DM534 — Øvelser Uge 40

Introduktion til Datalogi, Efterår 2021

Jonas Vistrup, med tilføjelser af Rolf Fagerberg

---

1 I

1.1

Opskriv i pseudokode algoritmen Sequential Search ved hjælp af operationerne **readNext()**, **isEndOfFile()**, **open()** og **close()** fra interface'et sekventiel tilgang.

SVAR version 1:

```
SequentialSearchV1(filename, x)
  file = open(filename)
  i = 0
  while not file.isEndOfFile()
    if file.readNext() == x
        file.close()
        Return i
    i = i + 1
  file.close()
  Return "Not Found"
```

SVAR version 2:

```
SequentialSearchV2(filename, x)
  file = open(filename)
  i = 0
  while not file.isEndOfFile() && file.readNext() !=x
    i = i + 1
  if file.isEndOfFile()
    file.close()
    Return "Not Found"
  else
    file.close()
    Return i
```

1.2

Opskriv pseudokode algoritmen for merge af to lister ved hjælp af operationerne **readNext()**, **isEndOfFile()**, **writeNext(data)**, **open()** og **close()** fra interface'et sekventiel tilgang.

SVAR:

```
Merge(filename1, filename2, outputname)
  file1 = open(filename1)
  file2 = open(filename2)
  output = open(outputname)

  empty1 = false
  empty2 = false
  if file1.isEndOfFile()
    empty1 = true
  else
    x = file1.readNext()
  if file2.isEndOfFile()
    empty2 = true
  else
    y = file1.readNext()

  while not empty1 and not empty2
    if x < y
        output.writeNext(x)
        if file1.isEndOfFile()
            empty1 = true
        else
            x = file1.readNext()
    else
        output.writeNext(y)
        if file2.isEndOfFile()
            empty2 = true
        else
            y = file2.readNext()

  if empty2
    output.writeNext(x)
    while not file1.isEndOfFile()
        output.writeNext(file1.readNext())

  if empty1
    output.writeNext(y)
    while not file2.isEndOfFile()
        output.writeNext(file2.readNext())
```

1.3

I denne opgaver repræsenterer vi mængder som sorterede lister uden dubletter. For eksempel vil de to mængder $A = \{5, 3, 9, 8\}$ og $B = \{3, 2, 9, 10, 27\}$ være repræsenteret som disse sorterede lister:

$$A = [3, 5, 8, 9]$$

$$B = [2, 3, 9, 10, 27]$$

Beskriv en algoritme til at beregne repræsentationen af foreningsmængden $X \cup Y$ ud fra repræsentationen af to mængder X og Y.

SVAR:

Brug Merge med følgende ændring: Hvis et merge-step ser to ens elementer som forreste i A og B, flyttes det ene over sidst i C og det andet smides væk. Hvis et merge-step ser to forskellige elementer som forreste i A og B, flyttes det mindste over sidst i C.

Alle elementer i A og B vil optræde i C. For elementer, som findes i både A og B, vil kun én af de to dubletter optræde i C (der kan kun være to dubletter i alt af hvert element, da A og B hver især er uden dubletter). Derved bliver C en sorteret liste uden dubletter, som krævet for at passe med vores repræsentation af mængder.

1.4

Beskriv en algoritme til at flette (merge) indholdet af tre sorterede lister A, B og C sammen til én sorteret liste D. Hvad er køretiden for din algoritme?

SVAR:

Et merge-skridt ændres til følgende: Sammenlign (nuværende) forreste element i A, B, og C, og flyt det mindste af disse over sidst i D. Dette gøres, sålænge ingen af tre lister A, B, C er blevet tomme. Når én af dem bliver tom, fortsættes som for normal (to-vejs) merge med de to resterende lister.

Køretid er $O(|A| + |B| + |C|)$, da der i hvert merge-skridt flyttes ét element over i C.

1.5

Givet en algoritme til at flette indholdet af tre sorterede lister A, B og C sammen til én sorteret liste D, beskriv en variant af Mergesort baseret på denne. Hvad er køretiden for din algoritme?

