okw-besvarelse.md

Kort Besvarelse!

I filene sammens med dette har du begge C kodene for del 1 og del 2 av arbeidskravet. Koden når kjørt, svarer på alle spørsmålene i oppgaven. I oppgave 2 har jeg også lagt til flere `#define` for å gjøre det enklere å endre på parametere som tabellstørrelse, antall nøkler og fyllingsgrader om man vil prøve ut diverse konfigurasjoner.

Note: Jeg har brukt GitHub Copilot (GPT-5) i en større mengde sammenlignet med andre prosjekter i arbeidet med del 2 av oppgaven. Dette er notert i kommentarene i koden. Litt usikker på reglemanget rundt større bruk av KI til slike oppgaver, så kommenter gjerne hvis dette er problematisk. Jeg har i alle fall forsøkt å være åpen om det.

Del 2 - Oppgave 6 (det som ikke står i printf i koden)

Svarene på spørsmålene i del 2 baseres på følgende resultater fra kjøring av programmet:

```
- Fyllingsgrad 50%:
  Linear probing   -> kollisjoner: 249950, tid: 21.85 ms
  Double hashing   -> kollisjoner: 192945, tid: 25.71 ms

- Fyllingsgrad 80%:
  Linear probing   -> kollisjoner: 1574116, tid: 32.25 ms
  Double hashing   -> kollisjoner: 809419, tid: 38.51 ms

- Fyllingsgrad 90%:
  Linear probing   -> kollisjoner: 3754671, tid: 42.74 ms
  Double hashing   -> kollisjoner: 1402290, tid: 41.00 ms

- Fyllingsgrad 99%:
  Linear probing   -> kollisjoner: 24105781, tid: 85.75 ms
  Double hashing   -> kollisjoner: 3625102, tid: 45.01 ms

- Fyllingsgrad 100%:
  Linear probing   -> kollisjoner: 709072893, tid: 1175.08 ms
  Double hashing   -> kollisjoner: 12148224, tid: 69.10 ms
```

a) Tar fler kollisjoner mere tid?

- Ja, det er en klar sammenheng mellom antall kollisjoner og tid brukt.

b) Er det grenser på hvor full en hashtabell bør være?

- Ja, du ser tydelig sammenheng på tidsbruk på fyllingsgrad 99% og 100%. Spessielt for lineær probing øker tiden kraftig når tabellen er helt full.
- Tabellen vil selfølgelig være mere opptimal jo mindre fylt den er, men en total fyllingsgrad på 80-90% er ofte et bra kompromiss mellom minnebruk og ytelse. Og den bør i følge observasjonen av dette eksperimentet minst være under 100% full.

c) Er ytelsen ulik for ulike typer hashing?

- Ja, dobbel hashing er vesentlig bedre enn lineær probing når fyllingsgraden øker. Dette er fordi dobbel hashing sprer nøklene bedre utover tabellen og dermed reduserer antall kollisjoner. Men når fyllingsgraden er lav (50% og 80%) er forskjellen i ytelse ikke så stor, og man kan observere at lineær probing faktisk er raskere i disse tilfellene.

d) Andre interessante observasjoner?

- Ved høy fyllingsgrad (99% -> 100%) øker antall kollisjoner og tidsbruk dramatisk for lineær probing, mens dobbel hashing håndterer dette mye bedre.
- Ved 100% fyllingsgrad blir lineær probing ekstremt ineffektiv, der dobbel hashing fortsatt klarer å holde en relativt lav kollisjonsrate og tidsbruk.