IS105

Gruppe 3
NoName

ICA04

Gruppemedlemmer:
Ali Al Musawi
Tor Borgen
Ann Margrethe Ly Pedersen
Brage Fosso
Adrian Lorentzen
Arne Bastian Wiik
Morten Schibbye

Alle kode henvisninger ligger i README.md
https://github.com/GB-Noname/is105-ica04

Oppgave 1

a. Txt filene har forskjellig størrelse på grunn av at den ene ble skrevet i Windows, hvor i Windows så blir det brukt 2 kontrollkarakterer(Carriagereturn og Linefeed) i motsetning til Linux som har kun 1 fordi Linux trenger ikke å bruke Carrigereturn.
Tekstfiler består av linjer skilt av carriagereturn og linefeed karakterer.
Hver gang du trykker på retur (enter) på tastaturet, setter du inn et usynlig tegn som kalles line ending. Forskjellige operativsystemer/programmer håndterer linjeendringer på ulike måter.
Det er i hovedsak programmene som behandler linjeendringer, men på ulike operativsystemer fins det konvensjoner som de fleste utviklere følger slik at det blir en "unison" tolking av linjeendringer på disse operativsystemene.
Når du ser på endringer i en fil, håndterer Git linjestykker på sin egen måte. Git er riktig konfigurert til å håndtere linjestykker, den kan åpne input filer i binær modus, og dermed kan du også få ulike størrelser på to helt samme filer.

   Vi har lagt txt filene på en github pages server for å forenkle start av program. En absolutt lokal filepath er ugunstig å bruke, en relativ path vil fungere iom at alle har ulik mappestruktur til repo. Men for videre utvikling valgte vi å gå for webserver der filepath blir lik for alle.
   Det var også i begynnelsen problemer med å clone repo pga instillinger i git installeringen. Dette resulterte i at filene ble konvertert til windows format linjeslutt. Dermed ga webserver mening for å spare tid på av-reinstallering av git for alle.
   Dette hjalp oss også veldig mtp fremtidige ICA'er da behovet for å hente filer viste seg å være gunstig.

Oppgave 2

b.
ubuntu@newtest:~/golang/fileinfo$ ./fileinfo /dev/stdin
Information about a file /dev/stdin
File size in bytes: 0 File size in kibibytes: 0 File size in mibibytes: 0 File size in gibibytes: 0
Permissions: Dcrw--w----
Is a directory: false
Is a regular file false
Is not append only file
Is a device file
Is a UNIX character device
Is a UNIX block device
Is not a symbolic link

ubuntu@newtest:~/golang/fileinfo$ ./fileinfo /dev/ram0
Information about a file /dev/ram0
File size in bytes: 0 File size in kibibytes: 0 File size in mibibytes: 0 File size in gibibytes: 0
Permissions: Drw-rw----
Is a directory: false
Is a regular file false
Is not append only file
Is a device file
Is not a UNIX character device
Is a UNIX block device
Is not a symbolic link

Begge gir 0 I størrelse pga. det er systemfiler / block device.
Disse bruker til å kommunisere med kernel.
Samme årsak er at de gir false på både directory og regular file.
De er device filer/ system filer.
Stdin er character device mens ram0 er kun block device.
Med andre ord er ram0 en enhet av fixed size og brukes til å kontrollere RAM, dette gir lese/skrive funksjonalitet som er ulik fra vanlige character device filer.

c.

Vi har kjørt programmet på windows og linux basert system. Med identiske filer kompatible med hvert system. Disse gir tilnærmet identiske resultat utenom rettigheter (noe som er logisk mtp linux vs windows rettighetsystem) Konklusjonen er at krysskompilering fungerer bra for både windows og linux da kjernen til golang konverterer syscalls riktig til hver plattform.

Oppgave 3
a.

