Jon Bergland

Deloppgave 1: Hashtabell med tekstnøkler

Implementasjon:

For å implementere hashtabell lagde jeg egne klasser for enkeltlenket lister og noder. Her ble boka brukt som inspirasjon, men med streng-verdier istedenfor double. Siden det er 124 navn i listen valgte jeg runde opp til nærmeste toerpotens, 128. Dette gjør at lastfaktoren blir litt høy, men man får testet om hashfunksjon gir god nok spredning.

For hashfunksjonen implementere jeg en basert på heltallsmultiplikasjon.

```
public static int multHash(int k, int x) {
  final int A = 1327217885;
  int mask = (1 << x) - 1;
  return ((k * A >>> (32 - x)) & mask);
}
```

Her ganges variabelen k med A for å sikre god spredning i tabellen. Produktet blir så høyre skiftet slik at mengden bits i produktet blir redusert og bare x bits gjenstår. I for hashingen av navnetabellen er ønskes bare de 7 siste bitsene. Til slutt blir produktet maskert ved hjelp av bitvis AND operasjon for å sikre at produktet havner i den ønskete rekkevidden (0 – 127).

For å konvertere en streng til et heltall bruke jeg denne funksjonen:

```
public int weighString(String string) {
  int sum = 0;

  for (int i = 0; i < string.length(); i++) {
    int value = Character.getNumericValue(string.charAt(i));
    sum += value * (i + 1);
  }
  return sum;
}
```

Her konvertere jeg hver char i strengen til sin Unicode ekvivalente verdi. Deretter ganget jeg verdien med karakterens nummer i strengen (+ 1 for å ikke å gange med 0).

```java
class HashtabellWithStrings {
  int pow = 7;
  int size = (int) Math.pow(a:2, pow);
  int antElements = 0;
  int collisions = 0;
  List<LenketListe> hashTabell = new ArrayList<>();

  HashtabellWithStrings() {
    for (int i = 0; i < size; i++) {
      hashTabell.add(new LenketListe());
    }
  }
```

```java
public void addValue(String string) {
  int hashValue = multHash(weighString(string), pow);
  LenketListe listAtIndex = hashTabell.get(hashValue);

  if (listAtIndex.getHead() != null) {
    System.out.println("Collision between: " + string + " and
    listAtIndex.insertAtFront(string);
    collisions++;
  } else {
    listAtIndex.insertAtFront(string);
  }
  antElements++;
}
```

For implementeringen av Hashtabellen (se kodeeksempel over) fylles først en Liste med tomme Lenkede lister. For hver gang det legges til et nytt element, sjekkes det om listen er tom eller ikke. Hvis listen er tom, blir bare det nye elementet lagt inn. Hvis listen ikke er tom, blir det først skrevet ut hvilke to elementer som hadde en kollisjon og så blir elementet lagt inn som vanlig.

Utskrift:

Kollisjoner:

```
Collision between: Håkon Duås Bjerkestrand and Nathalie Graidy Andreassen
Collision between: Vitaliy Konstantinovich Bravikov and Borgar Barland
Collision between: Scott Langum Du Plessis and August Reitan Bøgseth
Collision between: Edvard Bjørklund Eide and Ylva Johanne Björnerstedt
Collision between: Usman Ghafoorzai and Ingrid Midtmoen Døvre
Collision between: Vetle Solheim Hodne and Bård Helge Hansen
Collision between: Sigrid Hoel and Cathrine Evensen
Collision between: Sofia Simone Håbrekke and Usman Ghafoorzai
Collision between: Lida Victoria Johnsen and Charlotte Adele Heidenstrøm Corapi
Collision between: Jon Erik Klakken and Anders Nikolai Holsen
Collision between: Kristiane Skogvang Kolshus and Jonathan Jensen
Collision between: Daniel Moe Kvarsnes and Magnus Eik
Collision between: Jacob Lein and Sigrid Hoel
Collision between: Anders Lundemo and Håvard Johannes Versto Daleng
Collision between: Markus Stuevold Madsbakken and Lida Victoria Johnsen
Collision between: Aryan Malekian and Marius Strand Fredriksen
Collision between: André Wikheim Merkesdal and Aryan Malekian
Collision between: Matteus Rødøy Mitei and Kristiane Skogvang Kolshus
Collision between: Amund Mørk and Edvard Berdal Eek
Collision between: Jonas Reiher and Elise Christine Gjestad
Collision between: Vahideh Rezaei and Shiza Ahmad
Collision between: Therese Synnøve Rondeel and Vahideh Rezaei
Collision between: Embret Olav Rasmussen Roås and Vetle Pettersen
Collision between: Erik Sandvik and Vetle Traran Bjørnøy
Collision between: Kaamya Shinde and Maamoun Adnan Hmaidoush
Collision between: Garv Sood and Theo Holmvik
Collision between: Sigurd Spook and Markus Evald Dalbakk
```

```
Collision between: Eva Stamatovska and Jakob Huuse
Collision between: Henrik Sund and Garv Sood
Collision between: Håkon Sørli and Anne Cecilie Skår Nilsen
Collision between: Jeffrey Yaw Annor Tabiri and Markus Øyen Lund
Collision between: Sara Taghypour and Nicolai Forsberg Sommerfelt
Collision between: Nikolai Tandberg and Andreas Madsen
Collision between: Jeppe Evensen Thy and Anders Noel Lothe Morille
Collision between: Nicklas Persia Tufteland and Nikolai Tandberg
Collision between: Vegard Torkildsen Udnes and Kaamya Shinde
Collision between: Victor Udnæs and Aksel Estensen
Collision between: Simon Leirvik Zahl and Markus Hysing Jøssund
Collision between: Johannes Aamot-Skeidsvoll and Madeleine Negård
```

```
Hent mitt navn: Found: Jon Bergland
Antall elementer: 124.0
Lastfaktor: 0.96875
Antall kollisjoner: 39
Kollisjoner per person: 0.31451612903225806
```

