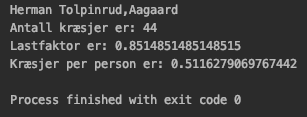
Utskrifter fra øving 4

1)

Optimalt burde vel ‘krasjer per person’ være rundt 0,75. I dette eksempelet er det 0,85 men oppgaven sa at tabellen ikke burde være mye større enn listen med navn. Derfor brukte jeg første primtall etter 100 (101). For å fikse dette er det bare å bruke et større primtall i attributtet ‘totalBuckets’.



2)

Oppgaven ba om å ikke ha en mye større array enn det antall elementer som skulle legges inn. Startet da først med første primtall over 10 millioner, men dette skapte latterlig mange krasj per tall (over 13). Etter dette så jeg at vi kunne bruke opp til 35% større array enn antall elementer så jeg gjorde det både for å få ned antall krasj og tidsbruk. Jeg lagde tilfeldige tall mellom 1 og 100 millioner for å ha stor variasjon av tilfeldige tall. Da fikk jeg utskrift som dette:

Et bilde som inneholder tegning

Automatisk generert beskrivelse

Testet tid mot Java ved å kjøre 5 tester og ta gjennomsnittet av tidsforbruket i hver test for å bestemme en gjennomsnitlig bruk av tid for å legge inn alle elementene i hashtabellen min og Java sin.

|  |  |  |
| --- | --- | --- |
|  | DoubleHash (min) | HashMap (Java) |
| Forsøk 1 | 9943ms | 9625ms |
| Forsøk 2 | 9731ms | 10778ms |
| Forsøk 3 | 7502ms | 9877ms |
| Forsøk 4 | 8829ms | 9580ms |
| Forsøk 5 | 12484ms | 10404ms |
| Gjennomsnitt | 9698ms | 10053ms |

Som tabellen viser var min metode bittelitt raskere enn Java sin, men min var også mer ustabil. Tok noen ekstra tester etter de fem som jeg logget og HashMap holdt seg stort sett mellom 9,5s og 10,5s, men gikk opp i 12,5s en gang. DoubleHash derimot ser man fra det jeg har logget at er mer ustabil, og når jeg testet videre var den så rask som 6,2s og så treig som 14,5s. Altså tiden mer enn doblet seg mellom de to testene.