SVAR:

Iterativ Mergesort: Merge n lister af længde 1 sammen tre og tre. Det giver $n/3$ sorterede lister af længde 3. Merge disse sammen tre og tre, hvilket giver $n/3^2$ lister af længde 3^2. Efter i runder har listerne længde 3^i. Fortsæt til der er 1 sorteret liste af længde n. Da $3^i = n \Leftrightarrow i = \log_3 n$, tager dette $\log_3 n$ runder. Hver runde tager $O(n)$ tid (da der flyttes ét element per merge-step i hver merge i laget). Så samlet køretid er $O(n \log_3(n)) = O(n \log(n))$.

Rekursiv Mergesort: I stedet for at lave rekursive kald på to cirka lige store halvdele, så kan der nu laves tre rekursive kald på tre cirka lige store dele, som kan merges (med tre-vejs merge) efter at de tre rekursive kald har sorteret dem. Køretid er igen (da der laves omtrent de samme merges som for den iterative version) $O(n \log_3(n)) = O(n \log(n))$.

1.6

Hvis en hashfunktion h er givet ved $h(x) = x \mod 11$, på hvilke pladser i tabellen ender tallene $25, 75, 125, 175$?

SVAR:

- $25 \mod 11 = 3$.
- $75 \mod 11 = 9$.
- $125 \mod 11 = 4$.
- $175 \mod 11 = 10$.

1.7

Hvis en hashfunktion h er givet ved $h(x) = x \mod 11$, hvor mange pladser i hashtabellen har mere end ét element, når der indsættes elementerne $34, 65, 122$ og 155?

SVAR:

- $34 \mod 11 = 1$.
- $65 \mod 11 = 10$.
- $122 \mod 11 = 1$.
- $155 \mod 11 = 1$.

Kun plads 1 har mere end ét element, så svaret er én plads.

1.8

Beregn med lommeregner svaret på følgende: Hvis 3 elementer indsættes tilfældigt i et array med 7 pladser, hvad er sandsynligheden for, at der ikke er to elementer som ender på samme plads?

SVAR: $1 \cdot 6/7 \cdot 5/7 \approx 0.612$.

1.9

Beregn med lommeregner følgende svaret på følgende: Hvis 5 elementer indsættes tilfældigt i et array med 12 pladser, hvad er sandsynligheden for, at der ikke er to elementer som ender på samme plads?

SVAR: $1 \cdot 11/12 \cdot 10/12 \cdot 9/12 \cdot 8/12 \approx 0.382$.

2 II

2.1

(a) Beskriv en algoritme `sortedInsert(A,x)`, der som input tager en sorteret liste `A` og som output returnerer en sorteret liste indeholdende elementerne i `A` tilføjet elementet `x`.

SVAR:

Lav en liste `B`, som kun indeholder `x`. Den er automatisk sorteret. Derfor kan man køre Merge på `A` og `B`, og dette vil generere det ønskede output. Der laves $|\texttt{A}|+1$ mergeskridt i alt, dvs. at køretiden er $O(|\texttt{A}|)$.

Alternativ formulering: Løb gennem `A`. Kopier elementer til en ny liste, sålænge man passerer elementer mindre end `x`. Derefter indsættes `x` i den nye liste. Derefter kopieres resten af elementerne i `A`. Køretiden er stadig $O(|\texttt{A}|)$.

(b) Beskriv, hvordan man kan lave en sorteringsalgoritme `Insertionsort` via gentagen brug af `sortedInsert()` fra forrige opgave.

SVAR:

Løb gennem input-listen. Ved det første element, opret en liste `A`, som indeholder kun dette element (og derfor er sorteret). Ved resten af elementerne, kald `sortedInsert(A,x)`, hvor `x` er det element, man er kommet til. Returner til sidst `A`. Da `A` hele tiden sorteret og til sidst indeholder alle elementer, er algoritmen korrekt (dvs. den sorterer input).

(c) Hvad er den asymptotiske køretid for `Insertionsort`?

SVAR:

Da længden af `A` stiger med en for hvert skridt gennem input, giver køretiden fra (a) at den samlede køretid for `Insertionsort` er proportional med

$$1 + 2 + 3 + \cdots + (n-2) + (n-1) + n$$

Den sum er på slides vist at være $O(n^2)$. Dette er derfor køretiden for `Insertionsort`.