Vi har tre hovedmetoder for å operere filer i Golang. "io/ioutil" pakken lar oss implementere Input/Output funksjoner som read og write. Pakken "os" gir et

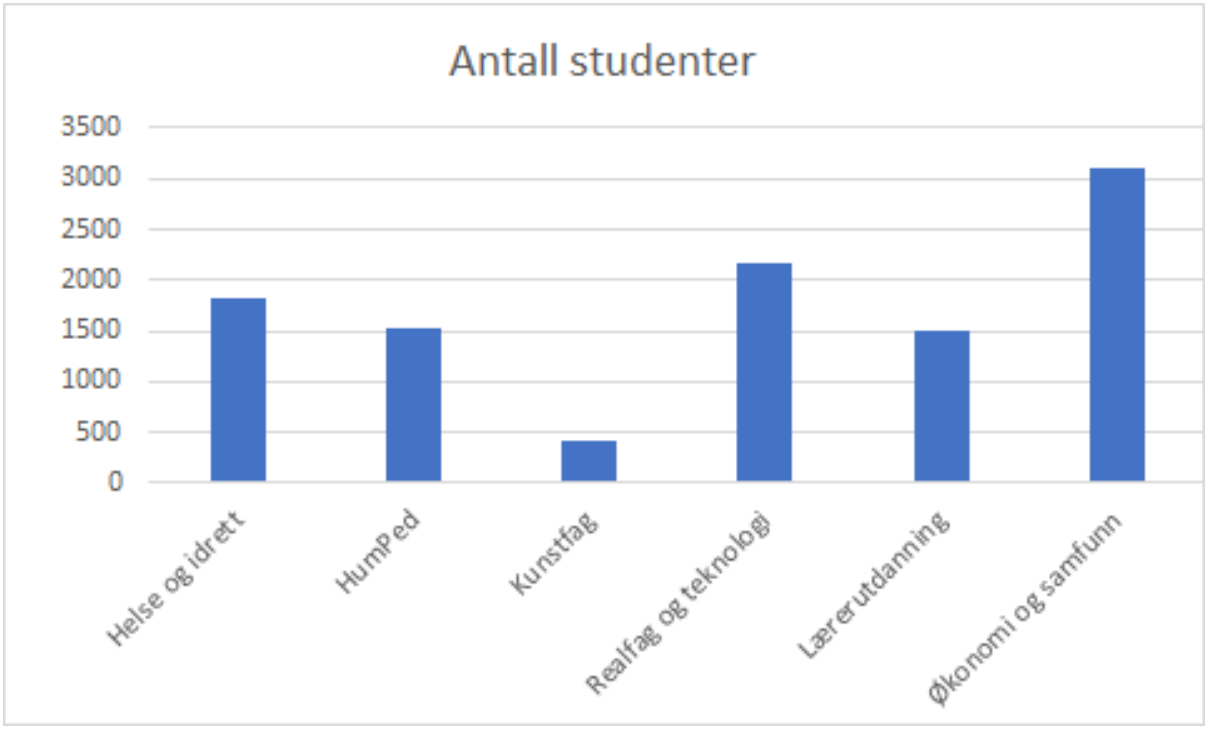
plattform-uavhengig grensesnitt til operativsystemets funksjonalitet. Vi har også "bufio" pakken. Bufio implementerer bufret I / O. Det pakker en io.Reader eller io.Writer objekt og skaper et annet objekt (Reader eller Writer) som også implementerer grensesnittet. "buffer", er i utgangspunktet et sted vi leser eller skriver våre bytes til før vi faktisk gjør et systemkall for å sette dem på et fysisk medium som f.eks harddisken.

c.

Vi laget benchmark tester for tre forskjellige metoder i hvordan vi kan åpne og lese filer. Benchmark testene gir oss en tidsramme på hvor lang tid koden blir utført på, altså hvor fort filene kan leses. Når det er sagt har ikke metodene så mye funksjonalitet, de er mer basert på effektivitet

Oppgave 4

a.



