UiT Norges arktiske universitet

Hjemmeeksamen i:	INF-1101 Datastrukturer og Algoritmer
Dato for innlevering:	10.05.2023
Tidspunkt for innlevering:	Senest Kl 14:00
Kursansvarlig:	Einar Holsbø
Kontaktinfo:	einar.j.holsbo@uit.no
Antall sider:	6 (inkl forside)
Vekting av spørsmål, eller annen informasjon:	Kode vektes omtrent 40 % Rapport vektes omtrent 60 %
Viktig informasjon om sitering og plagiering:	1. Dette er en individuell eksamen som skal besvares uten samarbeid med andre eller en KI (for eksempel ChatGPT). 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. Husk at det er heller ikke tillatt å kopiere fra egne tidligere innleverte besvarelser uten å referere.

Viktige Datoer

- Starttidspunkt: Onsdag 22.03.2023 kl. 09:00
- Innleveringstidspunkt: Onsdag 10.05.2023 kl. 14:00

Oversikt

I denne eksamensoppgaven skal du implementere en *Indeks* abstrakt datatype *(ADT).* Indeksen skal støtte indeksering av tekstdokumenter og søk etter spesifikke ord i en samling av dokumenter. Vedlagt er en eksempelapplikasjon (indexer.c) som bruker indeks ADTen for å bygge en enkel søkemotor med et web-grensesnitt som evaluerer søkeord på et sett med lokale filer og returnerer lenker til de relevante filene.

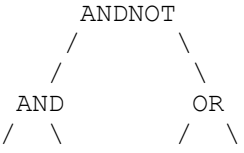
Søke-syntax

Søke-syntaxen til index ADTen må støtte spesifikke ord, kombinert med de boolske operatorene AND, OR og ANDNOT.
BNF grammatikken for søke-syntaxen er:

```
query   ::= andterm
        | andterm "ANDNOT" query
andterm ::= orterm
        | orterm "AND" andterm
orterm  ::= term
        | term "OR" orterm
term    ::= "(" query ")"
        | <word>
```

Du er nødt til å implementere en parser for denne grammatikken. Vi anbefaler å implementere en *rekursiv descendant parser (recursive descent parser).* Slike parsere implementeres ved å bruke en enkelt rekursiv funksjon for hver regel i grammatikken. Søkeuttrykket er som regel representert med et *abstrakt syntax tre,* hvor rot-noden representerer operatoren og hvert subtre representerer en operand (altså et søkeord eller en ny operator).

For eksempel, et abstrakt syntax tre for søket "(graph AND theory) ANDNOT (dot OR math)" vil se slik ut:



```
    /    \     /    \
graph theory dot  math
```

Søk blir evaluert ved å traversere syntax treet. Resultatet fra evalueringen av en løv-node (et ord) er det settet med dokumenter som inneholder ordet. Interne noder (boolske operatorer) blir evaluert med de korresponderende *sett* operasjonene. For eksempel, en OR node evaluerer til *unionen* mellom de to barne-nodene, og en AND node evaluerer til *snittet* av barne-nodene.

Indeks-strukturen

En indeks er en datastruktur som gjør søk effektivt. Gitt et søkespråk som definert over, så vil en passende indeks data struktur være en *invertert indeks.* En slik indeks assosierer hvert ord til et sett med dokumenter som inneholder ordet. Sett og hashmap implementasjonene fra tidligere oppgaver kan benyttes. Den inverterte indeksen til dokumentene vil da bli:

```
{<ord>: [<fil_1>, <fil_2>, ..., <fil_n>]}
```

Applikasjon

Inkludert i prekoden til oppgaven ligger det kode for et program som heter *indexer.* Dette programmet tar et argument, som er navnet på mappen med tekstfiler som skal indekseres. Programmet lager en ny instans av indeks ADTen og legger til alle filene i indeksen med `index_addpath()` funksjonen, og så starter den en web server på port 8080. Hvis du åpner en nettleser og går til adressen `http://localhost:8080/` så vil du få opp et web grensesnitt hvor du kan søke. Hvert søk blir håndtert av *indexer* ved bruk av `index_query()` funksjonen, som returnerer en liste med dokumenter (filnavn) som blir listet på resultatsiden. Alt du trenger å gjøre for å fullføre *indexer* programmet er å implementere *indeks ADTen.* Du trenger ikke å forandre noe i web grensesnittet, men du kan gjøre det om du ønsker.

I tillegg har du et hjelpeprogram som heter *assert_index,* som tester om indeks implementasjonen er korrekt.

Søkeresultat og rangering

Søk i webgrensesnittet blir sendt til *indexer* programmet som en tekststreng. Før `index_query()` kalles, så vil *indexer* dele opp søkestrengen og legge hver token i en lenket liste. For eksempel så vil søkestrengen "graph AND theory" gi listen:

```
"graph" -> "AND" -> "theory"
```

`index_query()` vil da få denne listen som et argument. Returverdien fra `index_query()` må være en lenket liste som inneholder dokumentene (filnavn) som

matcher søkestrengen. Hvert element i listen må være en data struktur av typen `query_result_t` (se *index.h*). Det forventes at nodene i listen er sortert på relevans til søket. Desto tidligere et dokument er i listen, desto høyere relevans til søket har det. Vi anbefaler å bruke tf-idf algoritmen for å bestemme sorteringen av listen.

