Introduktion til datastrukturer

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

Philip Bille

Introduktion til datastrukturer

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

Datastrukturer

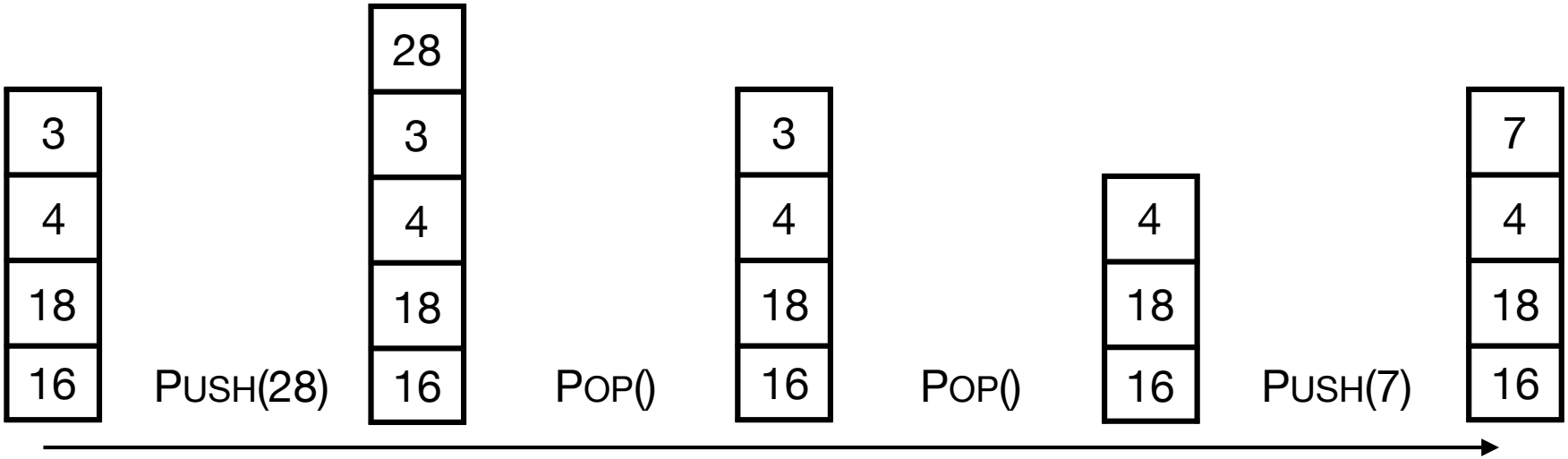
- Datastruktur. Metode til at organise data så det kan søges i/tilgås/manipuleres effektivt.
- Mål.
  - Hurtig
  - Kompakt
- Terminologi.
  - Abstrakt vs. konkret datastruktur.
  - Dynamisk vs. statisk datastruktur.

