Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Filip Strömbäck och Magnus Nielsen

2025-08-26

TDDI16
Datastrukturer och algoritmer
Datortentamen (DAT1)
2025-08-26, 08–12

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.
- Lämna in en fil per numrerad uppgift.
- 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:	20	59	11	5	16	43	9	35	73	44	82	74	85	37	63	84
1:	5	11	20	59	9	16	35	43	44	73	74	82	37	63	84	85
2:	5	11	20	59	16	43	9	35	73	44	82	74	85	37	63	84
3:	5	9	11	20	16	43	59	35	73	44	82	74	85	37	63	84
4:	74	73	63	59	44	43	37	35	9	5	16	11	20	82	84	85
5:	20	35	11	5	16	9	37	59	73	44	82	74	63	43	84	85
Sorterad:	5	9	11	16	20	35	37	43	44	59	63	73	74	82	84	85

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) Insertionsort (1)

(c) Selectionsort (1)

(d) Heapsort (1)

(e) Mergesort (vi räknar ”nivåer” av merge-operationer, första nivån merge:ar listor med 1 element, andra med 2 element, tredje med 4 element och så vidare) (1)

2. **Hashtabeller** (4 p)

Vi har en hashtabell med linjär adressering och några element instoppade. Hashfunktionen är h(x) = x mod size (arrayindex står under arrayen för tydlighets skull).

10	8	30	Null	Null	15	6	7	17	29
0	1	2	3	4	5	6	7	8	9

(a) I vilken ordning kan elementen ha stoppats in? Det finns flera lösningar. Ge fyra korrekta lösningar för maxpoäng. (2)

(b) Om vi tar bort 17 från den delvis fyllda hashtabellen ovan (remove / delete), hur kommer tabellen se ut? Det finns flera olika lösningar. Ge två olika lösningar som inte hashar om hela tabellen utom möjligen i värsta fallet, och förklara dem, för maxpoäng. (2)

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 = 0; i*i < n; i++) {
    result += i;
  }
  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 1;
  }
}
```

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

```
int fun(int n, int m) {
  int result = f(n);
  result = result + fn(m);
  for (int i = 1; i < n; i = i * 2) {
    result++;
  }
  for (int i = 1; i < m; i = i * 2) {
    result++;
  }
  return result;
}
```

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

