# Rapport - øving 5

## Deloppgave 1

Programmet leser inn «navn.txt» linje for linje, og legger disse inn i en «hash table». Dersom det oppstår kollisjoner mellom en generert nøkkel og en allerede eksisterende nøkkel, vil dette bli håndtert med lenkede lister.

Jeg valgte størrelsen på hash tabellen på bakgrunn av antallet navn i «navn.txt» (135).

Lastfaktoren vil i dette tilfellet bli rundt 75% (nærmere bestemt 0,745856), da den valgte størrelsen er satt til 181.

Antall kollisjoner per person endte opp med å bli rundt 0,303704.

Oppslag i hash-tabellen fungerer både på det første og eventuelt senere elementer i en lenket liste. Disse elementene vil ha samme nøkkel, men blir derfor søkt etter faktisk navn (for å traversere de lenkede listene).

Jeg har lagt inn et oppslag av et navn som ligger «ytterst» i en lenket liste på hash-tabellens indeks 100. I denne lenkede listen ligger det tre navn, og alle vil returnere korrekt navn.

**Videre (fra neste side) følger skjermbilder av utskrift fra programmet:**

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Programmet skriver først ut alle kollisjoner som oppstår under lesing fra fil og innsetting i hash-tabell. Deretter søkes det etter en person som befinner seg ytterst i en lenket liste (på indeks 100 i tabellen).

Her viser programmet at den først møter på Harry og deretter Helene, før Per til slutt blir funnet.

Til slutt skrives det ut informasjon om det totale antall kollisjoner som oppstod under innsetting, lastfaktoren og gjennomsnittlig antall kollisjoner per person.

## Deloppgave 2

Jeg valgte å lage hash tabellen i størrelse 10 000 019, som er det første primtallet etter 10 000 000.

De unike tallene ble generert ved å sette inn 1 som det første elementet, mens påfølgende tall ble lagt på de foregående verdiene med et tall mellom 10 og 100 (for å kunne forholde meg til int-størrelser på tallene i tabellen).

Deretter gjennomførte jeg mange ulike kjøringer på ulike mengder fyllingsgrad ifra talltabellen over til hash tabellen. Det ble brukt både «linear probing» og «double hashing» i innsettingene her, på to separate (men like) hash tabeller.

Antallet kollisjoner ble lagret som et attributt i hash tabellen som jeg kalte «collisions».

**Videre (fra neste side) følger de ulike eksperimentene i forbindelse med ulike lastfaktorer:**

**Målt tid per lastfaktor (uten lastfaktoren 1,00, se neste bilde)**

**A graph of a load factor

Description automatically generated**

**Målt tid per lastfaktor (inkludert lastfaktor 1,00)**

**A screenshot of a graph

Description automatically generated**

Vi kan se at det er en betydelig stor last for den lineære probingen å gjennomføre en fullstendig innsetting. Double hashing vil forholde seg relativ rask selv ved en lastfaktor på 1,00. Vi kan observere at lineær probing er tilnærmert lik double hashing ved lavere lastfaktorer, men at double hashing blir den raskeste ved lastfaktorer fra rundt 0,8 og oppover.

**Antall kollisjoner per lastfaktor (uten lastfaktorer 0,99 og 1,00, se neste bilde)**

**A screenshot of a graph

Description automatically generated**

Her kan vi se at antallet kollisjoner øker relativt sakte hos de to typene åpen adressering, men at den lineære probingen alltid møter på flere kollisjoner. Forskjellene mellom metodene øker desto nærmere vi kommer en lastfaktor på 1,00.

Vi kan (på neste bilde) se at det er en enorm forskjell i antallet kollisjoner, da den lineære vil ende opp med omtrent 166 ganger så mange kollisjoner som double hashing.

**Antall kollisjoner per lastfaktor (inkludert lastfaktorer 0,99 og 1,00)**

**A screenshot of a graph

Description automatically generated**

**Konklusjon:**

Vi kan se at det vil være tydelige forskjeller ved å ha litt ledig plass i en hash tabell, både med tanke på ressursbruk grunnet antall kollisjoner, men også tidsforbruk.

1. Vi kan se at det er en viss sammenheng mellom antall kollisjoner og tidsforbruk, da programmet vil bruke lengre tid på å sjekke hver indeks i tabellen kontra å benytte seg av double hashing. Dog er det svært lite forskjell i tidsforbruk mellom metodene ved lavere lastfaktorer (0,9 og lavere).
2. Etter testing, kan vi se at det vil være fornuftig å forholde seg til en lastfaktor rundt 0,8, altså rundt en 80 % fyllingsgrad. Det kunne ha vært fornuftig å teste nærmere for å finne en absolutt beste lastfaktor for dette tilfellet. Vi kan iallfall konkludere med at vi bør holde oss unna lastfaktorer over 0,9 i dette tilfellet.
3. Vi kan se likheter i ytelse og antall kollisjoner ved lavere lastfaktorer (0,8 og lavere), men at forskjellene mellom de to typene åpen adressering øker drastisk desto nærmere vi kommer en lastfaktor på 1,00 (hovedsakelig etter 0,9). Double hashing vil i dette tilfellet være en bedre tilnærming.
4. Små endringer i hash-funksjonene kan ha enorm påvirkning på antall kollisjoner og tidsforbruk!