Introduktion til datastrukturer

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

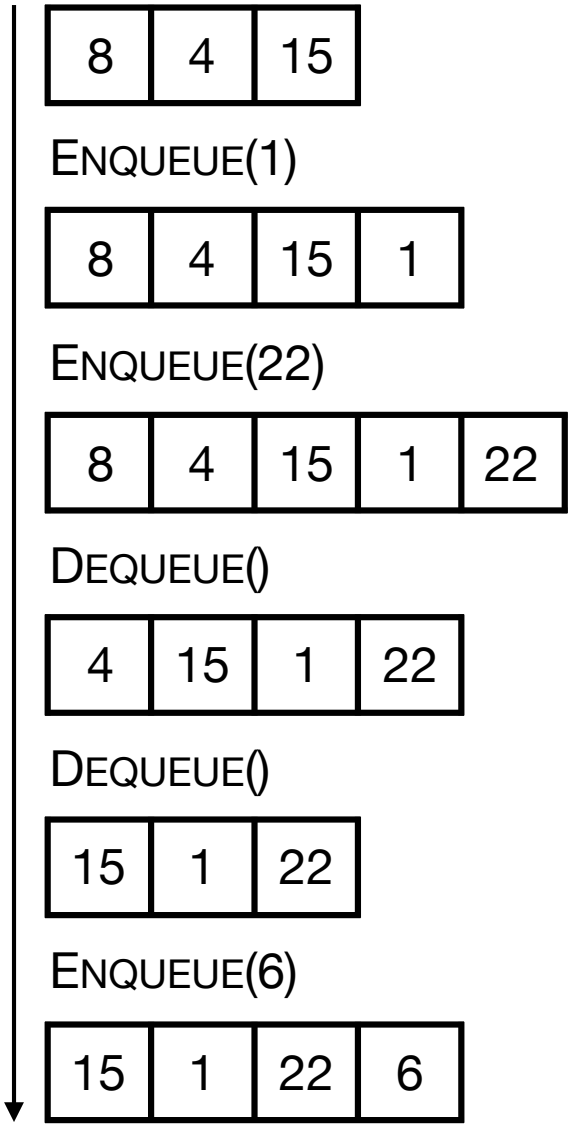
Stak

- Stak. Vedligehold en dynamisk sekvens (stakken) S af elementer under følgende operationer.
  - PUSH(x): tilføj et nyt element x til S.
  - POP(): fjern og returner det seneste tilføjede element i S.
  - ISEMPTY(): returner sand hvis S ikke indeholder nogle elementer.



Kø

- Kø. Vedligehold en dynamisk sekvens (køen) K af elementer under følgende operationer.
  - ENQUEUE(x): tilføj et nyt element x til K
  - DEQUEUE(): fjern og returner det tidligst tilføjede element i K.
  - ISEMPTY(): returner sand hvis K ikke indeholder nogle elementer.

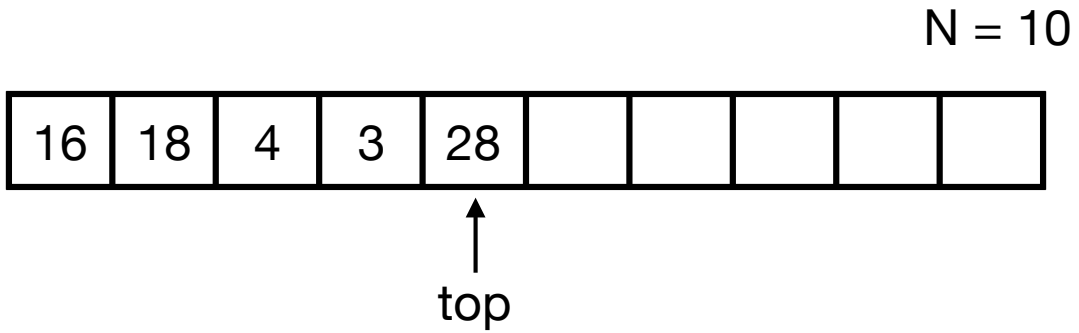


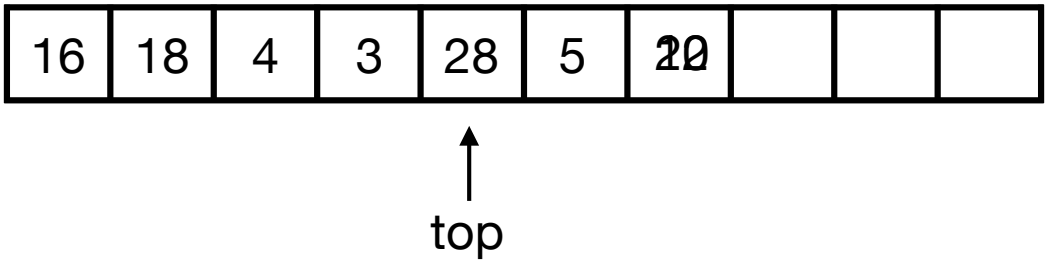
Anvendelser

- Stakke.
  - Virtuelle maskiner
  - Parsing
  - Funktionskald
  - Backtracking
- Køer.
  - Skedulering af processer
  - Buffering
  - Breddeførst søgning

