UiT Norges arktiske universitet

Hjemmeeksamen i:	INF-1101 Datastrukturer og Algoritmer
Innleveringsfrist (dato og tidspunkt):	11.05.2022 Senest kl 13:00
Kursansvarlig:	Morten Grønnesby
Antall sider:	Eksamen: 8 sider inkludert forside. Vedlegg A Sensorveiledning: 4 sider.
Support:	Du kan ringe 776 20 880 for support på eksamensdagen.
Vekting av spørsmål, eller annen informasjon:	Kode vektes 50% Rapport vektes 50%
Viktig informasjon om sitering og plagiering:	1. Dette er en individuell eksamen som skal besvares uten samarbeid med andre. 2. Alle hjelpemidler er tillatt (egne notater, pdfer fra forelesningene, lærebok, internett etc). 3. Alle eksamener som leveres i WISEflow blir automatisk sjekket for plagiat. Det er ikke tillatt å kopiere medstudenter, nettressurser, kilder, eller litteratur uten referanser.

Viktige Datoer

- Starttidspunkt: Onsdag 23.03.2022 kl 10:15
- Innleveringstidspunkt: Onsdag 11.05.2022 kl 13:00
- Frivillig *design review*: Uke 14 og 15

Vær nøye på å levere innen fristen, det er ikke mulig å levere eksamen etter tidsfristen.

Uke 14 og 15 vil vi sette av tid til *design review,* dette er et individuelt møte mellom kandidatene og kursstaben hvor kandidaten forklarer sin tenkte løsning på eksamen og får veiledning om noe skulle være uklart. Dette er et tiltak for å sikre at ingen har misforstått eller sitter for mye fast på hjemmeeksamen.

Det vil komme en oversikt over tid og sted for hver enkelt student på canvas.

Oversikt

I denne eksamensoppgaven så skal du implementere en *index abstrakt datatype (ADT).* Indeksen skal støtte indeksering av ord i tekstdokumenter og støtte søk etter spesifikke ord. Funksjonaliteten vil ligne veldig på en typisk *ctrl + f* funksjonalitet i f. eks VS Code. Vedlagt så ligger det en test-applikasjon, med et kommandolinjegrensesnitt som kan brukes til å søke etter ord og vise resultatene.

Index data strukturen

For å kunne søke etter ord i tekstdokumenter, så må du implementere en *index* data struktur. Indeksen skal ha følgende funksjonalitet:

- `index_t *index_create()`
  - Lager en ny, tom index struktur

- `void index_destroy(index_t *index)`
  - Frigjør den gitte indexen og alle ord assosiert med denne.

- `void index_add_document(index_t *idx, char *document_name, list_t *words)`
  - Legger til alle ord fra et dokument i indexen

- `search_result_t *index_find(index_t *idx, char *query)`
  - Gitt et søkeord, returnerer et søkeresultat som korresponderer til søkeordet i form av en `search_result_t struct.`

- `char *autocomplete(index_t *idx, char *input, size_t size)`
    - Gitt en tekst streng på minst 3 bokstaver, returnerer det nærmeste ordet som starter med strengen.

I tillegg så må indexen støtte følgende funksjonalitet for **søkeresultat**:

- `char **result_get_content(search_result_t *res)`
    - Returnerer innholdet i det nåværende dokumentet hvor søkeordet ble funnet. Innholdet returneres i form av en dobbeltpeker til tekststrenger, der hver streng er et ord i dokumentet (inkludert mellomrom og tegn). Flere kall returnerer det neste dokumentet og NULL om det ikke er flere dokumenter igjen.

- `int result_get_content_length(search_result_t *res)`
    - Returnerer lengden på det nåværende dokumentet. Dvs lengden på arrayet av content.

- `search_hit_t *result_next(search_result_t *res)`
    - Henter neste resultat i form av en index i innholdet til dokumentet. Funksjonen skal returnere en `search_hit_t` struct, som inneholder indeks og lengde på søketreffet. Dvs hvis ordet «flint» er det 28. og 101. symbolet i innholdet til dokumentet, så skal next først returnere (28, 5), deretter (101, 5) ved neste kall. Om du implementerer søk på hele setninger så må lengden være lengden på hele setningen, inkludert mellomrom.

Indeks strukturen må holde orden på alle dokumentene og alle ordene som forekommer i dokumentene. Her kan det være lurt å lage en egen `document_t` struct for å holde orden på et dokument som brukes internt i index.c.

