentliga problemet.

I stället vill du skriva ett program som utifrån lösningen från (a) föreslår ett mindre antal pekarvariabler som behöver läggas till för att alla objekt som var minnesläckor ska bli nåbara från lokala och globala variabler. Om möjligt så vill du att dessa pekare ska ligga i andra objekt, snarare än att lägga till nya lokala eller globala variabler. Notera att din lösning behöver inte hitta det minimala antalet variabler att lägga till, men ska hitta en mycket bättre lösning än att lägga till nya variabler för varje läckt objekt.

Beskriv vilka datastrukturer du använder, samt den algoritm du använder för att lösa problemet. Beskriv gärna algoritmen i punktform eller med pseudokod. Du behöver inte detaljerat beskriva sådant som finns i standardbiblioteket (men nämn gärna kort hur det du använder fungerar).