Obligatorisk oppgave 1 — INF1020 h2005

Frist: fredag 7. oktober

Oppgaven skal løses individuelt, og må være godkjent for å kunne gå opp til eksamen.

Før innlevering må retningslinjene "Krav til innleverte oppgaver ved Institutt for informatikk" være lest, se
`http://www.ifi.uio.no/studinf/skjemaer/erklaring.pdf`

Ordliste

I denne oppgaven skal vi tenke oss at vi vil holde et register over alle norske ord (med alle bøyninger), og at vi skal lage operasjoner for å slå opp i dette for å se om man har stavet riktig etc. Ordene som skal stå i registeret blir oppgitt på en fil, og første del av oppgaven vil være å lese inn dette, og legge opp registeret. En rimelig komplett liste av alle norske ord med alle bøyninger vil fort bli på nærmere en million ord. For at vårt register skal få en overkommelig størrelse (ikke mer enn 15 000 ord) skal vi derfor bare ta med et utsnitt av registeret, men dette vil være komplett fra et gitt ord til et annet i den sorterte rekkefølge. De aktuelle ordene ligger i filen `ordbok1.txt` på kurssiden. Programmet skal imidlertid skrives som om "alle" ord er med i registeret.

Et oppslag i registeret får man gjort ved å oppgi et ord, og man skal da i første omgang få vite om ordet står i registeret eller ikke. Dersom man får negativt svar skal man i stedet få en liste over "liknende" ord i registeret. En nøyere definisjon av hva som skal regnes som "liknende ord" er gitt under.

MERK: I denne oppgaven blir man påtvunget en bestemt implementasjon (se under), og man kan, som alltid, diskutere om akkurat denne implementasjonen ville være den beste for nettopp det registeret, den datamengden og den dynamikken vi arbeider med her. Bakgrunnen for at vi har valgt denne implementasjonen er imidlertid dels at vi ønsket å anvende flere av de teknikker som er sentrale i pensum, og dels at denne implementasjons-måten, eller varianter av den, blir mer meningsfylte om man har et større register, der det kanskje også løpende kan komme tillegg til registeret. Med en million ord ville det f.eks. være rimelig

å holde dataene på sekundær-lager (gjerne da som et B-tre i stedet for som et binært søketre), og da ville hash-mekanismen som vi skal bruke ha en mye viktigere funksjon.

Noen detaljer

Med "liknende ord" som ord X skal vi mene:

- Ord som er lik X, bare at to bokstaver ved siden av hverandre er byttet om.
- Ord som er lik X, bare at én bokstav er skiftet ut med en annen.
- Ord som er lik X, bare at én bokstav er fjernet.
- Ord som er lik X, bare at det er lagt til én bokstav foran, bak eller innimellom.

Antall ord på fila overgår ikke femten tusen, og de kommer i helt tilfeldig rekkefølge i forhold til sorteringsrekkefølge etc. Ordene ligger fortløpende, med en eller flere blanke (eller linjeskift) mellom hvert ord. Linjelengden er ikke over 80 tegn.

Alle ordene er i små bokstaver, og vi antar også at ord blir gitt fra brukeren i små bokstaver. Det siste skal imidlertid sjekkes før oppslag starter.

Innlesing og lagring av ordene

Ordene skal lagres i et vanlig binært søketre, der nodene har en vanlig tekst-variabel. Vi skal anta at ordene kommer fra fila i så tilfeldig orden at de kan settes inn på enkleste vis uten at treet blir spesielt ubalansert.

Når vi skal lete etter "liknende ord" vil vi komme til å slå opp mange ord i registeret, og bare få av dem vil gi tilslag. For å effektivisere dette (ved hurtig å kunne få en pekepinn om et gitt ord står i registeret) skal vi også lage en hash-struktur, der vi for hver hash-indeks bare har angitt *om* det finnes ord i tabellen med denne hashindeksen (men ikke hvilket ord det er, og om det eventuelt er flere). For hver hashindeks trenger man derfor bare en boolsk verdi, som ideelt sett kunne implementeres som et 'bit' i maskinen (men se under).