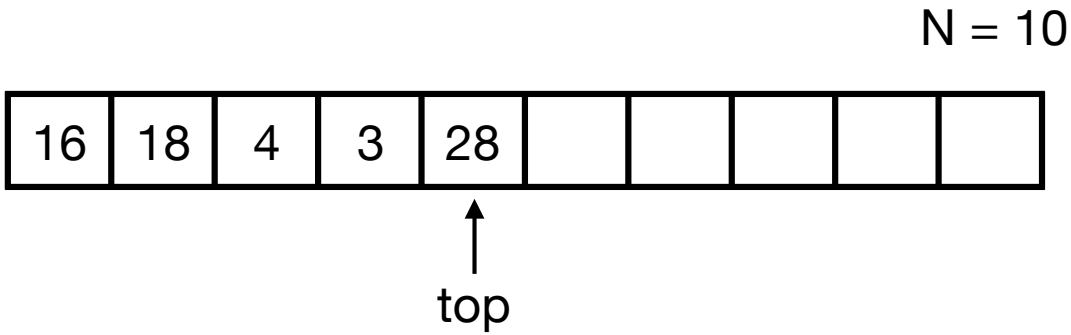
Implementation af stak med tabel

- Stak. Stak med kapacitet N vha. tabel.
- Datastruktur.
  - Tabel S[0..N-1]
  - Index top i S.
- Operationer.
  - PUSH(x): Tilføj x på S[top+1], sæt top = top + 1
  - POP(): returner S[top], sæt top = top - 1
  - ISEMPTY(): returner sand hvis og kun hvis top = -1.
  - Tjek for overløb og underløb i PUSH og POP.




16
18
4
3
28
5
top

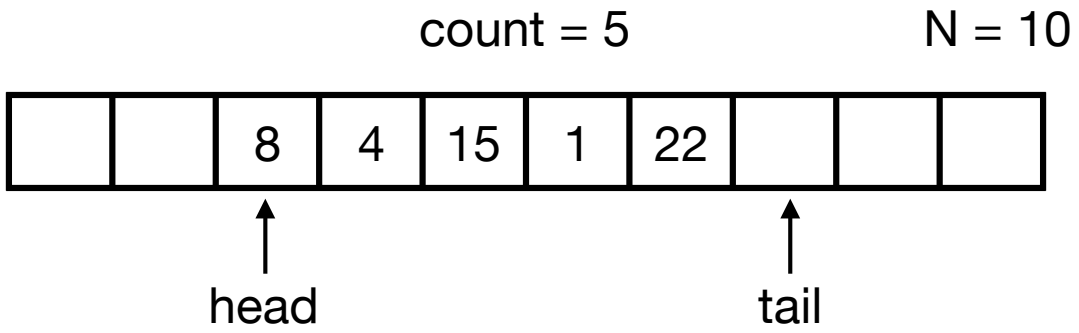
Implementation af stak med tabel

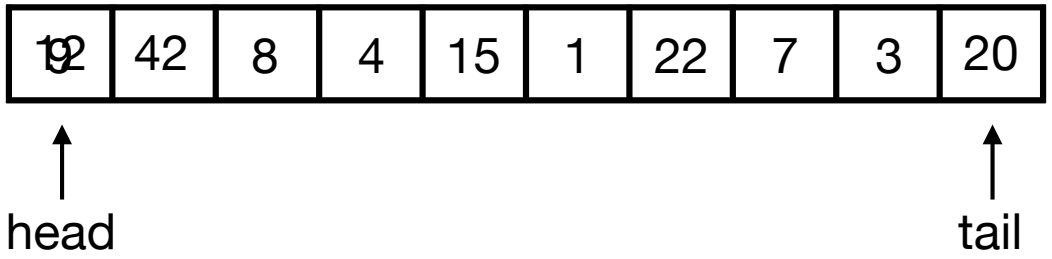


- Tid.
  - PUSH i Θ(1) tid.
  - POP i Θ(1) tid.
  - ISEMPTY i Θ(1) tid.
- Plads.
  - Θ(N) plads.
- Mangler.
  - Vi skal kende kapacitet N fra start.
  - Vi spilder plads når antal elementer er << N.

