

Studieretningsprojekt 2022

| |
|--------------------------------|
| Elev id: 3x 06 |
| Elev: Balder Westergaard Holst |

| | |
|--------------|--|
| Fag: | Vejleder: |
| Matematik A | Jens Christian Larsen Tlf.: 26279734 Email: jl@soroeakademi.dk |
| Informatik C | Kristian Kjeldgaard Hoppe Tlf.: 28825466 Email: kh@soroeakademi.dk |

| |
|---|
| Område: Store-O notation og sorteringsalgoritmer |
| <p>Opgaveformulering:</p> <p>Gør rede for kompleksitetsklasserne givet ved Store-O notation, idet du giver eksempler på repræsentanter af forskellige klasser.</p> <p>Beskriv sorteringsproblemer generelt, og forklar algoritmerne for insertion- og merge-sort. Vis at insertion-sort har en værste-tilfælde-udførelse på $O(n^2)$.</p> <p>Implementer algoritmerne insertion- og merge-sort i Python og undersøg den faktiske udførelsestid af de to algoritmer på lister af tilfældige tal af forskellig størrelse. Vurder algoritmernes udførelsestid.</p> <p>Indfør begrebet træ, og bevis relevante sætninger om træer, især højden af et træ. Bevis, at en vilkårlig sorteringsalgoritme baseret på sammenligning, vil være nedadtil begrænset af i dens værste-tilfælde-udførelsestid med $n \cdot \lg(n)$ (Stirlings formel må antages).</p> <p>Insertion-sort har en bedste-tilfælde-udførelsestid på $O(n)$: Hvorfor er det ikke et modbevis til sætningen om at den nedre grænse for sorteringsalgoritmer er $n \cdot \lg(n)$?</p> <p>Omfang 15-20 ns.</p> |

Sorteringsalgoritmer og O-notation

Matematik A og Informatik C

Vejledere: Jens Christian Larsen og Kristian Kjeldgaard Hoppe

Balder Westergaard Holst

25. marts 2022

Indhold

| | | |
|----------|------------------------------------|----------|
| 1 | Indledning | 3 |
| 2 | Sorteringsalgoritmer | 4 |
| 2.1 | Hvad er sortering? | 4 |
| 2.2 | Insertionsort | 5 |
| 2.3 | Mergesort | 7 |
| 2.4 | Analyse af Insertionsort | 7 |
| 3 | Resume | 8 |
| 4 | Konklusion | 9 |

Figurer

| | | |
|-----|---|---|
| 2.1 | Eksempel på sortering af en liste | 4 |
| 2.2 | Pseudokode til insertionsort | 6 |
| 2.3 | Indsæt element først i listen | 6 |

1 | Indledning

Dette er min indledning pt.

2 | Sorteringsalgoritmer

2.1 Hvad er sortering?

Sortering er helt lavpraktisk at sætte en mængde data en rækkefølge på baggrund af dataens attributter. Det kunne f.eks. være alfabetisk eller efter farve eller størrelse. Det er dog ikke entydigt hvilken sorteringsstrategi der ville være hurtigst. Disse sorterings-strategier kan også kaldes sorterings-algoritmer, det er ikke kun mennesker der kan benytte sorterings-algoritmer til f.eks. at sortere kort, men computere kan også, og de gør det for det meste også hurtigere. De næste to afsnit omhandler to forskellige sorterings-algoritmer, og deres måde at sortere en liste med tal.

Herfra vil jeg kun forholde mig til sortering af lister med tal, og sortere dem på baggrund af deres størrelse (se figur 2.1)

$$[4,2,5,3,1] \longrightarrow [1,2,3,4,5]$$

Figur 2.1: Eksempel på sortering af en liste

2.2 Insertionsort

Insertionsort er en af de mere simple sorteringsalgoritmer. Dette er pseudokode for algoritmen [1, s. 104]. Det er ikke en tilfældighed at denne algoritme hedder insertionsort. Den fungerer nemlig ved at gennemgå hvert element i listen, og placere det hvor det passer ind i de elementer der allerede er sorteret. Dette er nok til dels den måde man f.eks. ville sortere sin hånd i Uno eller 500. At algoritmen gennemgår alle elementerne i listen kronologisk kan vi se allerede i linje 2, da algoritmen her begynder med en for-lykke, der tæller for hvert element i listen, dog starter den ved 2. element i listen. For at gøre koden mere læsbar sættes elementet som algoritmen er nået til ind i variabelen *e* i linje 3. Det næste algoritmen gør, er at checke om elementet har en mindre værdi end det første element i listen, hvis dette er sandt rykkes rykkes alt før elementet et tak til højre, og elementet sættes ind først i listen se figur 2.3

[balder](#)

