

Øving 4 – Hashtabeller

Algoritmer og datastrukturer

Ingebrigt Hovind

Deloppgave 1:

Tabellengden er satt til 128 for å komme til nærmeste toerpotens.

Tester med mitt eget navn for å sjekke at det går an å hente ut navn som ligger i tabellen, tester med navn som ikke finnes i tabellen for å sjekke at dette også gir riktig resultat.

Kunne ha oppnådd mer gunstig lastfaktor dersom man hadde lagd kortere tabell, men hadde da måtte akseptert flere kollisjoner og tregere program med annen hashfunksjon.

Bruker C++ sin innebygde linkedlist for å håndtere kollisjoner.

```
Index 103:
Index 104:
Index 105: Mai Helene,Grosås => Henrik Latsch,Haugberg =>
Index 106:
Index 107: Stine,Rygh => Ilona,Podliashanyk =>
Index 108: Lars Brodin,Østby =>
Index 109:
Index 110:
Index 111: Karl Klykken,Labrador =>
Index 112:
Index 113: Lukas Øystein Normann,Stjernen =>
Index 114: Øyvind,Henriksen =>
Index 115: Ida Heggen,Trosdahl => Jenny Farstad,Blindheimsvik => Mattias Agentoft,Eggen => Arvid Jr,Kirkbakk =>
Index 116:
Index 117: Lea,Grønning =>
Index 118:
Index 119:
Index 120:
Index 121: Joakim Skogø,Langvand =>
Index 122: Olaf,Rosendahl =>
Index 123:
Index 124: Endré,Hadzalic =>
Index 125: Jostein Johansen,Aune =>
Index 126:
Index 127:
Lastfaktor: 0.671875
Kollisjoner per navn: 0.255814
Ingebrigt finnes i tabellen
```

Deloppgave 2:

Nærmeste toerpotens over 10 000 000 er $2^{24} = 16\,777\,216$, som er ca 65 % større enn antall tall, men pga hashfunksjonen jeg bruker så er man avhengig av å bruke toerpotens til tabellengden.

I hash 1 så ganger jeg det tilfeldige tallet med primtallet 2654435769 og så deler på 2^8 ved bruk av høyreskift for å hente ut de første 2^{24} fra unsigned int som kan inneholde verdier opp til 2^{32} .

```
unsigned hashOne(int num){  
    const unsigned knuth = 2654435769U;  
    //får et svar mellom 0 og  $2^{24}$   
    return num * knuth >> (32-24);  
}  
  
unsigned hashTwo(int num){  
    //gir kun oddetall  
    unsigned value = ((2*num+1)%twoToTwentyFour);  
    return (value);  
}
```

I hash 2 så lager jeg et oddetall mellom 0 og tabellengden, oddetall er alltid relativt primiske med toerpotenser, noe som betyr at jeg vil få dekt hele tabellen.

Ser at metoden min er mye tregere på Windows enn på Linux, men heldigvis så er det ikke så mange som bruker windows, så det går bra.

```
ingebrigt@ingebrigt-ThinkPad-L4  
rch=native -o main && ./main  
Millisekunder: 488  
Antall kollisjoner: 5209537  
Lastfaktor: 0.596046  
.. Millisekunder: 4322
```

```
Millisekunder: 1067  
  
D:\uni - 2\Algoritmer og datastrukturer\Oevinger\Oeving4\Del2>g++ OpenAddressingHash.cpp -Ofast -march=native -o main  
  
D:\uni - 2\Algoritmer og datastrukturer\Oevinger\Oeving4\Del2>main  
Millisekunder: 15551  
Antall kollisjoner: 1979766418  
Lastfaktor: 0.596046  
Millisekunder: 112  
  
D:\uni - 2\Algoritmer og datastrukturer\Oevinger\Oeving4\Del2>
```

Her også så kunne jeg ha oppnådd en mer gunstig lastfaktor ved å bruke en kortere tabellstørrelse, men på grunn av hashmetoden jeg ville bruke så måtte jeg ha en toerpotens som var lik tabellengden.