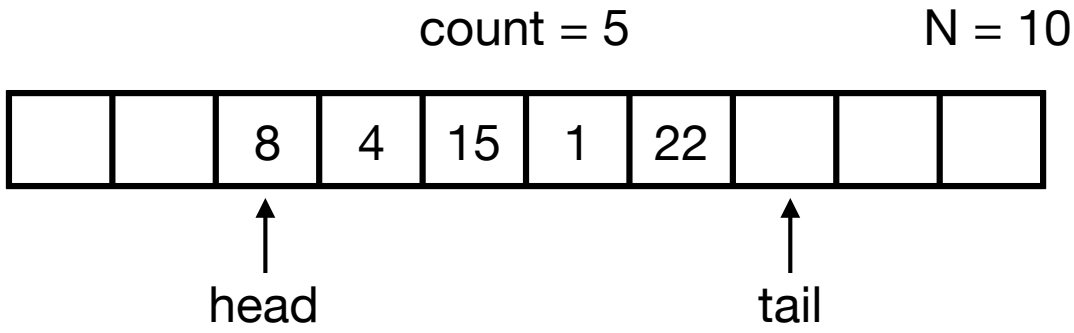
Implementation af kø med tabel

- Kø. Kø med kapacitet N vha. tabel.
- Datastruktur.
  - Tabel S[0..N-1]
  - Indeks head (tidligst indsatte element) og tail (næste ledige element) i S og tæller count (antal elementer i kø).
- Operationer.
  - ENQUEUE(x): Tilføj x på S[tail], opdater count og tail cyklisk.
  - DEQUEUE(): returner S[head], opdater count og head cyklisk.
  - ISEMPTY(): returner sand hvis og kun hvis count = 0.
  - Tjek for overløb og underløb i DEQUEUE og ENQUEUE.




42
8
4
15
1
22
7
3
20
head
tail

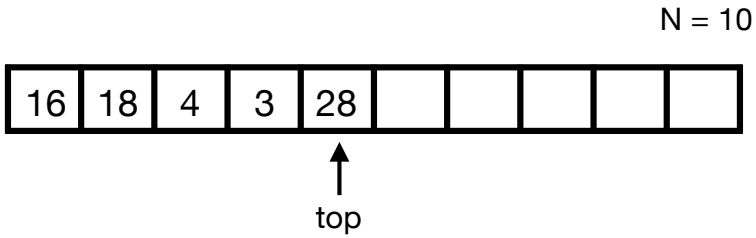
Implementation af kø med tabel



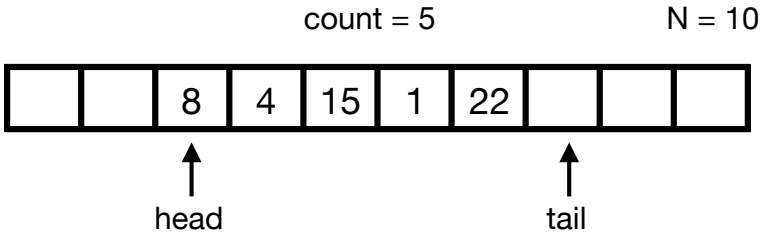
- Tid.
  - ENQUEUE i Θ(1) tid.
  - DEQUEUE i Θ(1) tid.
  - ISEMPTY i Θ(1) tid.
- Plads.
  - Θ(N) plads.
- Mangler.
  - Vi skal kende kapacitet N fra start.
  - Vi spilder plads når antal elementer er << N.

Stakke og køer

- Stak.
  - Tid. PUSH, POP, ISEMPTY i Θ(1) tid.
  - Plads. Θ(N)



- Kø.
  - Tid. ENQUEUE, Dequeue, ISEMPTY i Θ(1) tid.
  - Plads. Θ(N)



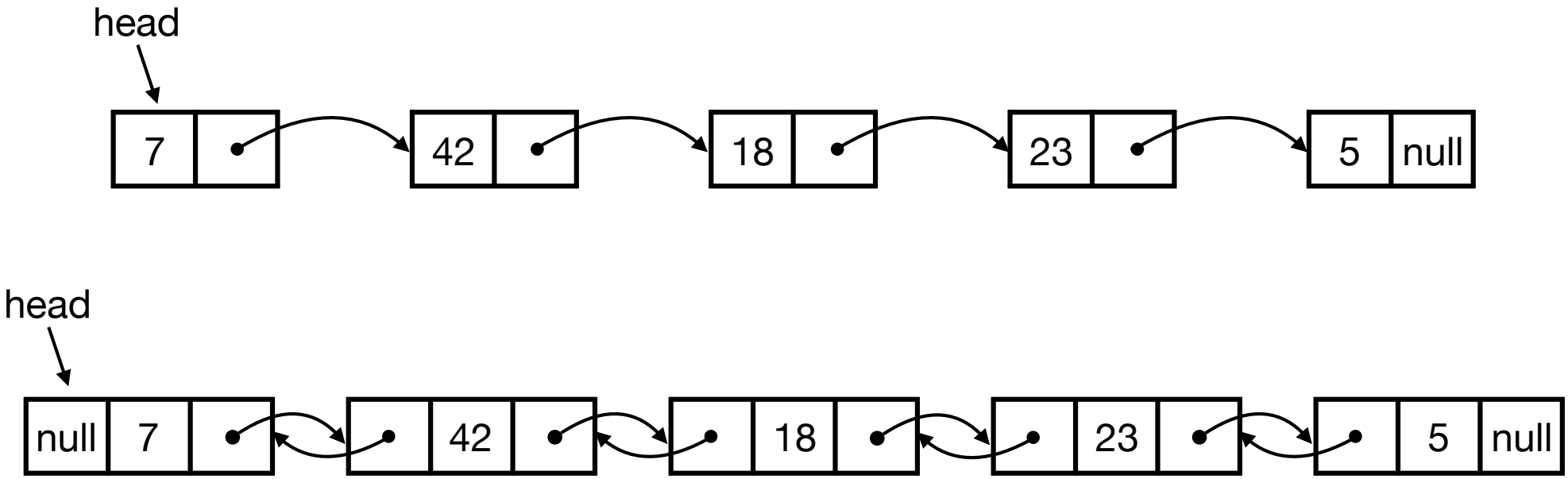
- Udfordring. Kan vi komme ned på lineær plads med samme tid?