(se afsnit 2.4 for store-O analysen)

```
1  funktion insertionsort(liste) {
2      for i = 1 til i = n {          # her er n længden af listen
3          e = liste[i]                # dette er elementet i listen
4
5          if e < liste[0] {            # hvis elementet er større end det første element i listen
6              for j = j til j = 0{
7                  liste[j] = liste[j-1] # ryk elementerne på pladserne 0 til j, et tak frem
8              }
9              liste[0] = e              # sætter dette element forrest i listen
10             }
11         else{
12             j = i                    # j er en nu tæller der starter på i
13             while liste[j-1] > e {    # kør mens liste[j-1] er større end elementet
14                 liste[j] = liste[j - 1] # ryk liste[j] et tak til højre
15                 j -= 1
16             }
17             liste[j] = element        # indsæt elementet hvor det passer ind
18         }
19     }
20     return(liste)                    # returnerer den sorterede liste
21 }
```

Figur 2.2: Pseudokode til insertionsort

[5,7,9,4,10,6,2]

[5,5,7,9,10,6,2]

[4,5,7,9,10,6,2]

Figur 2.3: Indsæt element først i listen

2.3 Mergesort

2.4 Analyse af Insertionsort

3 | Resume

Dette er hvad jeg har skrevet og fundet ud af.

4 | Konklusion

Her er mine konkluderende sætninger

Litteraturliste

1. Dietzfelbinger, M. & Mehlhorn, K. *Algoritmer og datastrukturer* <https://github.com/thorehusfeldt/algoritmer-og-datastrukturer/blob/master/ad-book.pdf> ().

Bilag 1 - Algoritmer og Datageneration

Insertionsort Algoritmen

```
1 def insertionsort(l):
2     for i in range(1,len(l)):
3         element = l[i]
4
5         if element < l[0]:
6             for j in range(i,0,-1):
7                 l[j] = l[j-1]
8                 l[0] = element
9         else:
10            j = i
11            while(l[j-1]>element):
12                l[j] = l[j-1]
13                j -= 1
14            l[j] = element
15    return(l)
```

Mergesort Algoritmen

```
1 def mergesort(l):
2     if len(l) <= 1:
3         return(l)
4     else:
5         return(merge(mergesort(l[:len(l)//2]),mergesort(l[len(l)//2:])))
6
7 def merge(a,b):
8     c = []
9     while True:
10        if (len(a) == 0):
11            return(c + b)
12        elif (len(b) == 0):
13            return(c + a)
14        elif (a[0] <= b[0]):
15            c.append(a[0])
16            a.pop(0)
17        else:
18            c.append(b[0])
19            b.pop(0)
```

Kode til test af algoritmerne

```
1 import random
2 import time
3 import pandas as pd
```

```
4 import os
5
6 import sys
7 sys.path.insert(1, './algoritmer')
8 from mergesort import *
9 from insertionsort import *
10
11 # Denne funktion timer køretiden af en funktion med input l og returnerer funktionen køretid i
12 # millisekunder
13 def test(fun,l):
14     start_time = time.perf_counter()
15     fun(l)
16
17     return(time.perf_counter() - start_time)
18
19 # Denne funktion returnerer en liste af tilfældige tal mellem 0 og 1000, med n elementer
20 def createRandomList(n):
21     return([random.randint(0,1000) for i in range(n)])
22
23 # Laver en mappe i filsystemet hvis der ikke allerede er en med stien
24 def makeIfNeeded(dir_path):
25     if(os.path.isdir(dir_path) == False):
26         print(f"made dir: {dir_path}")
27         os.mkdir(dir_path)
28     return(dir_path)
29
30 # Finder det næste versionsnummer for til navngivning af fil på baggrund af indholdet i en folder
31 def newVersionNumber(dir_path,extention):
32     file_names = os.listdir(dir_path)
33     version = 0
34
35     thisfilename = f"{version}{extention}"
36
37     while(thisfilename in file_names):
38         version += 1
39         thisfilename = f"{version}{extention}"
40
41     return(thisfilename)
42
43 # Dette er funktionen der tester en liste med funktioner og gemmer deres køretider
44 def fullTest(functions):
45
46     # hvor mange datapunkter pr. n-værdi
47     trials = 10
48
49     data_dir = "../data/"
50     version_number = newVersionNumber(data_dir,"")
51
52     seed = time.time()
53     print(f"Seed: {seed}")
54
55     for function in functions:
56
57         # i denne liste gemmes antallet af elementer at den liste som algoritmen sorterer for hvert
58         # datapunkt.
59         ns = []
60         # i denne liste gemmes den tid det tager at sorterer listen med n elementer
61         times = []
62
63         # Bruger det samme seed til test at hver algoritme. på den måde er det de samme
64         # pseudo-tilfældige liste som algoritmerne sorterer
65         random.seed(seed)
66
67         # Vi laver testen et antal (trials) gange pr. n-værdi
68         for trial in range(0,trials):
```

```
67
68     # En lykke der køre et abitrært antal gange (jo højere en i-værdi jo højere maks antal
69     elementer i listen)
70     for i in range(0,80):
71
72         # Jeg bruger en potensfunktion til at fa flere datapunkter tættere på y-aksen og
73         færre lange operationer (pga. lange liste)
74         n = round(pow(1.1,i))
75
76         print(f"function=\"{function.__name__}\": Trial: [{trial+1}/{trials}] {i=},{n=}") #
77         lidt feedback
78
79         # gennererer en tilfældig liste
80         l = createRandomList(n)
81
82         # gem størrelsen af listen der skal sorteres
83         ns.append(n)
84         # gen den tid det tager at sortere listen
85         times.append(test(function,l))
86
87
88     data = {
89         "n": ns,
90         "t": times
91     }
92
93     version_dir = makeIfNeeded(data_dir + version_number + "/")
94     algorithm_dir = makeIfNeeded(version_dir + function.__name__ + "/")
95     full_path = algorithm_dir + newVersionNumber(algorithm_dir, ".csv")
96
97     print(f"\ndata saved to \"{full_path}\"")
98
99     pd.DataFrame(data).to_csv(full_path,index = False)
100
101 if __name__ == "__main__":
102     functions = [mergesort,insertionsort]
103
104     fullTest(functions)
```

