Cheatography

Regular Expressions Cheat Sheet

by Dave Child (DaveChild) via cheatography.com/1/cs/5/

Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

POSIX

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion
?!	Negative lookahead
?<=	Lookbehind assertion
?!= or ?<!	Negative lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed
\xxx	Octal character xxx
\xhh	Hex character hh

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

String Replacement

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string

Some regex implementations use \ instead of $.

Sorteringsalgoritmer og O-notation

Matematik A og Informatik C

Vejledere: Jens Christian Larsen og Kristian Kjeldgaard Hoppe

Balder Westergaard Holst

24. marts 2022

Indhold

1 Indledning 2

2 Resume 3

3 Store-O-Notation 4

4 De to Algoritmer 5

4.1 Insertionsort 5

4.1.1 Analyse af Insertionsort 5

4.2 Mergesort 5

4.3 Sammenligning af Algoritmerne 6

4.4 Den Optimerede Mergesort 6

5 Den Hurtigste Køretid 7

6 Konklusion 8

1 | Indledning

Dette er min indledning pt.

2 | Resume

Dette er hvad jeg har skrevet og fundet ud af.

3 | Store-O-Notation

$$O(f(n)) = \{g(n) : \exists c > 0 : \exists n_0\}$$

4 | De to Algoritmer

4.1 Insertionsort

Insertionsort implementeret i python:

```python
def insertionsort(l):
   for i in range(1,len(l)):
      element = l[i]

      if element < l[0]:
         for j in range(i,0,-1):
            l[j] = l[j-1]
         l[0] = element
      else:
         j = i
         while(l[j-1]>element):
            l[j] = l[j-1]
            j -= 1
         l[j] = element
   return(l)
```

4.1.1 Analyse af Insertionsort

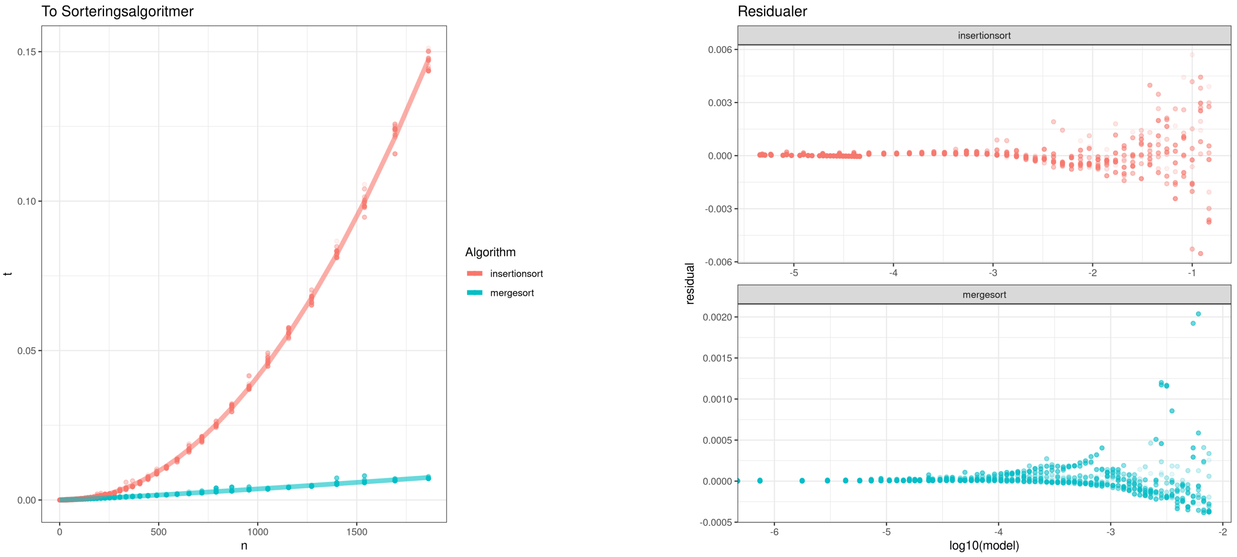
Funktionen for køretiden af denne algoritme er en del af mængden $O(n^2)$.