Introduktion til datastrukturer

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

Hægtede lister (linked lists)

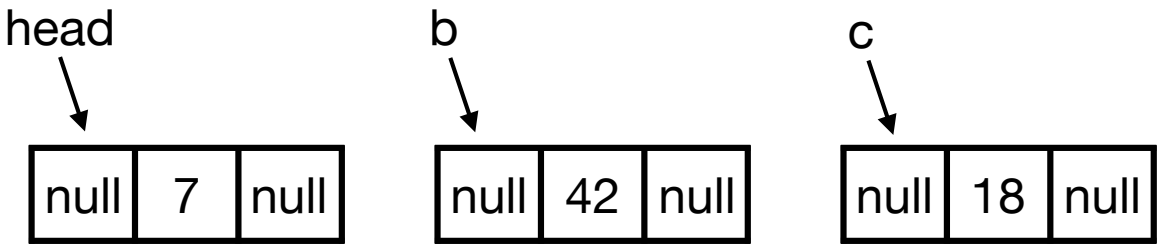
- Hægtede lister.
  - Datastruktur til at vedligeholde en dynamisk sekvens af elementer i lineær plads.
  - Rækkefølge af elementer bestemt af referencer/pegere kaldet hægter.
  - Effektiv at indsætte og fjerne elementer eller sammenhængende dele af elementer.
  - Dobbelt-hægtede vs enkelt-hægtede.

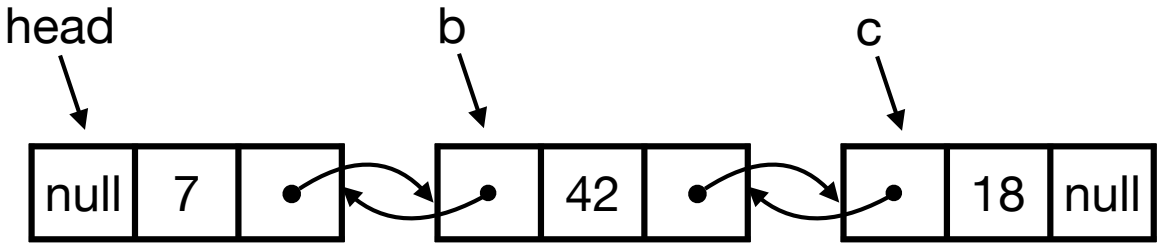


Hægtede lister

- Dobbelt-hægtede lister i Java.