```
bool findAndRemove(vector<int> arr, int value) {
  for (int i = 0; i < arr.size(); i++) {
    if (arr.at(i) == value) {
      // Tar bort elementet på plats 'i'.
      arr.erase(arr.begin() + i);
      return true;
    }
  }
  return false;
}
```

4. **Linjära intervall** (9 p)

Ett problem som ibland förekommer i CAD-mjukvara (Computer Aided Design) är att hålla koll på vilka delar av ett intervall som täcks av någon geometri. Exempelvis är detta användbard för rendering eller för att hitta vad användaren har klickat på.

Du har fått i uppgift att designa en lämplig datastruktur för det 1-dimensionella fallet. Datastrukturen ska innehålla en samling *intervall*. Varje intervall består av två heltal, (s, e), där s är koordinaten där intervallet startar och e är koordinaten där intervallet slutar. Vi betraktar intervallet som halvöppet, vilket innebär att det täcker alla punkter x där $s \leq x < e$. Intervallet vi lagrar som $(1, 3)$ täcker alltså punkterna 1 och 2, men inte 3. Datastrukturen antar att inga intervall överlappar med varandra.

För att minimera minnesanvändningen av din datastruktur planerar du att lagra intervallen i en dynamisk array, `intervals`, där varje element är ett par (s, e) som beskrivet ovan. Datastrukturen innehåller också en booleansk variabel, `sorted`.

Du implementerar datastrukturen enligt nedan:

`create()`: Skapar en ny tom datastruktur:

1. Initiera `sorted` till en tom array, och sätt `sorted` till `true`.

`insert(i)`: Sätter in ett nytt intervall i datastrukturen:

1. Sätt `sorted` till `false`.
2. Sätt in intervallet i sist i `intervals`.

`find(x)`: Hittar och returnerar det intervall som innehåller punkten x, eller null om inget intervall innehåller punkten.

1. Om `sorted` är `false`:
   1.1. Sortera `intervals` med hjälp av heapsort. Första elementet i varje par (s) används som nyckel, och sorteras i stigande ordning.
   1.2. Sätt `sorted` till `true`.
2. Använd binärsökning för att hitta det intervall vars startpunkt är närmast mindre än x, lagra det som f. Om inget sådant finns, returnera null.
3. Om $f.e \geq x$, returnera f. Annars, returnera null.

Svara på följande frågor. Motivera dina svar.

(a) Vilken tidskomplexitet har `insert` och `find` ovan? Uttryck tidskomplexiteten i termer av antal intervall i datastrukturen, n. (2)

(b) Antag att datastrukturen används av ett program som först sätter in n intervall i datastrukturen och sedan anropar `find` n gånger. Vad blir den totala tidskomplexiteten uttryckt i n? (2)

(c) Antag att datastrukturen används av ett program som i stället anropar `find` mellan varje insättning. Det vill säga, programmet sätter in ett intervall och anropar sedan `find`. Detta görs n gånger. Vad blir den totala tidskomplexiteten i detta fallet, uttryckt i n? (2)

(d) Föreslå en alternativ implementation av datastrukturen som förbättrar den totala tidskomplexiteten för åtminstone ett av programmen beskrivna i (b) och (c), och bibehåller tidskomplexiteten i det andra. Finns det någon nackdel med din implementation? (3)

Exempelvis: Kommer du fram till att (b) blir $\mathcal{O}(n^5)$ och (c) blir $\mathcal{O}(n^2)$ kan du föreslå en implementation som är $\mathcal{O}(n^4)$ för (b) men bibehåller $\mathcal{O}(n^2)$ för (c).

5. **Värmeberäkningar** (6 p)

I sommar har det bitvis varit varmt i delar av Sverige. När det är varmt länge kallas det för en värmebölja. SMHI använder följande klimatologiska definition på en värmebölja: ”en sammanhängande period då dygnets högsta temperatur är minst 25.0°C minst fem dagar i sträck.”[1] I den här uppgiften betraktar vi att ett dygn börjar klockan 00:00 och slutar klockan 23:59.[2]

Du har nyligen fått tag på historisk temperaturdata för mätstationer över hela Sverige (från 1950-talet och framåt), och du är nyfiken på när olika delar av Sverige har haft värmebölja. Som du ser i tabellen nedan så är datan för detaljerad för att du enkelt ska analysera den själv. Därför vill du i ett första steg skriva ett program som utifrån mätdata för *en* stad beräknar när det har varit värmebölja i den staden. Indata ser ut som nedan (du kan anta att den är sorterad efter datum och tid):

Datum	**Tid**	**Temperatur**
...	...	...
2025-06-01	00:00	18.1°C
2025-06-01	00:05	19.4°C
2025-06-01	00:10	19.3°C
...	...	...
2025-06-01	12:58	24.9°C
2025-06-01	13:02	25.0°C
2025-06-01	13:07	24.9°C
...	...	...
2025-06-01	23:56	19.2°C
2025-06-02	00:01	18.1°C
...	...	...