Kode

Som utgangspunkt får du utlevert følgende kode:

- data/
  - cacm/ - Mappe med .html filer for å teste systemet ditt
- include/
  - **index.h** – Spesifiserer interfacet til index ADTen.
  - **map.h** – Spesifiserer interfacet til map ADTen.
  - **set.h** – Spesifiserer interfacet til sett ADTen.
  - **list.h** – Spesifiserer interfacet til list ADTen.
  - **common.h** – Definerer hjelpefunksjoner.
  - **httpd.h** – Spesifiserer interfacet til en web server. Brukes av indexer programmet for å starte en web server.
  - **printing.h** – Hjelpefunksjoner for å printe debug og error info.
- src/
  - **hashmap.c** – Implementerer en map ADT basert på en hashtable.
  - **aatreeset.c** – Implementerer en sett ADT basert på AA trær.
  - **linkedlist.c** – Implementerer en liste ADT basert på en dobbelt lenket liste.
  - **common.c** – Implementerer hjelpefunksjoner.
  - **indexer.c** – Indekser programmet. Bruker index implementasjonen for å bygge en index over ord.
  - **httpd.c** – Implementerer en web server.
  - **assert_index.c** – Utfører en rekke tester for å finne ut om alle ordene i et dokument er til stede i indexen.
- **template.html/style.css** – Web filer brukt av http serveren.
- **Makefile** – Make kommandoene for å bygge programmet.
- **parse_wiki_articles.py** – Parser ut et visst antall Wikipedia artikler fra en wikidump filsti. (OBS, krever at man laster ned wikipediaartiklene fra wikidump, se under)
- **README.md** – Readme fil for oppgaven. Gir detaljer om kompilering og hvordan få tak i mer data.

Som tidligere må du legge til en ny .c fil som implementerer ADTen din (f eks index.c), og legge til navnet til denne i Makefilen som verdien til INDEX_SRC-variabelen.

Vi gjør alt av kode tilgjengelig som en zip-fil. Du finner filen i kursets Canvas rom.

For å teste systemet så har vi lagt ved noen .html filer i data/cacm/. Dette datasettet er relativt lite, og det er anbefalt å teste på et større datasett. Et greit sted å starte er den norske wikipedia fra November 2006, som kan lastes ned her: https://dumps.wikimedia.org/other/static_html_dumps/November_2006/no/wikipedia-no-html.7z

Se README.md (sees best med en Markdown editor) for mer detaljer.

Rapport

Du kan anta at leseren (sensor) har en grunnleggende forståelse for de ADTene du bruker i oppgaven, og du trenger derfor ikke å ha en veldig utpreget teknisk bakgrunn. Rapporten skal inneholde en beskrivelse av implementasjonen din, og gi en konkret beskrivelse av hvordan du har løst følgende problemstillinger:

- Hvordan en indeks bygges. (hashmap -> dokumentliste).
- Hvordan en søkestreng bygges opp (eks rekursiv descent, abstrakt syntax tre).
- Hvordan rekkefølgen på resultatene beregnes (eks. tf-idf).

I tillegg til disse problemstillingene så skal du gjøre en **ytelsesanalyse** av søkemotoren (se neste seksjon). Rapporten har en begrensning på ***3500 ord***. Rapporten kan være på *norsk, nynorsk* eller *engelsk.*

Rapporten skal ha følgende struktur:

- **Introduksjon**
  - Fortell kort hva som er i denne rapporten og hvorfor
- **Teknisk bakgrunn**
  - *Kort* om ADT-ene du bruker
- **Design og implementering**
  - Hvordan henger systemet ditt sammen helt overordnet?
  - Hva er viktige detaljer ved implementeringen av systemet?
- **Eksperimenter**
  - Hva har du tenkt å måle og hvordan har du tenkt å måle det?
- **Resultater**
  - Hva sier tallene du målte? Lag gjerne (helst) figurer
- **Diskusjon**
  - Hvordan tolker du resultatene?
- **Konklusjon**
  - Fortell kort hva du gjorde og hva du så

Det bør 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 blir dette tolket som plagiering, som er en form for fusk på eksamen. Fusk på eksamen har store akademiske konsekvenser nærmere forklart her: https://uit.no/eksamen#modal_671895 (dette gjelder også for koden din).

Ytelsesanalyse og profilering

Her skal du inkludere en grundig ytelsesanalyse av søkemotoren din i rapporten. Analysen hører hjemme i eksperimenter-, diskusjons- resultatdelen i rapporten. Her skal du gjøre rede for de valgene som er tatt i oppgaven og hvordan implementasjonen av indeks påvirker ytelsen til søkemotoren. Ting som kan være interessant å måle er:

- Tiden det tar å bygge en indeks.
- Søketiden, avhengig av lengden på søkestrengen.
- Søketiden, avhengig av størrelsen på indeksen.

For å måle tid i programmet så kan man generelt gjøre dette på to måter. Den ene er å bruke tidtaking direkte i koden. I common.h så er det definert en `gettime()` funksjon som kan brukes til dette formålet. Den andre måten er å bruke en *profiler* som vil gi en mer nøyaktig oversikt over tidsbruk per funksjon. Nærmere detaljer er gitt i READMEen.

Det er viktig å merke seg en ting: Når man forbedrer et aspekt av en implementasjon, så må man som regel ofre noe annet (time-space trade-off), som å bygge indeks hurtig sammenlignet med hurtig søk.

Prøv å kjør flere eksperimenter og varier forskjellige parametre som: Antall dokumenter, set-implementasjonen, lengden på søkestrengen. Bruk figurer og grafer for å illustrere hvordan implementasjonen din skalerer med forskjellige N. Prøv også å si noe om hvorfor ting skalerer som de gjør.

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.*

Eksamen skal leveres i 2 deler: Rapporten leveres som hoveddokument, i .pdf. Koden leveres som et vedlegg i .zip. Ikke legg ved wikipedia-dump fra 2006 sammen med koden.

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: Start så tidlig som mulig. Det vil ikke bli gitt utsettelse på eksamen. Vi regner med at denne oppgaven tar 40-60 timer. Planlegg deretter.

[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.