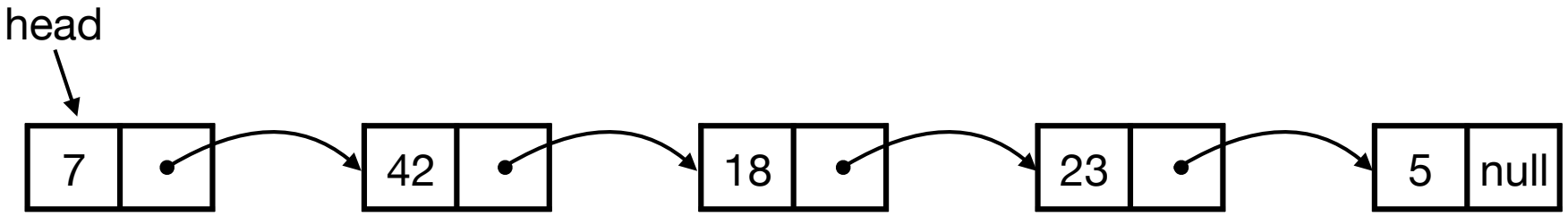
```
class Node {
    int key;
    Node next;
    Node prev;
}

Node head = new Node();
Node b = new Node();
Node c = new Node();
head.key = 7;
b.key = 42;
c.key = 18;

head.prev = null;
head.next = b;
b.prev = head;
b.next = c;
c.prev = b;
c.next = null;
```

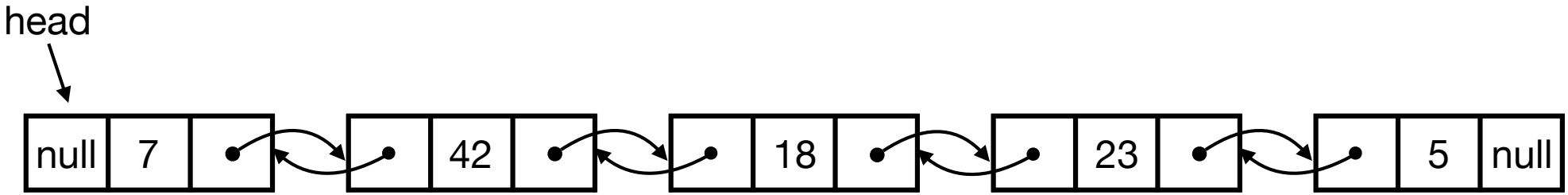




Hægtede lister

- Simple operationer.
  - SEARCH(head, k): returner knude med værdi k i listen. Returner null hvis den ikke findes.
  - INSERT(head, x): indsæt knude x i starten af listen. Returner ny head.
  - DELETE(head, x): fjern knude x i listen.





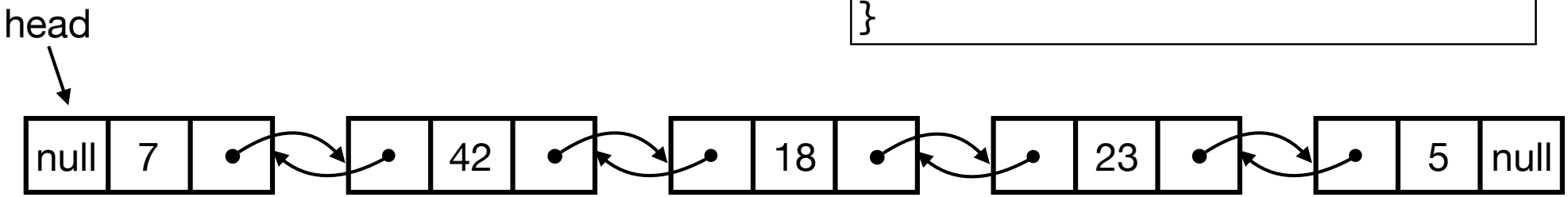
Hægtede lister

- Operationer på dobbelthægtet liste i Java.

```
Node Search(Node head, int value) {
    Node x = head;
    while (x != null) {
        if (x.key == value) return x;
        x = x.next;
    }
    return null;
}
```