Deloppgave 2: Hashtabeller med heltallsnøkler – og ytelse

1. Implementerer en hashtabell for heltall (se kode under). Her har jeg satt størrelsen til å være et primtall ca. 20% mer enn ti millioner. Siden nærmeste toerpotens var mer enn 35% større enn ti millioner, valgte jeg her å bruke et primtall i stedet.

```
class HashtabellWithInt {
  int size = 12_000_017;
  int antElements = 0;
  int collisions = 0;
  List<Integer> hashTabell = new ArrayList<>();

  public HashtabellWithInt() {
    for (int i = 0; i < size; i++) {
      hashTabell.add(e:null);
    }
  }
```

Under er implementasjonen min for dobbel hashing. For de to hashfunksjonene brukte jeg noen basert på restdivisjon, da de passer bedre til tabeller som er primtall.

Funksjonen addValue tar inn et tall og konverter det til et tall som er innenfor rekkevidden av tabellen. Hvis ingenting står på den indeksen, blir tallet satt inn der. Hvis det står et tall der allerede og det tallet ikke er det samme som den orginale verdien, blir den andre hashfunksjonen brukt til å finne det neste tallet i rekka. Dette gjentas til en ledig plass er funnet. Viktig her er at den andre hashfunksjonen bruker det originale tallet og ikke den konvertere verdien. Antallet elementer i tabellen blir også mindre enn ti millioner, siden det sjekkes for duplikater i innsetingen.

```
public void addValue(int value) {
  int hashValue = restHash1(value, size);
  Integer element = hashTabell.get(hashValue);
  if (element == null) {
    hashTabell.set(hashValue, value);
    antElements++;
    return;
  } else if (element != value) {
    int kollisjonsflytt = restHash2(value, size);
    do {
      hashValue = (hashValue + kollisjonsflytt) % size;
      element = hashTabell.get(hashValue);
      collisions++;
    } while (element != null && element != value);
    if (element == null) {
      hashTabell.set(hashValue, value);
      antElements++;
    }
  }
}
```

```
public int restHash1(int k, int m) {
  return k % m;
}
public int restHash2(int k, int m) {
  return (k % (m - 1)) + 1;
}
```

Under er hvordan jeg fylte en tabell med 10 millioner tilfeldige tall som var spredt utover et område på 1 milliard. Her vises også hvordan jeg tok tiden.

```
HashtabellWithInt hashTabell = new HashtabellWithInt();
List<Integer> randomNumbers = generateRandomNumbers(size:10_000_000, max_size:1_000_000_000);

long beforeTime = System.currentTimeMillis();
for (int i = 0; i < randomNumbers.size(); i++) {
  hashTabell.addValue(randomNumbers.get(i));
}

long afterTime = System.currentTimeMillis();
```

Koden under er hvordan jeg tok tiden på den innebygde hashmap funksjonen i Java.

```
HashMap<Integer, Integer> hashMap = new HashMap<>();
beforeTime = System.currentTimeMillis();
for (int i = 0; i < randomNumbers.size(); i++) {
  hashMap.put(randomNumbers.get(i), randomNumbers.get(i));
}
afterTime = System.currentTimeMillis();
```

```
Tidsforbruk på 10_000_000 tall: 3090ms
Antall elementer i hashtabell: 9950336
Lastefaktor: 0.8291934919758863
Kollisjoner: 11237533
Kollisjoner per element: 1.1293621642525438
Java.Hashmap
Tidsforbruk for java.Hashmap på 10_000_000 tall: 3120ms
Antall elementer i hashmap: 9950336
```

Over er utskriften jeg fikk fra programmet. Her kan man se at det i gjennomsnitt skjer ca. 1.13 kollisjoner per element. Dette kan nok bli forbedret ved å benytte bedre hashfunksjoner (kanskje kombinere restdivisjon og helttallsmultiplikasjon) eller øke størrelsen på tabellen.

Akkurat i denne kjøringen slo min hashtabell Java sin innebygde HashMap, men ikke med mye. I andre kjøringer slo Hashmap min. Med litt optimaliseringer tror jeg at den kunne slått Java sin mer konsekvent.