Bilag 2 - Databehandling og Plots

Kode til databehandling og generering af plots

```
1 library(ggplot2)
2 #library(tikzDevice)
3
4 #working dir
5 setwd("/home/Balder/Documents/Skole/Gym/SRP/data/5")
6
7
8 #import data
9 dataset <- read.csv("mergesort/0.csv",header=TRUE,sep=",")
10
11 #dir = "1"
12 algorithm_dirs = list.files()
13
14 M = NULL
15 for (j in 1:length(algorithm_dirs)){
16   algorithm_dir = algorithm_dirs[j]
17
18   files = list.files(algorithm_dir)
19
20   for (i in 1:length(files)){
21     file_path = paste(algorithm_dir,files[i],sep="/")
22     print(file_path)
23     m = read.csv(file_path,header=TRUE,sep=",")
24     m$Algorithm = algorithm_dirs[j]
25     M = rbind(M, m)
26   }
27 }
28
29
30 M$algorithm = factor(M$Algorithm)
31 summary(M)
32
33 # punktmængder for hver algoritme
34 m_merge = subset(M,M$algorithm=="mergesort")
35 m_insertion = subset(M,M$algorithm=="insertionsort")
36
37 # laver modeller
38 model_merge = nls(t~a*n*log2(n), data=m_merge, start=list(a=0.000001))
39 model_insertion = nls(t~a*n^2 + b*n + c, data=m_insertion, start=list(a=1,b=1,c=1))
40
41
42 # Sætter ny path til hvor outputtet skal være
43 setwd("/home/Balder/Documents/Skole/Gym/SRP/img")
44
45 # gemmer r2-værdierne i to filer
46 writeLines(toString(round(with(m_merge,cor(t,n)),digits=3)),"r2-merge.txt")
47 writeLines(toString(round(with(m_insertion,cor(t,n)),digits=3)),"r2-insertion.txt")
48 print("r2 saved to files")
49
```



```
50
51 # laver modelerede v?rdier for hver n
52 m_merge$model = predict(model_merge)
53 m_insertion$model = predict(model_insertion)
54
55 m_merge$residual = resid(model_merge)
56 m_insertion$residual = resid(model_insertion)
57
58 # kombinerer de to
59 M = rbind(m_merge,m_insertion)
60
61 summary(M)
62
63
64 ggplot(M, aes(x=n, y=t, colour=Algorithm)) +
65   geom_point(size=1.5,alpha=0.1,shape=19) +
66   geom_line(aes(x=n, y=model,color=Algorithm), size=2, alpha=0.6) +
67   labs(title="To Sorteringsalgoritmer") +
68   theme(legend.position = c(.9, .9)) + # virker ikke!!
69   guides(colour = guide_legend(override.aes = list(alpha = 1))) + # lav legend alpha 1
70   theme_bw()
71
72 ggsave("toAlgoritmer.png")
73
74 ggplot(M, aes(x=log10(model), y=residual, colour=Algorithm)) +
75   geom_point(size=1.5,alpha=0.1,shape=19) +
76   labs(title="Residualer") +
77   facet_wrap(~algorithm,scales="free",ncol=1) +
78   theme_bw() +
79   theme(legend.position="none")
80 ggsave("toAlgoritmerResidual.png")
81
82 C = data.frame(
83   Algorithm = unique(M$Algorithm),
84   R2 = c(with(m_merge, cor(t,n)), with(m_insertion, cor(t,n)))
85 )
86
87 write.table(C, "r2.txt", quote=FALSE,sep="\t", row.names=FALSE)
```
