

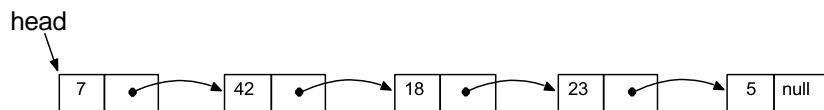
IDMA Exercises

- Stacks, Queues, and Lists -

Opgaver markeret med (ekstra) kan gemmes til resten af opgaverne er regnet. Pas på: Opgaver markeret med ****** er svære og ******* er meget svære.

1. **Algoritmer på hægtede lister** Kig på den hægtede liste og algoritmerne Foo og BAR nedenfor. Løs følgende opgaver:

- 1.1. Håndkør Foo(*head*).
- 1.2. Forklar hvad Foo gør.
- 1.3. Håndkør BAR(*head*, 0).
- 1.4. 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)
```

2. **Implementation af hægtede lister** Antag at x er et element i en enkelt-hægtet liste. Løs følgende opgaver.

- 2.1. Antag at x ikke er det sidste element i listen. Hvad er effekten af den følgende kodestump?
x.next =x.next.next;
- 2.2. 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;
- 2.3. Hvorfor gør følgende kodestump *ikke* det samme som ovenstående opgave?
x.next =t;
t.next =x.next;

3. Sorterede hægtede lister

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

- 3.1. 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.
- 3.2. Professor Gørtz foreslår at man kan forbedre indsættelsesalgoritmen ved at benytte binær søgning. Har hun ret?

4. Betragt nedenstående kø K implementeret ved hjælp af en tabel (et array). K.head = 3 og K.tail = 8.

1	2	3	4	5	6	7	8
		C	O	M	B	I	

Angiv hvordan køen ser ud efter følgende operationer: Enqueue(D), Enqueue(T), Dequeue(), Enqueue(U).

1	2	3	4	5	6	7	8

Stjerneopgaver (svære ekstraopgaver):

1. **Kø med stakke** [*] CLRS 10.1-6.

2. **Dynamiske mængder med forening**

Vi er interesseret i at vedligeholde en familie af mængder af heltal $F = S_1, \dots, S_k$ under følgende operationer.

- **MAKE-SET(x)**: Tilføj mængden $\{x\}$ til F .
- **REPORT(S_i)**: Rapporter (f.eks. udskriver) 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 på nær 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 af $S_1 \cup S_2$ 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.

- 2.1. 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.

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

- 2.3. [*] 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).

3. [**] Lad der være givet en liste af sorteret tal. Ud over at hvert listeelement har next, prev, og key vil vi også give elementet en variable jump. Betragt følgende kode

Algoritme 3 Jumpinit(L)

```
x = L.head
while x <> NIL do
    x.jump = x.next
    x=x.next
```

Algoritme 4 Jumpshort(L)

```
x = L.head
while x.jump <> NIL do
    x.jump = x.jump.jump
    if x.jump <> NIL x=x.jump
```

Algoritme 5 Jumpset(L)

```
x = L.head
Jumpinit(L)
while x.jump <> NIL do Jumpshort(L)
```

- 3.1. Forklar hvad effekten er af jumpset er
 - 3.2. Hvad er tidskompleksiteten?
 - 3.3. En knude kan få ændret sin jump pegere flere gange. Vi foreslår nu følgende ændring: Til hver knude laver vi en tabel, som indeholder alle de jump-pegere som en knude har fået tildelt på et tidspunkt. Lav pseudokode. Hvor meget plads bruger vi? Hvordan hænger pladsen sammen med tiden fra opgave 5.2
 - 3.4. Vis hvordan vi kan søge efter om et tal er i listen, indsætte og slette elementer (husk at listen er sorteret).
 - 3.5. Hvad er tidskompleksiteten af din løsning fra opgave 5.4?
4. [**] Lad der til hver træknude være tilknyttet en farve. Givet et træ T og en knude x fra træet lader vi T_x være træet der består af x 's efterkommere (bemærk x er en efterkommer til sig selv). Lad træer være repræsenteret på samme måde som i CLRS figur 10.10, dog vælge vi for søskende en dobbelthæget repræsentation.
 - 4.1. Vis at man givet en knude x fra et træ T i en tid proportional med træets størrelse kan beregne antal knuder i T_x .
 - 4.2. Vis hvordan man kan givet en knude x fra træet T i $O(1)$ tid kan dele træet i to træer et bestående af T_x og et bestående af de andre knuder. Vi kalder operationen $Fjern(x)$.
 - 4.3. Lad alle knuderne til at starte med have en farve. Sikre at $Fjern(x)$ bevare egenskaben at to knuder har samme farve hvis og kun hvis de er i det samme træ – du kan antage at alle knuder starter med samme farve, og du må tildele nye farver når du kalder $Fjern(x)$. Hvor lang tid bruger $Fjern(x)$ nu?
 - 4.4. [**] Konstruer $Fjern(x)$ således at givet ét træ T med n knuder så vil en sekvens af n $Fjern$ operation tage $O(n \log n)$ tid og bevare egenskaben at to knuder har samme farve hvis og kun hvis de er i samme træ.

Bemærkninger / Credits

Nogle opgaver er stærkt inspireret af opgaver stillet af Philip Bille og Inge Li Gørtz i kurset *Algoritmer og Datastrukturer*, på DTU (<http://www2.compute.dtu.dk/courses/02105+02326/2015/#generelinfo>).

Some exercises are heavily inspired by the course *Algorithms and Data Structures* at DTU given by Philip Bille and Inge Li Gørtz (<http://www2.compute.dtu.dk/courses/02105+02326/2015/#generelinfo>).