4.2 Mergesort

Mergesort implementeret i python [1, s. 106]:

```python
def mergesort(l):
   if len(l) <= 1:
      return(l)
   else:
      return(merge(mergesort(l[:len(l)//2]),mergesort(l[len(l)//2:])))

def merge(a,b):
   c = []
   while True:
      if (len(a) == 0):
         return(c + b)
      elif (len(b) == 0):
         return(c + a)
      elif (a[0] <= b[0]):
         c.append(a[0])
         a.pop(0)
      else:
         c.append(b[0])
         b.pop(0)
```



(a) $y = 0.941$ (b) $y = 3sinx$

Figur 4.1: Sammenligning af insertionsort og mergesort

4.3 Sammenligning af Algoritmerne

4.4 Den Optimerede Mergesort

5 | Den Hurtigste Køretid

Bevis for at den hurtigste køretid for en sorteringsalgoritme er $n \cdot log(n)$

6 | 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

```
def insertionsort(l):
    for i in range(1,len(l)):
        element = l[i]

        if element < l[0]:
            for j in range(i,0,-1):
                l[j] = l[j-1]
            l[0] = element
        else:
            j = i
            while(l[j-1]>element):
                l[j] = l[j-1]
                j -= 1
            l[j] = element
    return(l)
```

Mergesort Algoritmen

```
def mergesort(l):
    if len(l) <= 1:
        return(l)
    else:
        return(merge(mergesort(l[:len(l)//2]),mergesort(l[len(l)//2:])))

def merge(a,b):
    c = []
    while True:
        if (len(a) == 0):
            return(c + b)
        elif (len(b) == 0):
            return(c + a)
        elif (a[0] <= b[0]):
            c.append(a[0])
            a.pop(0)
        else:
            c.append(b[0])
            b.pop(0)
```

Kode til test af algoritmerne

```
import random
import time
import pandas as pd
import os

import sys
sys.path.insert(1, './algoritmer')
from mergesort import *
from insertionsort import *

# Denne funktion timer køretiden af en funktion med input l og returnerer funktionen køretid i milisekunder
def test(fun,l):
   start_time = time.perf_counter()

   fun(l)

   return(time.perf_counter() - start_time)

# Denne funktion returnerer en liste af tilfældige tal mellem 0 og 1000, med n elementer
def createRandomList(n):
   return([random.randint(0,1000) for i in range(n)])

# Laver en mappe i filsystemet hvis der ikke allerede er en med stien
def makeIfNeeded(dir_path):
   if(os.path.isdir(dir_path) == False):
      print(f"made dir: {dir_path}")
      os.mkdir(dir_path)
   return(dir_path)

# Finder det næste versionsnummer for til navngivning af fil på baggrund af indholdet i en folder
def newVersionNumber(dir_path,extention):
   file_names = os.listdir(dir_path)
   version = 0

   thisfilename = f"{version}{extention}"

   while(thisfilename in file_names):
      version += 1
      thisfilename = f"{version}{extention}"

   return(thisfilename)

# Dette er funktionen der tester en liste med funktioner og gemmer deres køretider
def fullTest(functions):

   # hvor mange datapunkter pr. n-værdi
   trials = 10

   data_dir = "../data/"
   version_number = newVersionNumber(data_dir,"")

   seed = time.time()
   print(f"Seed: {seed}")

   for function in functions:

      # i denne liste gemmes antallet af elementer at den liste som algoritmen sorterer for hvert datapunkt.
      ns = []
      # i denne liste gennes den tid det tager at sorterer listen med n elementer
      times = []

      # Bruger det samme seed til test at hver algoritme. på den made er det de samme pseudo-tilfældige liste som algoritmerne sorterer
      random.seed(seed)

      # Vi laver testen et antal (trials) gange pr. n-værdi
      for trial in range(0,trials):

        # En lykke der køre et abitrært antal gange (jo højere en i-værdi jo højere maks antal elementer i listen)
        for i in range(0,80):

                # Jeg bruger en potensfunktion til at fa flere datapunker tættere på y-aksen og færre lange oberationer (pga. lange liste)
            n = round(pow(1.1,i))

            print(f"function=\"{function.__name__}\": Trial: [{trial+1}/{trials}] {i=},{n=}") # lidt feedback

            # gennererer en tilfældig liste
            l = createRandomList(n)

            # gem størrelsen af listen der skal sorteres
            ns.append(n)
            # gen den tid det tager at sortere listen
            times.append(test(function,l))


        data = {
            "n": ns,
            "t": times
            }

        version_dir = makeIfNeeded(data_dir + version_number + "/")
        algorithm_dir = makeIfNeeded(version_dir + function.__name__ + "/")
        full_path = algorithm_dir + newVersionNumber(algorithm_dir,".csv")

        print(f"\ndata saved to \"{full_path}\"\n")

        pd.DataFrame(data).to_csv(full_path,index = False)


if __name__ == "__main__":
    functions = [mergesort,insertionsort]

    fullTest(functions)
```