Hashlengden skal her være omtrent ti ganger så stor som antall ord i registeret, og om vi slår opp på et ord som ikke er i registeret vil vi da i ni av ti tilfeller umiddelbart få vite at ordet ikke er å finne. Denne hash-tabellen ville i et virkelig tilfelle vært implementert slik at vi bare brukte ett 'bit' på hver indeks (slik at vi fikk 8 indekser i hver byte, og

at tabellen for f.eks. en million ord altså ville kreve omkring 1,2 Mbyte). Vi skal imidlertid implementere den som en boolean array av passelig lengde (som normalt trenger en byte til hver indeks). I vårt tilfelle skal denne altså være omtrent 150 000 lang, og en viktig del av oppgaven blir å finne en hash-funksjon som for vanlige ord (oftest på fra tre til ti bokstaver) sprer godt over hele området på 150 000 indekser. Den bør også være enkel, med tanke på raskt oppslag. Selv om første bokstav hos oss bare vil ha én eller noen få verdier, skal hash-funksjonen lages som om vi hadde en komplett ordliste.

Når man er ferdig med innlesningen skal man skrive ut litt statistikk på hvor "bra" lagringen av ordene ble. Følgende skal rapporteres:

For treet:

- Dybden av den dypeste noden i treet (vi følger læreboka og sier at roten har dybde null)
- Hvor mange noder det er på hver dybde fra null til største dybde
- Hva er den gjennomsnittlige dybde over alle nodene

Du kan anta at den største dybden ikke blir større enn 100, og kan enten gjøre denne registreringen mens du setter inn, eller i et eget gjennomløp etter hele innsettingen.

I tillegg skal det rapporteres hva som er det første (alfabetisk) og siste ordet i ordlisten.

For hashingen:

- Angi hvor mange hashindekser som er brukt, satt opp mot antall ord i registeret.
- Del hashindeksene inn i intervaller med n indekser i hvert. Tell antall brukte indekser i hvert intervall. Angi hvor mange intervaller som har b brukte indekser, for alle b fra 0 til n. Bruk n=50, som altså i gjennomsnitt skulle gi rundt fem brukte indekser per intervall.

(Hva som er fornuftige tall og fordelinger her trengs det statistikk for å ha kvalifiserte meninger om, men man vil vel ha en intuitiv idé om hva som er "helt vilt").

Bruk av registeret

Det å undersøke om ett gitt ord X er med i registeret skal vi kalle et "oppslag". Dette gjøres altså ved først å beregne hashindeksen til X, og

å slå opp på denne i hashtabellen. Om det her er angitt at intet ord har denne indeks kan man umiddelbart svare at X ikke finnes. I motsatt fall må man også søke etter X i søketreet. Om det finnes her svarer man positivt, i motsatt fall negativt.

Når brukeren ber om å få vurdert et ord X, skal det dermed foregå slik: Først gjør man et oppslag på X, og om svaret er positivt kommer man med et bekreftende svar, og er ferdig. I motsatt fall skal man systematisk generere alle "liknende ord" av X, gjøre oppslag på alle disse og skrive ut de som gir positivt svar.

I de tilfellene der det opprinnelige ordet ikke var i registeret (og man derfor gjør gjentatte oppslag på "liknende ord") skal man også skrive ut en liten statistikk til slutt over hvordan de enkelte oppslagene forløp. Man skal her angi tre tall:

- Antall oppslag som gav positivt svar
- Antall oppslag som gav negativt svar, men som passerte hash-testen
- Antall oppslag som gav negativt svar, og som ble stoppet i hash-testen

Om hash-funksjonen er godt laget skulle det siste tallet være omtrent ni (ti?) ganger så stort som det midterste.

Innlesning og utskrift

Det skal ikke legges stor vekt på vakker innlesing/utskrift etc., men det skal testes at det ordet brukeren vil ha vurdert består av bare små bokstaver.

