Ugeseddel 4

Litteratur:

- CLRS Introduktion til part III
- CLRS kapitel 10 (10.3 kun overfladisk)
- Modular arithmetic, side 54, sætning (3.8) og (3.9)
- CLRS kapitel 11 intro, 11.1, 11.2 (dog ikke beviser for theorem 11.1 og theorem 11.2)
- CLRS kapitel 11.3 dog ikke 11.3.3. Dertil er følgende kursorisk (da vi endnu ikke vil beskæftige os meget med f.eks. bitrepræsentation):
    - Paragraffen fra "Interpreting keys as natural numbers" til men før 11.3.1. Bemærk dog antagelsen i sidste linje: "In what follows, we assume that the keys are natural numbers"
    - I 11.3.1 er teksten kursorisk fra "For example, m should not be a ….".
    - I 11.3.2 er teksten kursorisk fra "Suppose that the word size of the machine is m…"
- CLRS 11.4 til og med Linear probing samt Theorem 11.6 (dog ikke beviset herfor)

Noter:

- 4.1 Stakke og køer - lister noter CLRS 10
- 4.2-3 hashing noter CLRS 11

Mål for ugen:

- Lister: Effektiv konstruktion og anvendelser
- Stakke og køer
- Træer (matematisk fundament gennemgås senere)
- Hvad en hashfunktion er, og hvornår disse bruges
- Hashing: med lister, uniform hashing, load factor**,** division og multiplikationsmetoden**,** og open addressing med fokus på lineær probering

Forelæsninger (og spørgertimer):

Tirsdag: CLRS kapitel 10, Stephen

- Spørgetimen tirsdag: Om CLRS kapitel 10, Stephen
- Torsdag: Lidt fra CLRS kapitel 10, og mest fra CLRS kapitel 11, Stephen
- Spørgetimen torsdag: Mest om CLRS kapitel 11, lidt om ugeopgaven, Stephen.

Opgaver tirsdag:

Pas på: Opgaver markeret med "*" er svære, "" er meget svære, og "***" har du ikke en chance for at løse.**

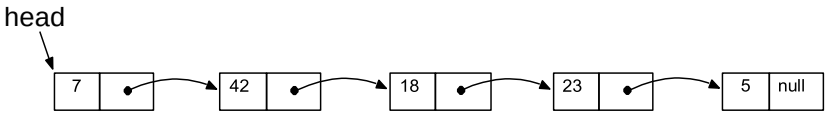
1. Algoritmer på hægtede lister Kig på den hægtede liste og algoritmerne FOO og BAR nedenfor. Løs følgende opgaver:

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. Effektive listeoperationer

3.1. CLRS 10.2-2

3.2. CLRS 10.2-3

4. Sorterede hægtede lister

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

4.1. 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.

4.2. Professor Gørtz foreslår at man kan forbedre indsættelsesalgoritmen ved at benytte binær søgning. Har hun ret? Binær søgning så du f.eks. ved toppunkt søgning.

5. Stakke og køer

5.1 CLRS 10.1-1.

5.2 CLRS 10.1-3.

5.3 CLRS 10.1-2 (ekstra)

6. Træer

6.1. CLRS 10.4-1

6.2. CLRS 10.4-2

6.3. CLRS 10.4-3 (ekstra)

6.4. CLRS 10.4-4 (ekstra)

7. Sammenligning af lister (ekstra)

7.1. CLRS 10-1

Opgaver torsdag:

1. Direkte hashing

1.1. CLRS 11.1-1

1.2. CLRS 11.1-2 (ekstra)

2. Hashing med kæder CLRS 11.2-2

3. Mere Hashing

3.1. CLRS 11.3-1

3.2. CLRS 11.3-4

4. Linear probing og open addressing

4.1. CLRS 11.4-1 dog kun for linear probing

4.2. CLRS 11.4-2

5. Håndkøring og egenskaber ved hashing

5.1. Indsæt følgende nøglesekvens $K = 7, 18, 2, 3, 14, 25, 1, 11, 12, 1332$ i tabel af størrelse 11 vha. af hægtet hashing med hashfunktionen $h(k) = k \bmod 11$.

5.2. Indsæt følgende nøglesekvens $K = 2, 32, 43, 16, 77, 51, 1, 17, 42, 111$ i tabel af størrelse 17 vha. linæer probering med hashfunktionen $h(k) = k \bmod 17$.

