Westerdals

Oslo School of Arts,
Communication and Technology

PG4200 – Algoritmer og datastrukturer

Introduksjonslab

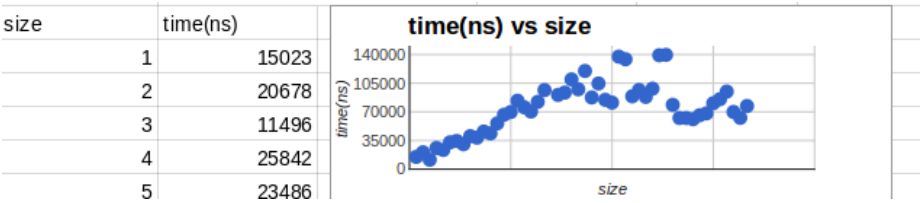
2. september 2016

1 Innledning

Du har samlet inn en liste med ord og ønsker å fjerne alle duplikater, slik at du kun sitter igjen med ett eksemplar av hvert ord. I denne laben skal vi skal se på ulike løsninger av dette problemet og sammenligne hvor lang tid hver løsning bruker på oppgaven.

I den grunnleggende metoden går vi gjennom listen fra første til siste ord. Hvert av disse ordene sammenlignes med de foregående ordene. Vi tar vare på ordet hvis det ikke er et duplikat. Se f.eks. metoden `BasicDedup.dedup()` i den vedlagte java-filen `Dedup.java`.

Målet med laben er å måle kjøretiden for deduplisering av lister av ulik lengde, med de ulike algoritmene. Når vi samler inn slike målinger på en systematisk måte kan vi hente dem inn i et regneark som vist i figur 1.



Figur 1: Skjermbilde av regneark med grafisk fremstilling av kjøretiden som funksjon av størrelsen på listen som dedupliseres. (Laget i google sheets)

1.1 Utdelt materiale

Alt det utdelte materialet finnes tilgjengelig her: Deduptest i google drive. Det er anbefalt å laste ned zip-filen, i og med at den inneholder nøyaktig det du trenger. Legg merke til at deler av koden avhenger av at du bruker java 8, og at koden er skrevet med tegnkoding utf8.

I de neste avsnittene forklarer jeg kort hva de ulike filene inneholder, men hvis du vil gå i gang med å herje med koden, kan du gå rett på `DedupTest.main` for å se hvordan de ulike komponentene brukes i praksis. Hvis du så står fast på noe, kan det hjelpe å lese hva som står i dette notatet.

1.1.1 `Dedup.java`

Denne java-filen definerer fem ulike dedupliseringsmetoder:

```
public interface Dedup {
    public static String[] hashSetDedup(String[] strs)
    public static String[] treeSetDedup(String[] strs)
    public static String[] arrayListDedup(String[] strs)
    public static String[] basicDedup(String[] strs)
    public static String[] sortDedup(String[] strs)
    /*...*/
}
```

Disse metodene gjør nesten nøyaktig det samme, og bruken er her illustrert ved `treeSetDedup`*

```
String[] words = new String[]{"så","gjør","vi","så"};
String[] uniqueWords = Dedup.treeSetDedup(words);

print(uniqueWords);
```

Når denne kodesnutten kjøres[1], får vi følgende utskrift:

```
gjør så vi
```

Koden over vil fungere om vi bytter ut `Dedup.treeSetDedup` med hvilken som helst av de andre metodene `Dedup.basicDedup`, `Dedup.arrayListDedup` et.c.

1.1.2 Dedup som grensesnitt (Avansert, men praktisk)

(Dette avsnittet kan du hoppe over dersom du ikke ønsker å utnytte at `Dedup` er et grensesnitt (`interface`).)

Kodefilen `Dedup.java` definerer grensesnittet `Dedup`, samt fem konkrete implementasjoner av dette grensesnittet, nemlig `BasicDedup`, `SortDedup`, `ArrayListDedup`, `TreeSetDedup` og `HashSetDedup`, med offentlige fabrikkmetoder i klassen `Dedup`:

[1] Obs: Forutsetter en meningsfull `print`-metode. En slik finnes f.eks. i `DedupTest.java`.

```
public static Dedup newHashSetDedup(){return new HashSetDedup();}
public static Dedup newTreeSetDedup(){return new TreeSetDedup();}
public static Dedup newArrayListDedup(){return new ArrayListDedup();}
public static Dedup newBasicDedup(){return new BasicDedup();}
public static Dedup newSortDedup(){return new SortDedup();}
```

Som grensesnitt kan `Dedup` kort oppsummeres slik:

```
public interface Dedup {
    /* Returnerer en duplikatfri liste av ord*/
    String[] dedup(String[] words);
}
```

Det betyr at `Dedup`-objekter implementerer en metode som fjerner duplikater fra lister av ord. I praksis bruker `Dedup`-objektene slik:

```
Dedup dedupObject = Dedup.newTreeSetDedup();
String[] words = new String[]{"så","gjør","vi","så"};
String[] uniqueWords = dedupObject.dedup(words);
print(uniqueWords);
```

Når denne kodesnutten kjøres, får vi følgende utskrift:

```
gjør så vi
```

Koden over vil fungere om vi bytter ut fabrikkmetoden `Dedup.newTreeSetDedup` med hvilken som helst av de andre fabrikkmetodene `Dedup.newBasicDedup`, `Dedup.newArrayListDedup` et.c..

1.1.3 `shakespeare.txt`

Denne filen inneholder alle ordene i Shakespeares samlede verker. Hensikten med å legge ved denne filen, er at vi skal testet metodene på data fra virkeligheten.