En enkel variant kan være at systemet skriver ut "Ord til vurdering:" eller liknende hver gang den er ferdig med forrige ord. Brukeren angir så et ord, som avsluttes med "Return". Dersom "ordet" bare er tegnet '/' kan dette passelig bety avslutning.

Forskjellige feilsituasjoner får man behandle på en rimelig måte, uten at det helt store forlanges.

Noen forslag til detaljer i implementasjonen

Merk at også de norske bokstavene æ, ø og å skal være med, både når man godkjenner små bokstaver, og når man skal skifte ut hver bokstav i et ord med alle andre muligheter. Dette går greit i Java, merk at man kan

sammenligne char-variable direkte for likhet ('<', ...) hvis man trenger det. Vi viser også til det rikholdige settet av metoder i classen Character. F.eks vil følgende programbit skrive ut "Bingo".

```
    char c='å', b = 'b', a = 'a';

    if (Character.isLetter(c) & b < c &  b >= a) {
        System.out.println("Bingo");
    } else {
        System.out.println("Huff");
    }
```

For å kunne gå gjennom alle bokstaver på en behagelig måte kan det være greit å ha en "char[] bnr = new char[29]", som fast er fylt med bokstavene fra 'a' til 'å'.

String- og StringBuffer klassene i Java er prima til visse formål, mens det til andre ting kan bli litt fiklete. Når det gjelder det å generere alle "liknende ord" vil nok mange mene at det siste er tilfelle, og vi anbefaler derfor at man holder en 'char[] ord = new char[maxLengde];' og en heltalls-variabel 'ordlengde' som angir det ordet man holder på med (med 'maxLengde' f.eks. lik 50). Muligens trenger man et par slike arrayer for å kunne arbeide fritt.

Denne array-formen av et ord bruker man så til å holde det aktuelle ordet i alle andre tilfeller enn når det lagres i treet, altså når man leser inn ett og ett fra fila, når man skal beregne hash-indeks for et ord, og man må også lage en liten rutine som skriver ut et ord på en slik form. Det å få det over på tekst-form når man skal arbeide i treet kan man legge inn i en metode, f.eks. ved at alle tre-oppslag bruker følgende metode:

```
boolean treOppslag(char[] ord, int ordlengde, boolean settInn) {
    String T = new String(ord);

    // Let opp om det finnes node med verdi T i treet
    // (ikke rekursivt!)

    // Om T ikke fantes og 'settinn' er TRUE,
    // sett inn ny node med T.

    // Om T ikke fantes returneres FALSE ellers TRUE
}
```

Man kan selvfølgelig også lage to metoder av dette, én for innsetting, og én for rent oppslag.

Det å komponere en rimelig metode for beregning av hash-indeks er altså en del av oppgaven, men det er antakeligvis greit å basere den på

at verdien av en char-variabel direkte kan finnes ved å sette en 'int' lik denne. F.eks vil følgende programutsnitt skrive ut 'int-verdien av K er: 75':

```
char d='K';
int i = d;
System.out.println("int-verdien av K er: " + i);
```

Et siste råd angående uttestingen: Det å implementere alle varianter av "liknende ord" kan være litt fiklete. Implementer derfor bare én enkel variant av dette først, f.eks. bare den med å skifte ut én og én bokstav med en annen. Når man så har fått alt til å virke, kan man til slutt bygge ut med de andre variantene av "liknende ord" (og det er jo også mulig å legge inn flere varianter enn det oppgaven foreskriver).

Levering

Hva som skal leveres:

- Utskrift av kildekoden.
- Testutskrift fra ordbok1.txt som blant annet viser:
  - De forskjellige statistikkene
  - Sjekking av ordene:
    - etterfølger
    - eterfølger
    - etterfolger
    - etterfølgern
    - tterfølger
- O-notasjon analyse av metoden(e) som implementerer generering av "liknende ord".

  Utfordring: Alle genereringene kan gjøres i $O(n)$ tid (der n er antall bokstaver i det opprinnelige ordet) når ordet først er lest inn i en passende datastruktur!
- Sørg også for at programmet kan kjøres av gruppelæreren, og angi hva den kjørbare versjonen heter!