Søkesyntax og Autocomplete

Index ADT-en skal støtte søk etter ord separert av mellomrom. Til å starte med så anbefales det å implementere søk etter kun ett enkelt ord, før man implementerer søk etter hele setninger. Søkestrengen tillater kun alfabetiske ASCII-tegn, og mellomrom. Du må selv splitte input inn i enkeltord om det forekommer flere ord separert av mellomrom. F eks:

`“Captain Flint said” -> [“Captain”, “Flint”, “said”]`

Om du implementerer søk på hele setninger så må du ta høyde for at mellomrom og spesielle tegn blir behandlet som egne ord når filene leses inn i indeksen. Dvs at følgende setning blir *tokenized* slik:

`“Captain Flint said” -> [“Captain”, “ ”, “Flint”, “ ”, “said”]`

Det vil følge med kode for en simpel User Interface (UI) som etter 3 bokstaver med input, kaller på en *autocomplete* funksjon med det som er input så langt, og indexen. Denne funksjonen skal returnere det nærmeste ordet assosiert med disse 3 bokstavene. Hvilket ord som er nærmest, må du bestemme selv og modifisere *trie.c* implementasjonen. Dette bør også beskrives i rapportens designseksjon. For eksempel så kan det være det nærmeste alfabetiske ordet eller det korteste ordet som begynner på inputen.
Husk: Tekststrengen som returneres må være null-terminert:

`“capt” -> “captain\0” <- Nærmeste ord, null-terminert.`

Søkeresultat og rangering

Søk i UIen blir sendt til `index_find()` som en tekststreng. Denne tekststrengen blir ikke delt opp på noen måte, og det er opp til deg å splitte strengen. Søkefunksjonen skal returnere en struct av typen `search_result_t` som videre brukes til å vise søkeresultatene og som holder konteksten for resultat funksjonene. UIen bruker `result_next()` for å hente neste resultat, internt i et dokument så bør resultatene være sortert på hvilket ord som kommer først i dokumentet. Du må bestemme deg for hvordan dokumentene skal sorteres. Her er det mulig å sortere alfabetisk eller ved flest treff.

Kode

Som utgangspunkt får du utlevert følgende kode:

- data/
  - **hamlet.txt –** Hamlet av Shakespear i ASCII tekst.
  - **treasure_island.**txt – Treasure Island av Robert Louis Stevenson I ASCII tekst.

- include/
  - **common.h –** Hjelpefunksjoner for å bla tokenize filer og sammenligne strings.
  - **index.h** - Spesifiserer interfacet til index ADTen.
  - **list.h** – Spesifiserer interfacet til en lenket liste.
  - **map.h** – Spesifiserer interface til et hash map.
  - **printing.h** – Hjelpefunksjoner for å printe log beskjeder.
  - **trie.h** – Spesifiserer interfacet til et trie.
  - **ui.h** – Spesifiserer interfacet for UIen.
- src/
  - **common.c** – Implementerer hjelpefunksjoner.
  - **hashmap.c** – Implementerer et chained hashmap.

- **index.c** – Fil for index implementasjonen, **implementer denne**.
- **linkedlist.c** – Implementerer en dobbelt-lenket liste.
- **main.c** – Hovedloopen i programmet.
- **trie.c** – Implementerer et trie. **Modifiser denne ut i fra ønsket autocomplete funksjonalitet.**
- **ui.c** – Implementerer UIen til programmet. Basert på ncurses.

- **Makefile** – Make filen som bygger programmet, targets er *index* og *test*
- **README.md** – Readme fil for oppgaven. Gir detaljer om kompilering. **Oppdater denne om det forandres noe på hvordan programmet bygges og kjøres.**

Husk at denne koden er kun et utgangspunkt og det er mulig at man ikke trenger alle datastrukturene som er lagt ved, eller at man bør legge til sine egne filer og kode. Alt dette avhenger hvordan du velger å designe index ADTen. Det finnes ikke ett riktig svar, men det er flere måter å løse oppgaven på.

Koden er tilgjengelig på canvasrommet til kurset under Oppgaver/Eksamen.

Rapport

Du kan anta at leseren (sensor) har en grunnleggende forståelse for de basis ADTene som har vært forelest om på kurset (hashmaps, lister, trær, array etc.), og du trenger derfor ikke å ha en veldig utpreget teknisk bakgrunn i rapporten.