5.3. Slet 111 og 51 fra tabellen produceret i opgave 14.2.

5.4. Antag at vi laver sletning i lineær probering *uden* at markere indgange som slettet (vi fjerner bare elementet hvis vi finder det). Giv en sekvens af ordbogoperationer, der viser at dette ikke virker korrekt.

5.5. Lad K være en sekvens af nøgler gemt i en tabel A vha. hægtet hashing. Givet A kan man effektivt finde det maksimale element i K?

6. Eksempler på Hashing

6.1. Lad H være en hægtet hashtabel (chained hashing) af størrelse 7 med hashfunktion $h(x) = 3x \bmod 7$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 4, 1, 12, 8, 10, 11.

6.2. Lad H være en hægtet hashtabel (chained hashing) af størrelse 5 med hashfunktion $h(x) = 3x \bmod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 5, 1, 11, 7, 10, 3.

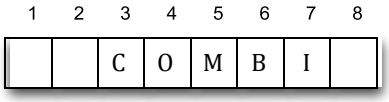
6.3. Lad H være en hashtabel med linær probering (linear probing) af størrelse 5 med hashfunktion $h(x) = x \bmod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 6, 2, 7, 16.

6.4. Indsæt nøglesekvensen $70, 5, 2, 23, 80, 13, 26, 8$ i en hashtabel, der anvender hægtet hashing og hashfunktion $h(k) = k \bmod 9$. Tegn indholdet af tabellen efter indsættelse af nøglesekvensen.

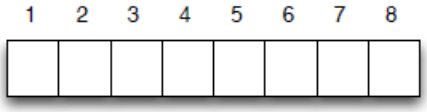
6.5. Indsæt nøglesekvensen $14, 48, 55, 74$ i en hashtabel, der anvender lineær probering og hashfunktion $k \bmod 7$. Tegn indholdet af tabellen efter indsættelse af nøglesekvensen

7. Tabeller og hashing

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



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



7.2. Lad H være en hægtet hashtabel (chained hashing) af størrelse 5 med hashfunktion $h(x) = x \bmod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 6, 7, 3, 14, 2

7.3. Lad H være en hashtabel med linær probering (linear probing) af størrelse 5 med hashfunktion $h(x) = x \bmod 5$. Angiv hvordan hashtabellen H ser ud efter indsættelse af tallene 6, 7, 3, 14, 2

Stjerneopgaver (svære ekstraopgaver):

1. Kø med stakke [*] CLRS 10.1-6.

2. Listevending

2.1 [*] CLRS 10.2-7

2.2 [*] Lav en rekursiv procedure som vender en liste. Bruger den ligeledes konstant ekstra plads?

3. Træer

3.1 [*] CRLS 10.4-5

3.2 [*] CLRS 10.4-6

4. 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 (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 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.

4.1. 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.

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

4.3. [$*$] 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).

5. Direkte hashing [*] CLRS 11.1-4

6. [$*$] **Sortering af små universer:** Lad $A[1..n]$ være en tabel af heltal fra $\{0, \ldots, n-1\}$. Giv en algoritme til at sortere A i $O(n)$ tid. *Hint:* start med at sætte tallene ind i en hægtet hashtabel med en passende hashfunktion som du definerer.

7. [**] 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)
```

7.1. Forklar hvad effekten er af jumpset er
7.2. Hvad er tidskompleksiteten?
7.3. 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 16.2
7.4. Vis hvordan vi kan søge efter om et tal er i listen, indsætte og slette elementer (husk at listen er sorteret).
7.5. Hvad er tidskompleksiteten af din løsning fra opgave 16.4?

8. [**] Lad der til hver træknude være tilknyttet en farve. Givet et træ T og en knude x fra træet lader vi Tx 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 dobbelthægtet repræsentation.
8.1. Vis at man givet en knude x fra et træ i en tid proportional med træets størrelse kan beregne antal knuder i Tx .
8.2. Vis hvordan man kan givet en knude x fra træet i O(1) tid kan dele træet i to træer et bestående af Tx og et bestående af de andre knuder. Vi kalder operationen Fjern(x).
8.3. 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?
8.4. [**] 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:

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.