Dynamisk lager

Vi ønsker at implementere et dynamisk lager. Lageret består af n lagerceller, $c_0, \ldots, c_{n-1}$, der til start alle er ledige. Et kaldene program, kaldet brugeren, skal kunne allokerer og deallokerer celler. Mere præsist skal vi understøtte følgende to operationer:

alloc find en værdi af i så at c_i er ledig. Noter at i er optaget, og returner i. Brugeren kan nu frit disponerer over c_i.

dealloc(i) Meld c_i ledig. Det forudsættes at brugeren havde allokeret c_i før kaldet.

Til implementeringen vil vi bruge en bit-map bestående af bits $b_0, \ldots, b_{n-1}$, der fortæller om cellerne er ledige eller optaget. Vi lader her $b_i = 1$ betyde at lagercelle c_i er optaget. Når systemet startes er $b_i = 0$ for alle i. Således starter alle lagerceller med at være ledige. I termer af bit-mappen, kan vi allokerer og deallokerer som følger:

alloc find en værdi af i så at $b_i = 0$. Sæt $b_i \leftarrow 1$, og returner i.

dealloc(i) sæt $b_i \leftarrow 0$.

I forbindelse med alloc, skal vi nu finde frem til en værdi af i så at $b_i = 0$. Vi ser på to mulige implementeringer, hvoraf kun den ene vil blive brugt i et givet lagersystem.

first-alloc — find den første ledige celle

- $i \leftarrow 0$;
- while $b_i = 1$ do $i \leftarrow i + 1$; (vi antager der er mindst en ledig celle)
- $b_i \leftarrow 1$;
- return i

next-alloc — find den næste ledige celle. Vi opererer her med en global variable q, der er sat til 0 når systemet starter med alle lagerceller ledige. Værdien af q ændres kun via kald til next-alloc.

- while $b_q = 1$ do $q \leftarrow q + 1 \bmod n$;
- $b_q \leftarrow 1$;
- return q

Det er klart, at hvis vi ser bort fra while-løkkerne, så tager alle operationer konstant tid, uanset implementeringen.

a. For $n = 4$, hvordan ændrer i, j og k sig i løbet af nedenstående 7 statements, hvis alloc er implemeteret med henholdsvis (a) first-alloc og (b) next-alloc.

$$i \leftarrow alloc; j \leftarrow alloc; dealloc(i); k \leftarrow alloc; i \leftarrow alloc; dealloc(j); j \leftarrow alloc; \ldots$$

I det følgende er n et vilkårligt stort tal. Antag at vi aldrig optager mere end $3/4$ af lageret — hvis brugeren forsøger at optage mere end $3n/4$ lagerceller stopper vores dynamiske lager blot med en fejlmeddelelse. Brugeren kommer nu med en sekvens af $N = 2n$ allokeringer og N deallokeringer, hvor allokeringerne og deallokeringerne er tilfældigt blandet mellem hinanden.

b. Hvor lang tid kan det i værste fald tage at udføre en alloc i sekvensen, hvis den (a) er implementeret med first-alloc, (b) er implementeret med next-alloc. Det interessante er, at finde ud af, hvor mange gange i henholdsvis q maximalt kan tælles op i while-løkken.

c. **(dette er det vigtigeste spøgsmål)** Hvor lang tid kan det maximalt tage at udføre en blandet sekvens af N alloc og N dealloc, hvis alloc (a) er implementeret med first-alloc, (b) er implementeret med next-alloc. Husk at højst $3/4$ af lageret må være optaget.

d. Hvad er så den amortiserede kompleksitet (aggregat-metoden) per operation med first-alloc henholdsvis next-alloc.