Rapporten skal inneholde en beskrivelse av hvordan du har designet indeksen din, og gi en konkret beskrivelse av hvordan du har løst følgende problemstillinger:

- □ Hvordan en indeks bygges og holder orden på dokumenter og ord internt.
- □ Hvordan søk blir håndtert. (hele setninger eller enkeltord blandet avhengig av implementasjonen din)
- □ Hvordan autocomplete er implementert.
- □ Hvordan søkeresultat håndteres og itereres.

I tillegg til disse problemstillingene så skal du gjøre en **ytelsesanalyse** av søkemotoren (se neste seksjon). Rapporten har en begrensning på ***8 sider***. Rapporten kan være på *bokmål, nynorsk* eller *engelsk.*

I rapporten, så må det tydelig fremkomme hvor man har hentet kunnskap fra i form av sitering av kilder slik. [1] Hvis man kopierer noe direkte fra en kilde (direkte sitat), så må dette også tydelig fremkomme av sitatet slik:

Dette er et direkte sitat fra en kilde. [2]

Hvis man ikke siterer kilder i rapporten så blir dette tolket som plagiering, som er en form for fusk på eksamen. Fusk på eksamen kan føre til store akademiske konsekvenser, nærmere forklart her: https://uit.no/eksamen#modal_671895 (dette gjelder også for koden din).

Ytelsesanalyse

Som en del av rapporten så skal du gjøre en ytelsesanalyse. Denne kan være en del av diskusjonsdelen i rapporten (som er rapportens viktigste). Her skal du redegjøre for hvorfor akkurat din løsning er en effektiv løsning på oppgaven. Her bør du nevne noe om *big-O* notasjon for de datastrukturene du har valgt, og hvordan de vil skalere med hundrevis av dokumenter og hundretusenvis av ord i indeksen. Her kan man også argumentere for at løsningen man har gått for ikke nødvendigvis er den mest effektive og diskutere eventuelle forbedringer på sin egen indeks. Ting som er interessante å diskutere er:

- Tiden det tar å bygge en indeks.
- Søketiden avhengig av antall ord i søkestrengen.
- Søketiden avhengig av antall dokumenter i indeksen.

En annen mulighet er å måle tiden direkte. Generelt kan man gjøre dette på to måter: Måle direkte i koden med `gettime()` eller bruke en *profiler.*

Ca midtveis i eksamen så vil vi publisere en *benchmark* kode som dere kan bruke til å stress-teste implementasjonen deres, men fokuser først på en fungerende implementasjon før dere optimaliserer.

Veiledningssamtale

Underveis i eksamen, så vil vi tilby en *frivillig* veiledningssamtale, i form av et *design review*. Det vil bli kunngjort i kursets Canvas-rom når dette vil skje, og hver student vil få sitt eget tidspunkt (som vil være enten i tidspunktet for forelesning eller kollokvie). Samtalen gir muligheten til å forklare din overordnede forståelse og tenkte løsning på oppgaven, og få tilbakemelding om man er på riktig veg eller ikke, evt få veiledning på hva man burde fokusere på og hvordan gå

frem for å løse oppgaven. Her bør man stille litt forberedt og kunne forklare på et konseptuelt nivå hvordan man tenker å løse oppgaven for at vi skal kunne hjelpe best mulig.

Deloppgaver og arbeidsplan

Det er opp til hver enkelt å komme til en egen forståelse av oppgaven og gjøre egne valg av datastrukturer og algoritmer for å løse eksamensoppgaven. Under så har vi listet ut de tingene vi anser som en grei start på eksamen og som bør prioriteres først:

- □ Implementere en index som mapper ord til plassering i et dokument.
- □ Implementere søk i indeksen som finner et enkelt søkeord i alle dokumentene. (f eks «`captain`»).
- □ Implementere søkeresultat håndtering (get content, get length og next).
- □ Implementere autocomplete som returnerer et relevant ord.
- □ Forklare designet av indeksen i rapporten (med oversiktlige figurer).
- □ Diskutere eventuelle forbedringer i diskusjonsdelen i rapporten.
- □ Diskutere ytelsen av indexen og gjøre enten en *big-O analyse* eller måle tiden.