I tabellen ovan kan vi se att den maximala temperaturen 2024-06-01 var 25.0°C (klockan 13:02). Problemet är alltså att hitta alla värmeböljor i indata. Alltså, sekvenser av minst 5 dagar där maxtemperaturen har varit minst 25.0°C grader. Exempelvis skulle utdata från programmet se ut som följer:

- ...
- Värmebölja från 2025-06-01 till 2025-06-10
- Värmebölja från 2025-07-20 till 2025-08-01
- ...

(a) Beskriv hur du kan implementera ett program som analyserar den mätdata du har. Programmet får data för *en* stad enligt ovan och ska producera en lista över intervall då det har varit värmebölja (exempelvis genom att skriva ut dem på formatet ovan, eller en lista med par). (4)

Beskriv vilka **datastrukturer** du använder, samt den **algoritm du använder** för att lösa problemet. Beskriv gärna algoritmen kortfattat **i punktform** eller med **pseudokod**. Du behöver inte detaljerat beskriva sådant som finns i standardbiblioteket (men nämn kort vilken algoritm/datastruktur som du anser används av standardbiblioteket).

(b) Vilken tidskomplexitet har ditt program uttryckt i antal antal mätpunkter (n). Motivera ditt svar. Vid behov kan du också använda d som antal dagar som mätdatan täcker. (1)

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

[1] `https://www.smhi.se/kunskapsbanken/meteorologi/temperatur/varmebolja`
[2] Detta motsvarar inte riktigt SMHI:s definition, men duger för uppgiften.

6. **Vattendistribution** (10 p)

På grund av låga grundvattennivåer har brunnarna i en by på Östgötaslätten börjat torka ut (varje hus har en egen brunn). För att lösa problemet har kommunen beslutat att ansluta de som har problem med dricksvatten till det kommunala vattennätet genom att gräva ner vattenledningar längs de vägar som redan finns i byn. Givetvis vill kommunen minimera sträckan väg som behöver grävas upp. Till sin hjälp har de producerat en tabell med alla vägar enligt nedan. Givetvis kan vatten flöda åt godtyckligt håll i vattenledningarna.

Från	**Till**	**Sträcka (m)**
Grengatan 1	Grengatan 3	50
Grengatan 3	Grengatan 5	53
Grengatan 1	Trädgatan 1	150
Trädgatan 1	Trädgatan 3	50
...	...	...

Notera: Tabellen ovan är sorterad för läsbarhet. Du kan inte anta att raderna finns i någon speciell ordning.

(a) Kommunen har bett dig bygga ett program som hjälper kommunen att planera hur de ska dra nya vattenledningar till ett hus vars brunn har torkat ut. Som indata får programmet: tabellen ovan, namnet på det hus som ska anslutas, och namnen på de hus som redan är anslutna till kommunalt vatten. Programmet ska sedan beräkna vilka vägar som behöver grävas upp för att installera ny vattenledning. Kommunen vill givetvis minimera sträckan som behöver grävas upp. Dels för att inte vara ivägen för invånarna, dels för att minimera mängden rör de behöver köpa in. (4)

Du kan anta att minst ett hus redan är anslutet till kommunalt vatten, och att det går att ta sig mellan alla par av hus via de vägar som finns i tabellen.

Beskriv vilka **datastrukturer** du använder, samt den **algoritm du använder** för att lösa problemet. Beskriv gärna algoritmen kortfattat **i punktform** eller med **pseudokod**. Du behöver inte detaljerat beskriva sådant som finns i standardbiblioteket (men nämn kort vilken algoritm/datastruktur som du anser används av standardbiblioteket).

(b) Vilken tidkomplexitet har din algoritm i (a)? Uttryck tidskomplexiteten i termer av antal hus h och antal vägar v. Motivera ditt svar. (1)

(c) En bit in i sommaren inser kommunen att de flesta i byn kommer att behöva anslutas till kommunalt vatten. Därför tänker de inte längre vänta på att vissa brunnar torkar ut, utan i stället ansluta alla direkt i förhoppningen om att det blir mer effektivt än att ta husen ett och ett. (4)

Skriv ett program som hjälper kommunen med detta. Programmet får som indata tabellen ovan och en lista med vilka hus som redan är anslutna till kommunalt vatten. Programmet ska sedan producera en lista med de vägar som behöver grävas upp för att *alla* hus ska bli anslutna till vattennätet. Likt tidigare vill du såklart minimera sträckan som behöver grävas upp.

Beskriv vilka **datastrukturer** du använder, samt den **algoritm du använder** för att lösa problemet. Beskriv gärna algoritmen kortfattat **i punktform** eller med **pseudokod**. Du behöver inte detaljerat beskriva sådant som finns i standardbiblioteket (men nämn kort vilken algoritm/datastruktur som du anser används av standardbiblioteket).

(d) Vilken tidkomplexitet har din algoritm i (c)? Uttryck tidskomplexiteten i termer av antal hus h och antal vägar v. Motivera ditt svar. (1)