Ugeseddel: Introduktion til Datastrukturer

Philip Bille

Om denne uge

Litteratur *Introduction to Algorithms*, Cormen, Rivest, Leisersons og Stein (CLRS): Introduktion til Del III + Kap. 10.

Opgaver

1 Stakke og køer

1.1 CLRS [*o*] 10.1-1.

1.2 Opgave 5.1 i eksamenssættet fra 2011.

1.3 CLRS 10.1-2.

1.4 CLRS [*o*] 10.1-3.

1.5 CLRS 10.1-6.

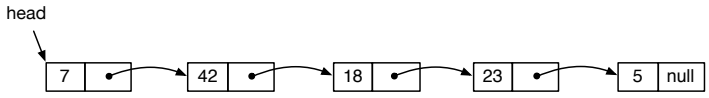
2 Algoritmer på hægtede lister Kig på den hægtede liste og algoritmerne FOO og BAR nedenfor. Løs følgende opgaver.

2.1 [*o*] Håndkør FOO(*head*).

2.2 [*o*] Forklar hvad FOO gør.

2.3 Håndkør BAR(*head*, 0).

2.4 Forklar hvad BAR gør.



Algoritme 1 Foo(*head*)

```
x = head
c = 0
while x ≠ null do
    x = x.next
    c = c + 1
end while
return c
```

Algoritme 2 Bar(x, s)

```
if x == null then
    return s
else
    return BAR(x.next, s + x.key)
end if
```

3 Implementation af hægtede lister Antag `x` er element i en enkelt-hægtet liste som beskrevet til forelæsningen. Løs følgende opgaver.

3.1 [o] Antag `x` ikke er det sidste element i listen. Hvad effekten af den følgende kodestump?

```
x.next = x.next.next;
```

3.2 [o] Lad `t` være et nyt element der ikke er i listen i forvejen. Hvad er effekten af følgende kodestump?

```
t.next = x.next;
x.next = t;
```

3.3 [o] Hvorfor gør følgende kodestump *ikke* det samme som ovenstående opgave?

```
x.next = t;
t.next = x.next;
```

4 Implementation af stakke og køer Løs følgende opgaver.

4.1 [†] Implementer en stak, der kan indeholde heltal ved hjælp af en enkelt-hægtet liste.

4.2 [†] Implementer en kø, der kan indeholde heltal ved hjælp af en enkelt-hægtede liste.

5 Sorterede hægtede lister Lad L være en enkelt-hægtet liste bestående af n heltal i *sorteret* rækkefølge. Løs følgende opgaver

5.1 Giv en algoritme til at indsætte et nyt tal i L således at listen bliver ved med at være sorteret. Din algoritme skal køre i $\Theta(n)$ tid. Skriv pseudokode for din algoritme.

5.2 Professor Gørtz foreslår at man kan forbedre indsættelsesalgoritmen ved at benytte binær søgning. Har hun ret?

6 Listevending Giv en algoritme til at "vende" en enkelt-hægtet liste om, dvs. producere en enkelt-hægtet liste bestående af de samme elementer i omvendt rækkefølge. Din algoritme skal køre i $\Theta(n)$ tid og ikke bruge mere end konstant plads udover pladsen brugt af selve listen.

7 Dynamiske tabeller og stakke Vi er interesseret i at implementere en stak ved hjælp af dynamiske tabeller uden at fastsætte en maksimal størrelse på tabellen til at starte med. Løs følgende opgaver.

7.1 [$*$] Generaliser dynamiske tabeller til også at kunne håndtere stakke, der "skrumper" undervejs (dvs. understøtter både PUSH og POP operationer). Køretiden af enhver sekvens af n operationer skal være $\Theta(n)$ og til ethvert tidspunkt skal din løsning bruge lineær plads i antallet af elementer i stakken.

7.2 [$**$] Vis hvordan man kan opnå $O(1)$ tid per stak operation med dynamiske tabeller og linæer plads i antallet af elementer i stakken. Kig kun på stakke der vokser og ignorer "skrump". *Hint:* Tænk på hvordan man kan fordele arbejdet.

8 Frivillig afleveringopgave: Dynamiske mængder med forening Vi er interesseret i at vedligeholde en familie af mængder af heltal $F = S_1, \ldots, S_k$ under følgende operationer.

- MAKE-SET(x): Tilføj mængden $\{x\}$ til F.
- REPORT(S_i): Rapporter alle elementerne i S_i.
- UNION(S_i, S_j): Tilføj mængden $S_i \cup S_j$ til F. S_i og S_j slettes fra F.
- DISJOINT-UNION(S_1, S_2): Ligesom UNION pånær at det antages at S_i og S_j er *disjunkte*, dvs, S_i og S_j ikke har nogle elementer til fælles. Hvis S_i og S_j ikke er disjunkte er resultatet af operationen udefineret.

F. eks. lad F bestå af 3 mængder $S_1 = \{2, 12, 5, 13\}$, $S_2 = \{6, 7, 1\}$ og $S_3 = \{8, 1, 7\}$. Et kald til DISJOINT-UNION(S_1, S_2) producerer $S_1 \cup S_2 = \{2, 12, 5, 13, 6, 7, 1\}$, hvorefter F består at $S_1 \cup S_2$ og S_3. Et kald til UNION$(S_1 \cup S_2, S_3)$ producerer mængden $S_1 \cup S_2 \cup S_3 = \{2, 12, 5, 13, 6, 7, 1, 8\}$ hvorefter F består af $S_1 \cup S_2 \cup S_3$. Løs følgende opgaver.

8.1 Giv en datastruktur, der understøtter MAKE-SET og DISJOINT-UNION i $O(1)$ tid og REPORT(S_i) i $\Theta(|S_i|)$ tid. *Hint:* benyt en passende listedatastruktur.

8.2 Udvid din datastruktur således at den også understøtter UNION og analyser tidskompleksiteten af din løsning.

8.3 [$*$] Giv en datastruktur, der understøtter MAKE-SET i $O(1)$ tid, REPORT(S_i) i $\Theta(|S_i|)$ tid og UNION(S_i, S_j) i $\Theta(|S_i| + |S_j|)$ tid (bemærk at DISJOINT-UNION ikke skal understøttes).