Bilag 2 - Databehandling og Plots

Kode til databehandling og generering af plots

```
library(ggplot2)
#library(tikzDevice)

#working dir
setwd("/home/Balder/Documents/Skole/Gym/SRP/data/5")


#import data
dataset <- read.csv("mergesort/0.csv",header=TRUE,sep=",")

#dir = "1"
algorithm_dirs = list.files()

M = NULL
for (j in 1:length(algorithm_dirs)){
   algorithm_dir = algorithm_dirs[j]

   files = list.files(algorithm_dir)

   for (i in 1:length(files)){
      file_path = paste(algorithm_dir,files[i],sep="/")
      print(file_path)
      m = read.csv(file_path,header=TRUE,sep=",")
      m$Algorithm = algorithm_dirs[j]
      M = rbind(M, m)
   }

}

M$algorithm = factor(M$Algorithm)
summary(M)

# punktm?ngder for hver algoritme
m_merge = subset(M,M$algorithm=="mergesort")
m_insertion = subset(M,M$algorithm=="insertionsort")

# laver modeller
model_merge = nls(t~a*n*log2(n), data=m_merge, start=list(a=0.000001))
model_insertion = nls(t~a*n^2 + b*n + c, data=m_insertion, start=list(a=1,b=1,c=1))


# Sætter ny path til hvor outputtet skal være
setwd("/home/Balder/Documents/Skole/Gym/SRP/img")

# gemmer r2-værdierne i to filer
writeLines(toString(round(with(m_merge,cor(t,n)),digits=3)),"r2-merge.txt")
writeLines(toString(round(with(m_insertion,cor(t,n)),digits=3)),"r2-insertion.txt")
print("r2 saved to files")
```

```

# laver modelerede v?rdier for hver n
m_merge$model = predict(model_merge)
m_insertion$model = predict(model_insertion)

m_merge$residual = resid(model_merge)
m_insertion$residual = resid(model_insertion)

# kombinerer de to
M = rbind(m_merge,m_insertion)

summary(M)


ggplot(M, aes(x=n, y=t, colour=Algorithm)) +
   geom_point(size=1.5,alpha=0.1,shape=19) +
   geom_line(aes(x=n, y=model,color=Algorithm), size=2, alpha=0.6) +
   labs(title="To Sorteringsalgoritmer") +
   theme(legend.position = c(.9, .9)) + # virker ikke!!
   guides(colour = guide_legend(override.aes = list(alpha = 1))) + # lav legend alpha 1
   theme_bw()

   ggsave("toAlgoritmer.png")

   ggplot(M, aes(x=log10(model), y=residual, colour=Algorithm)) +
      geom_point(size=1.5,alpha=0.1,shape=19) +
      labs(title="Residualer") +
      facet_wrap(~algorithm,scales="free",ncol=1) +
      theme_bw() +
      theme(legend.position="none")
   ggsave("toAlgoritmerResidual.png")

   C = data.frame(
            Algorithm = unique(M$Algorithm),
            R2 = c(with(m_merge, cor(t,n)), with(m_insertion, cor(t,n)))
   )

   write.table(C, "r2.txt", quote=FALSE,sep="\t", row.names=FALSE)
```