Når man har disse tingene på plass så kan man optimalisere med følgende ting:

- □ Effektivisere datastrukturene i indeksen (optimalisere med tanke på space-time).
- □ Implementere søk på hele setninger. (f eks «captain flint of the black pearl»)
- □ Prioritere visse ord i autocomplete (nærmeste alfabetiske ord, korteste ord først etc).
- □ Implementere en space-optimized autocomplete (Radix-tries eller Patricia-tries)
- □ Forklare det endelige designet av indexen i rapporten. (med figurer)
- □ Diskutere optimaliseringene som er gjort sammenlignet med den første implementasjonen.
- □ Plotte ut tidsmålinger før og etter optimalisering av indeksen. Samt kjøre benchmark kode på implementasjonen og stress-teste indeksen.

Innlevering

Husk at dette er en eksamen. Du må selv opparbeide deg en forståelse av oppgaven og jobbe selvstendig for å finne løsningen på oppgaven. Hvis du har generelle spørsmål om design og mulige løsninger så kan du diskutere med hjelpelærerne. De vil også kunne assistere deg hvis det er veldig spesifikke implementasjonsdetaljer du lurer på. Spør heller en gang for mye enn en gang for lite. ***Men husk å aldri kopiere kode eller rapport fra medstudenter.***

Eksamen skal leveres i 2 deler: Rapporten leveres som hoveddokument, i .pdf, koden leveres som et vedlegg i .zip.

Det er viktig at man ikke skriver hverken navn eller kandidatnummer i kode eller rapport, da dette blir tatt hånd om i Wiseflow.

Og det viktigste av alt: <u>Start så tidlig som mulig, også med rapporten. Det vil ikke bli gitt utsettelse på eksamen.</u>

[1] M. Grønnesby, "Eksempel på sitat," *INF-1101 Sitat Eksempler,* p. 1, 2021.

[2] M. Grønnesby, "Direkte Sitat," *INF-1101 Direkte sitering,* p. 1, 2021.

Vedlegg A: Sensorveiledning

Sensorveiledningen danner grunnlaget for hvordan sensorene vurderer eksamensoppgaven for kandidatene. Sensorveiledningen er inndelt i kategorier hvor hver kategori får en poengsum fra 0 til 10, deretter blir denne omregnet til poengtall basert på hvor mye denne kategorien er vektet. Maks poeng på eksamen er 100.
Sensorene vil få tilgang til et regneark hvor vekting regnes automatisk.
Eksempel på vekting:

Kategori – 30% -> Sensor vurderer kategorien til å oppnå 5/10, da regnes innvirkningen på total poengsum til å være 0.5*30 = 15 poeng til total poengsum.

Sensorveiledningen er lagt ved i eksamensoppgaven for å gi kandidatene innsikt i hva sensorene vil vurdere ut i fra og hvilken vekting som gjøres av de forskjellige kategoriene.

Rapport – 50%

Struktur – 10%

Struktur er en helhetsvurdering av rapportens format og språk. Det vurderes om kandidaten har brukt lesbare og ryddige tabeller for å presentere tidsmålinger som er gjort og at det er forståelige figurer som illustrerer designet for indeksen og evt andre deler av implementasjonen.

Rapporten bør også være delt inn i klare seksjoner, f.eks.: Introduksjon, Bakgrunn (evt Teknisk Bakgrunn), Design (evt Implementasjon), Diskusjon (og Resultat), Konklusjon. Disse seksjonene bør også avgrense hvor mye man snakker om, f eks så skal ikke introduksjon ha masse teknisk bakgrunn, men kort presentere oppgaven og den tenkte løsningen. Ikke alle disse seksjonene er nødvendige, om tekniske detaljer kommer tydelig frem der de er relevante, så er Teknisk Bakgrunn overflødig.

Kandidatene skal ikke vurderes på grammatikk og språk direkte, men om det er vanskelig å forstå hva kandidaten prøver å formidle så vil dette trekke noe ned.

Introduksjon – 3%

Kort og konsist legger frem oppgaven, samt den tenkte løsningen og gjerne en setning med hva som ble resultatet. Introduksjonen skal ikke inneholde mange tekniske detaljer. Om kandidaten har vurdert det nødvendig å gi teknisk bakgrunn, så bør denne komme i en egen seksjon.

Design/Implementasjon – 10%

Forklarer designet for løsningen på en forståelig måte. Her bør språket være enkelt og godt inndelt i seksjoner slik at det vil være mulig å få et helhetlig bilde av løsningen fra å lese teksten og se på figurene som er vedlagt.

Figurer er en stor fordel her, men de bør være lesbare og ryddige. Figurer som forvirrer mer enn de forklarer, gir ikke uttelling.