2.2

Hvis en hashfunktion h er givet ved $h(x) = x \mod 17$, på hvilke pladser i tabellen ender tallene $22, 72, 122, 172$?

SVAR:

- $22 \mod 17 = 5$.
- $72 \mod 17 = 4$.
- $122 \mod 17 = 3$.
- $172 \mod 17 = 2$.

2.3

Hvis en hashfunktion h er givet ved $h(x) = x \mod 17$, hvor mange pladser i hashtabellen har mere end ét element, når der indsættes elementerne $40, 74, 101 og 159$?

SVAR:

- $40 \mod 17 = 6$.
- $74 \mod 17 = 6$.
- $101 \mod 17 = 16$.
- $159 \mod 17 = 6$.

Kun plads 6 har mere end ét element, så svaret er én plads.

2.4

Lav et program med input n og k der for situationen hvor n elementer indsættes tilfældigt i et array med k pladser finder sandsynligheden for, at der ikke er to elementer som ender på samme plads.

SVAR version 1:

```
def noCollisionsRecursive(n,k):
    if n == 1:
        return 1.0
    else:
        return noCollisionsRecursive(n-1,k) * (k-(n-1))/k
```

SVAR version 2:

```
def noCollisionsIterative(n,k):
    s = 1.0
    for i in range(1,n):
        s = s*(k-i)/k
    return s
```

2.5

Hvis 200 elementer indsættes tilfældigt i et array med 100.000 pladser, hvad er sandsynligheden for, at der ikke er to elementer som ender på samme plads?

SVAR: Cirka 0.819.

2.6

Hvis n elementer indsættes tilfældigt i et array med 100.000 pladser, hvor stor skal n være for at sandsynligheden for, at der ikke er to elementer som ender på samme plads, bliver mindre end 1/2?

SVAR: Ved n = 373.

2.7

[Udfordrende] Beskriv en algoritme, der som input tager et tal K og to sorterede lister X og Y, hver med n tal, og finder ud af, om der findes et par at tal $x \in X$ og $y \in Y$ for hvilke $x + y = K$. Din algoritme skal køre i tid $O(n)$. Du skal argumentere for køretiden og for korrektheden af svaret (det sidste kan gøres med en invariant).

SVAR:

Først prøv det mindste tal i X og det største tal i Y. Hvis summen er lig K, terminer. Ellers hvis summen er mindre end K, så kan det mindste tal i X aldrig være x'et. Der kigges nu på resten af de mulige tal i X og den mindste af dem tages (andenmindste af X). Det omvendte gøres for Y, hvis summen er større end K. Dette forsætter rekursivt indtil x og y parret findes eller alle tal bliver udelukket som mulige x'er og mulige y'er.

Invariant: Hvis der findes et $x + y = K$, så ligger x i $X[i..n]$ og y ligger i $Y[1..j]$[1].

Basis: Ved start er $i = 1$ og $j = n$, så mulig x ligger i $X[1..n]$ og mulig y ligger i $Y[1..n]$ ellers findes der ikke en løsning. Dette er trivielt sandt.

Induktionskridt: Der er to cases, hvor løkken ikke terminerer. Hvis $X[i] + Y[j] < K$, så er $X[i]$ ikke længere en mulig x, da der ikke findes noget mulige y som er større end $Y[j]$. Altså incrementeres i, så $i' = i + 1$. De mulige x værdier er nu $X[i..n]$ per induktions antagelsen, undtagen $X[i]$, altså er de mulige x værdier $X[i + 1...n] = X[i'..n]$.

Det samme bevis kan laves for $X[i] + Y[j] > K$.

Siden løkken kun terminerer når $x + y = K$ er fundet eller også er $i = n + 1$ eller $j = 0$, som betyder der ikke er et muligt x eller et muligt y.

Køretiden er $O(n)$, da hvert skridt eliminerer et tal (eller terminerer), og siden der er $2n$ tal i alt, så må køretiden være $O(2n) = O(n)$

[1]Her er i og j indekser på de elementer, som vi er kommet til, dvs. at det er $X[i] + Y[j]$ og K, som vi sammenligner næste gang.