Karo F. Mahmoud
Jonatan Andre Vevang
Stian Wilhelmsen

28.09.22

Øving 5

Oppgave 1

Programmet leser fila med navn, og klarer å legge alle i hashtabellen. Disse blir lagt til ved hjelp av add-metoden nedenfor.

```java
public void add(int k, Node newNode) {
    int pos = divHash(k);
    if (table[pos] != null) {
        System.out.println("Collision. New name: "+newNode.value +", Old name: "+ table[pos].value +"\n");
        collisions++;

        newNode.next = table[pos];
        table[pos].next = null;
    }
    table[pos] = newNode; // Places node at open position in hashtable
}
```

Bilde 1: Add-metode som sørger for å legge til fil med navn i hashtabellen

Dersom det skjer noen kollisjoner når disse blir lagt til, blir disse håndtert ved hjelp av lenket lister. I tillegg blir totalt antall kollisjoner skrevet ut. Det blir også skrevet ut antall kollisjoner per person, og dette tallet er under 0,4. Programmet vil dessuten skrive ut lastfaktoren som representerer antall elementer hashtabellen inneholder, delt på størrelsen til hashtabellen. Til slutt klarer programmet å slå opp personer i ved hjelp av hashtabellen. Dette ble testet med et navn som ligger i hashtabellen, og gir derfor en true-verdi. Alle disse utskriftene kan observeres nedenfor.

```
Collision. New name: Morgan Torgersen, Old name: Arunan Gnanasekaran

Collision. New name: Mona Mahmoud Mousa, Old name: Daniel Skymoen

Collision. New name: Håkon Hansen Bergrem, Old name: Navid Muradi

Collision. New name: Ingrid Flatland, Old name: Beka Daniel Bonsa

Collision. New name: Steinar Nilsskog, Old name: Håkon Henriksen Juell

Collision. New name: Torstein Øvstedal, Old name: Jostein Lind Aanby




Total collisions 29.0
LoadFactor 0.76
Collisions per person 0.2543859649122807
true
```

Bilde 2: Utskrift av kollisjoner (ny og gammel verdi), antall kollisjoner totalt/per person, lastfaktor og en boolean-verdi utifra oppslag

Karo F. Mahmoud
Jonatan Andre Vevang
Stian Wilhelmsen

28.09.22

Oppgave 2

Programmet legger inn 10 millioner tilfeldige tall inn i en hashtabell og måler tiden (skriver ut tiden). Denne delen av programmet kan man legge merke til i metoden under. Metoden sørger også for å skrive ut lastfaktoren.

```
public static String runAlgorithmAndRecordTime(Hashtable hashTable,int amountOfNumbers) {
    Date start = new Date();
    int rounds = 0;
    double time;
    Date end;
    double lastFactor;
    do {
        fillHashTable(hashTable,amountOfNumbers);
        lastFactor = getLastFactor(hashTable,amountOfNumbers);
        end = new Date();
        ++rounds;
    } while (end.getTime()-start.getTime() < 1);
    time = (double) (end.getTime()-start.getTime()) / rounds;

    return "Time used with " + amountOfNumbers + " total numbers is: " + time + " ms" +
            "\nTotal number of collisions: " + hashTable.getCollisions() +
            "\nLastFactor is: " + lastFactor + "\n";
}
```

Bilde 3: Metode for å legge inn tilfeldige tall og måle tiden av dette

Kollisjoner blir dessuten håndtert ved hjelp av dobbel hashing, og totalt antall kollisjoner blir skrevet ut. I tillegg blir tidsmålingen prosessen ved bruk av hashtabell blir sammenlignet med tiden ved bruk av hashmap, ved å skrive ut tidsmålingen til tidsmåling ved bruk av begge. Alle utskriftene kan man legge merke til nedenfor. Man kan legge merke til at dersom man legger inn 10 millioner tilfeldige tall i en hashtabell istedenfor hashmap, blir dette utført over 4 ganger så raskt (sammenlignet med hashmap). Med tanke på at mange tall blir lagt inn i hashtabellen, vil det oppstå mange kollisjoner.

```
Time used with 10000000 total numbers is: 1020.0 ms
Total number of collisions: 6665809
LoadFactor is: 0.7692291716009513

Time used with java built in HashMap and 10000000 total numbers is: 4293.0 ms
```

Bilde 4: Utskrift av tidsmåling ved bruk av hashtabell I ms, totalt antall kollisjoner, lastfaktor og tidsmåling ved bruk av hashmap i ms