Designet bør holde seg til et konseptuelt nivå, og ikke gå inn i mange implementasjonsdetaljer. Hvis det er veldig spesifikke ting som er viktig for implementasjonen så bør disse komme i en egen implementasjonsseksjon.

Diskusjon – 20%

Diskusjonsdelen bør være den mest sentrale i rapporten. Her skal kandidaten presentere fordelene og ulempene med den valgte løsningen og hvordan disse relaterer til ytelsen i programmet. Her er det viktig at kandidaten har et bevisst forhold til space-time trade-off, altså hvis det er valgt en datastruktur som bruker mye minne, så er det til fordel tiden datastrukturen bruker.

Hvis kandidaten har valgt en mindre gunstig løsning, så bør dette diskuteres her og evt hvilke forbedringer som kunne vært gjort for å gjøre implementasjonen mer optimal. Med optimal her så må kandidaten selv gjøre rede for hva som er viktig (lav minnebruk eller lav tidsbruk), men i søkeapplikasjoner så er lav tidsbruk ofte prioritert. Hvis kandidaten har en veldig god utgreing om hvordan implementasjonen kan gjøres mer effektiv, så kan denne til dels veie opp for en litt mindre optimal implementasjon.

Hvis kandidaten har implementert en foreløpig løsning som deretter har vært optimalisert så kan dette med fordel diskuteres her. Det bør også redegjøres for hvorfor den endelige løsningen er mer optimal, enten ved å vise til en *big-O* analyse av de datastrukturene som er brukt, eller ved å måle tids og minnebruk.

Hvor enkel løsningen er å implementere spiller også en rolle her, en enklere løsning som bruker litt mer tid kan argumenteres å være bedre enn en løsning som er komplisert, men bruker litt mindre tid.

Hovedpoenget i diskusjonsdelen er at kandidaten er bevisst på valgene som er gjort i oppgaven, og at det presenteres resultater som støtter de valgene.

Konklusjon/Referanser – 2%

Konklusjonen bør nevne resultatene som ble funnet, eventuelle interessante funn. Og basert på disse resultatene konkludere hvor god implementasjonen er.

Referanser bør også være ryddige og til dels følge en siteringsstandard. Det bør også refereres i teksten hvor det er relevant.

Kode – 50%

Kompilerer – 10%

Koden kompilerer uten warnings og errors. Om det er noen warnings som ikke har noe betydning for funksjonen i programmet, så bør ikke dette trekke noe ned. Det trekker ned om koden ikke kompilerer pga flere errors og man ikke enkelt kan fikse dette for å få programmet til å kompilere.

Om det forandres noe på hvordan programmet kompilerer, så bør dette være oppdatert i README filen som ligger ved oppgaven.

Index – 10%

Funksjonaliteten som beskrevet i oppgaven er til stede og fungerer som forventet. Indeksen er implementert på en hensiktsmessig måte. Valg av datastrukturer spiller også inn her, om det er veldig mange kompliserte og uoversiktlige løsninger så trekker dette ned.

Effektivitet bør også tas med i vurderingen, hvis det er åpenbare valg som er ineffektive så trekker dette ned på totalen.

Det trekker noe ned om kandidaten ikke har implementert søk på flere ord eller hele setninger, men kun søk etter ett ord i gangen.

Autocomplete – 10%

Autoutfyllingen fungerer som forventet, og fyller ut ord som beskrevet i rapporten. Enten om dette er alfabetisk eller etter korteste match først. Hvis det ikke er noen spesiell form på dette så trekker dette noe ned, eller om kandidaten ikke har tenkt på dette.

Resultat – 10%

Implementasjonen fungerer som den skal uten noen videre bugs eller manglende funksjonalitet. Her gjøres det en helhetsvurdering ved å teste programmet, for å se om det oppfyller det som oppgaven forklarer.

Kommentarer/Lesbarhet – 10%

Koden bør ha en konsekvent kodestil og være lesbar. Det bør også kommenteres der det er hensiktsmessig. Det betyr ikke at hver linje kode skal kommenteres, men at der det er viktig å presisere ting, også kommenteres på en ryddig måte.

Hvis det viser seg at det er aspekter ved eksamen som veldig mange kandidater misforstår fordi oppgaven har vært for utydelig eller unnlat å nevne, så vil dette komme kandidatene til gode og delen av oppgaven kan strykes, evt at alle får uttelling for denne delen av oppgaven.

Karakteren settes ut ifra følgende karakterskala:

Karakter	Grense
A	89
B	77
C	65
D	53
E	41
F	0