Det er faktisk slik at ulike datasett kan gi ulike konklusjoner. Derfor er det viktig å teste metodene på datasett som er relevante i forhold til det vi ønsker å bruke metodene til. Dersom du ønsker å utforske dette, kan du f.eks legge kommandoen `Utils.readFrom(tall100.txt")`, eller `Utils.readFrom(tall100000.txt")` øverst i `main`-metoden. Obs: dette forutsetter at `tall100.txt` og `tall100000.txt`er tilgjengelige i arbeidskatalogen din.

1.1.4 `Utils.java`

Denne kodefilen inneholder en rekke praktiske verktøy.

`Utils.Sampler`: Et objekt av denne klassen kan brukes til å trekke ut tilfeldige utvalg av ord fra tekstfiler. Slik kan man f.eks. trekke ut to uavhengige tilfeldige utvalg fra filen `shakespeare.txt`:

```
/* Forutsetter at shakespeare.txt er tilgjengelig */
Utils.Sampler sampler = new Utils.sampler("shakespeare.txt");
String[] sampleI = sampler.get(100);
String[] sampleII = sampler.get(1000);
```

`Utils.Stopwatch`: Brukes til å ta tiden på operasjoner, som vist her[2]:

```
String[] words = Utils.sample(100);
Utils.Stopwatch timer = new Utils.Stopwatch()
Dedup.hashSetDedup(words);
long timeUsageInNanoSeconds = timer.elapsedTime();
```

`Utils.Output`: En klasse som håndterer output som kan hentes inn i regneark. Her har vi et eksempel på hvordan den kan brukes til å registere kjøretiden etter deduplisering av en liste med `size` ord:

```
Utils.Output out = new Utils.Output("hashsetdedup.csv");

/*...  dedup m/tidtaking som vist over ...*/
out.addMeasurement(size,timeInNanoSeconds);
```

Denne klassen skreddersydd for akkurat denne anvendelsen[3].

1.1.5 `DedupTest.java`

Denne fila inneholder eksempler på bruk av de andre klassene, og skal være et godt utgangspunkt for å løse oppgaven. Derfor vil det være et godt tips å starte med å herje med koden i denne filen

[2] Det er også mulig å måle tiden slik:
`long timeInNs = Utils.Stopwatch.elapsedTime(Dedup::hashSetDedup,words);`
Legg merke til dobbelkolon-syntaksen som brukes i forbindelse med referanser til metoder.

[3] En grunn til å lage en slik klasse er at vi *abstraherer* måleregistreringen. Det betyr at vi kan skrive programmet vårt uten å tenke på hvordan utskriftene skal se ut. Dersom vi så ønsker å endre på hvordan utskriftene ser ut, kan vi gjøre det ved å endre klassen `Utils.Output`.

2 Arbeidsoppgaver

2.1 Måle tidsbruk

Skriv et java-program som måler kjøretiden for dedupliseringsmetodene `Dedup.hashSetDedup`, `Dedup.treeSetDedup` et.c.

Mål kjøretiden for store og små utvalg av ord, som f.eks. vist her:

```
Utils.Output out = new Utils.output("arrayListDedup.csv");
for(int i = 0; i < 100; i++){
    int size = i*10;
    String[] sample = Utils.sample(size);
    Utils.Stopwatch timer = new Utils.Stopwatch();
    Dedup.arrayListDedup(sample);
    out.addMeasurement(size,timer.elapsedTime());
}
```

2.2 Analysere målinger i regneark

La oss si at du har brukt et `Utils.Output`-objekt til å skrive data til filen `measurements.csv`, som antydet her:

```
Utils.Output out = new Utils.Output("measurements.csv");

for (....){
    ...
    out.addMeasurement(size,timeInNanoseconds);
    ...
}
```

Nå kan du åpne filen "measurements.csv" i et regnearkprogram[4], kan man lage plott av typen *xy-scatter* som vist i figur 1. Dersom regnearkprogrammet ditt ikke plasserer dataene i to søyler, kan det hjelpe å endre den øverste linjen i kodeeksempelet over til

```
/* Bruk komma som separator */
Utils.Output out = new Utils.Output("measurements.csv",",");
```

[4]Google sheets fungerer fint, men mange vil sikkert foretrekke å bruke MS Excel eller Libre Office Calc.

2.3 Spørsmål som skal besvares

Med utgangspunkt i de grafiske fremstillingene skal du svare på følgende spørsmål:

2.3.1 Grunnleggende spørsmål

- Hva vil du si er den raskeste og hva er den tregeste implementasjonen?
- Hva er svaret på spørsmålet over når antallet ord som skal dedupliseres ligger mellom 0 og 50?
- Hva er svaret på spørsmålet over når antallet ord som skal dedupliseres ligger mellom 50 og 500?
- Hva er svaret på spørsmålet over når antallet ord som skal dedupliseres ligger mellom 500 og 5000?
- Hva er svaret på spørsmålet over når antallet ord som skal dedupliseres ligger mellom 5000 og 50000?

Legg merke til at det ikke finnes noen fasitsvar på dette. Resultatene vil inneholde mye tilfeldig variasjon, i tillegg til at de avhenger av hvilken datamaskin du kjører testen på.

2.3.2 Mer dyptpløyende spørsmål

Dette er svar som det antageligvis er vanskelig å finne svar på:

- Sammenlign to datamaskiner: Ser dere de samme tendensene? Hva er likt, og hva er forskjellig? (Tips: (1) Se på formen på kurvene og det innbyrdes forholdet mellom dem. (2) Se på de absolutte tallene: Kan man si at kjøretiden på datamaskin A systematisk er X prosent av kjøretiden til datamaskin B?)
- Spiller det noen rolle hvilen rekkefølge vi gjør målingene i?
- Hvilken rolle spiller javas innebygde søppelhåndtering?