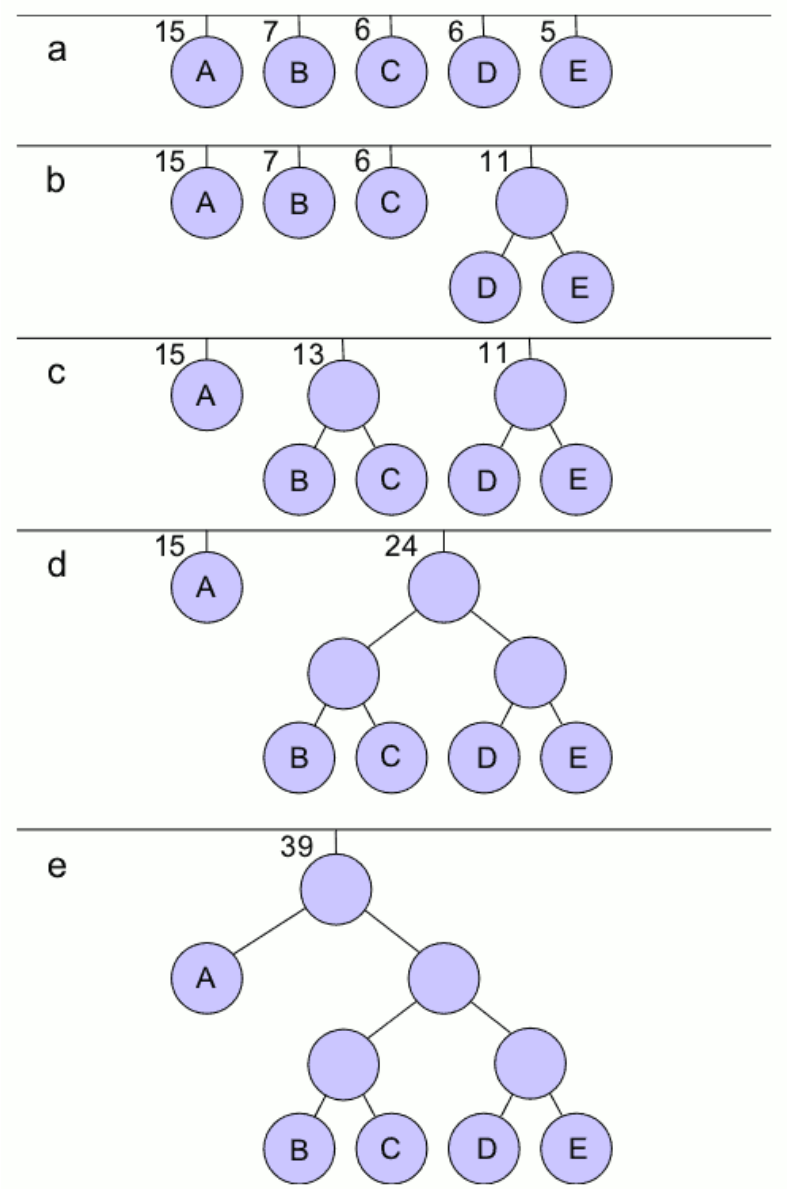
b. Kunstfakultetet får minst subtotal av bits og dermed minst informasjon.

c.

Fakultet	Antall studenter	Sannsynlighet for tilhørighet		Code	Log(1/p)	Subtotal of Bits			
Helse og idrett 17%	1829	17,35458772	17	"001"	1,73	5487			
HumPed 15%	1525	14,47006357	15	"000"	2,32	4575			
Kunstfag 4%	420	3,985197837	4	"111"	2,55	1260			
Realfag og teknologi 20	2166	20,55223456	20	"10"	2,73	4332			
Lærerutdanning 14%	1506	14,28978081	14	"110"	2,83	4518			
Økonomi og samfunn 3	3093	29,3481355	30	"01"	4,64	6186			
Sum	10539	100	100			26358	Bits need	2,5009963	"3"

d. For 100 gjennomsnittlige studenter blir det totalt 60+40+51+45+42+12 bits som må overføres altså % sannsynlighet*bits i kode som er lik 250 bit i gjennomsnitt.

e. Vi starter med å lage et tre der alle nodene har to mulige grener til nye noder. Vi fordeler nodenes plassering i treet basert på dette køsystemet:
   1. Opprett en node som representerer en sannsynlighet og legg den i prioritetskøen
   2. Om det er mer enn en node i treet: Fjern de to nodene med lavest sannsynlighet fra køen, lag en foreldrenode for disse to med sannsynligheten for begge to i foreldrenoden. Legg så foreldrenoden inn i køen.
   3. Gjenta til treet er blitt fylt og det ikke er noen par igjen i prioritetskøen. Alenenoder kobles opp mot toppnoden.



Illustrasjon hentet fra Wikipedias artikkel om Huffmann-Koding
https://en.wikipedia.org/wiki/Huffman_coding#/media/File:HuffmanCodeAlg.png