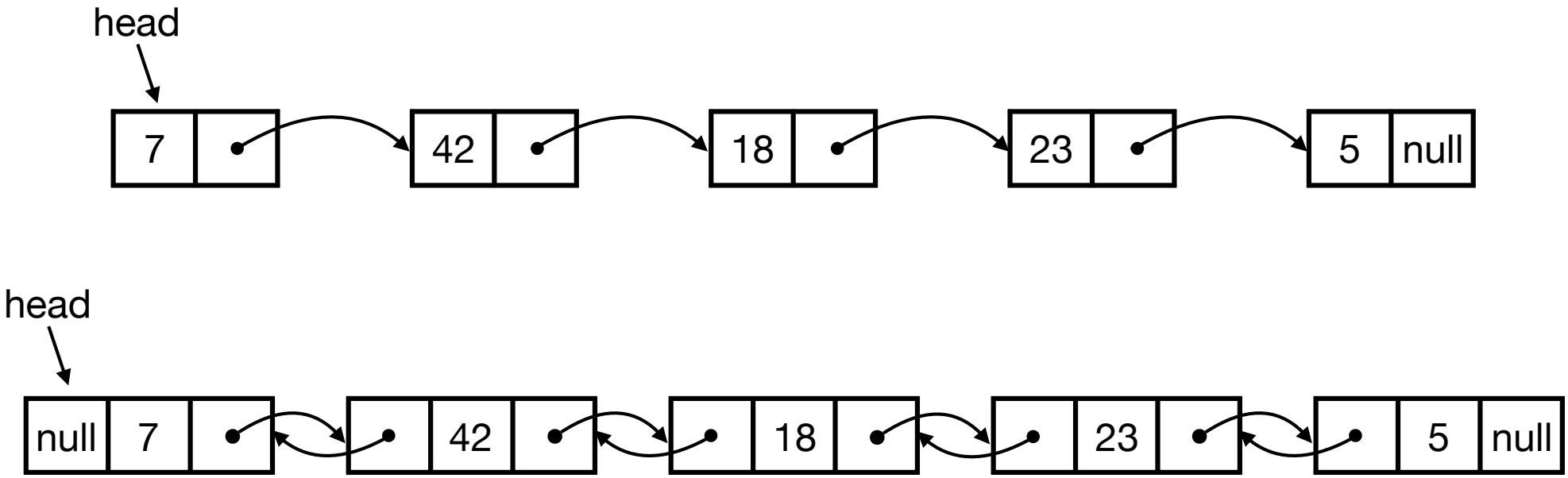
```
Node Insert(Node head, Node x) {
    x.prev = null;
    x.next = head;
    head.prev = x;
    return x;
}
```

```
Node Delete(Node head, Node x) {
    if (x.prev != null)
        x.prev.next = x.next;
    else head = x.next;
    if (x.next != null)
        x.next.prev = x.prev;
    return head;
}
```



- Opgave. Lad p være en ny knude med værdien 10 og lad q være knuden i listen med værdi 23. Håndkør `Search(head,18)`, `Insert(head,p)` og `Delete(head,q)`.

Hægtede lister



- Tid. Hvor hurtigt kører operationerne?
  - SEARCH i Θ(n) tid
  - INSERT og DELETE i Θ(1) tid.
- Plads.
  - Θ(n)

Implementation af stak og kø med hægtede lister

- Opgave. Overvej hvordan man kan implementere stakke og køer med hægtede lister effektivt.

- Stak. Vedligehold en dynamisk sekvens (stakken) S af elementer under følgende operationer.
  - PUSH(x): tilføj et nyt element x til S.
  - POP(): fjern og returner det seneste tilføjede element i S.
  - ISEMPTY(): returner sand hvis S ikke indeholder nogle elementer.

- Kø. Vedligehold en dynamisk sekvens (køen) K af elementer under følgende operationer.
  - ENQUEUE(x): tilføj et nyt element x til K
  - DEQUEUE(): fjern og returner det tidligst tilføjede element i K.
  - ISEMPTY(): returner sand hvis K ikke indeholder nogle elementer.

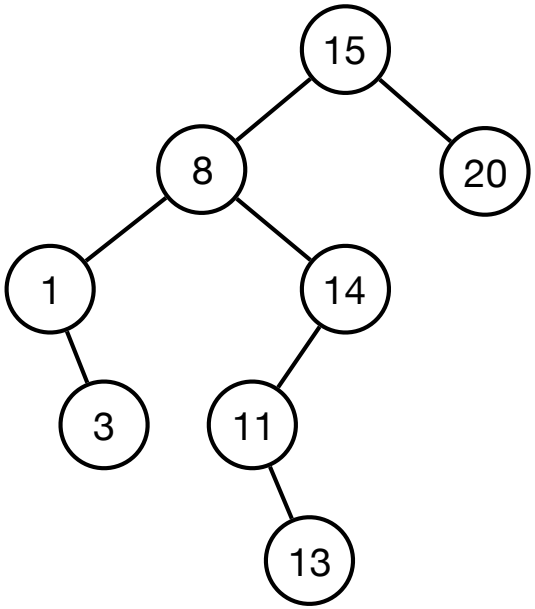
Stakke og køer

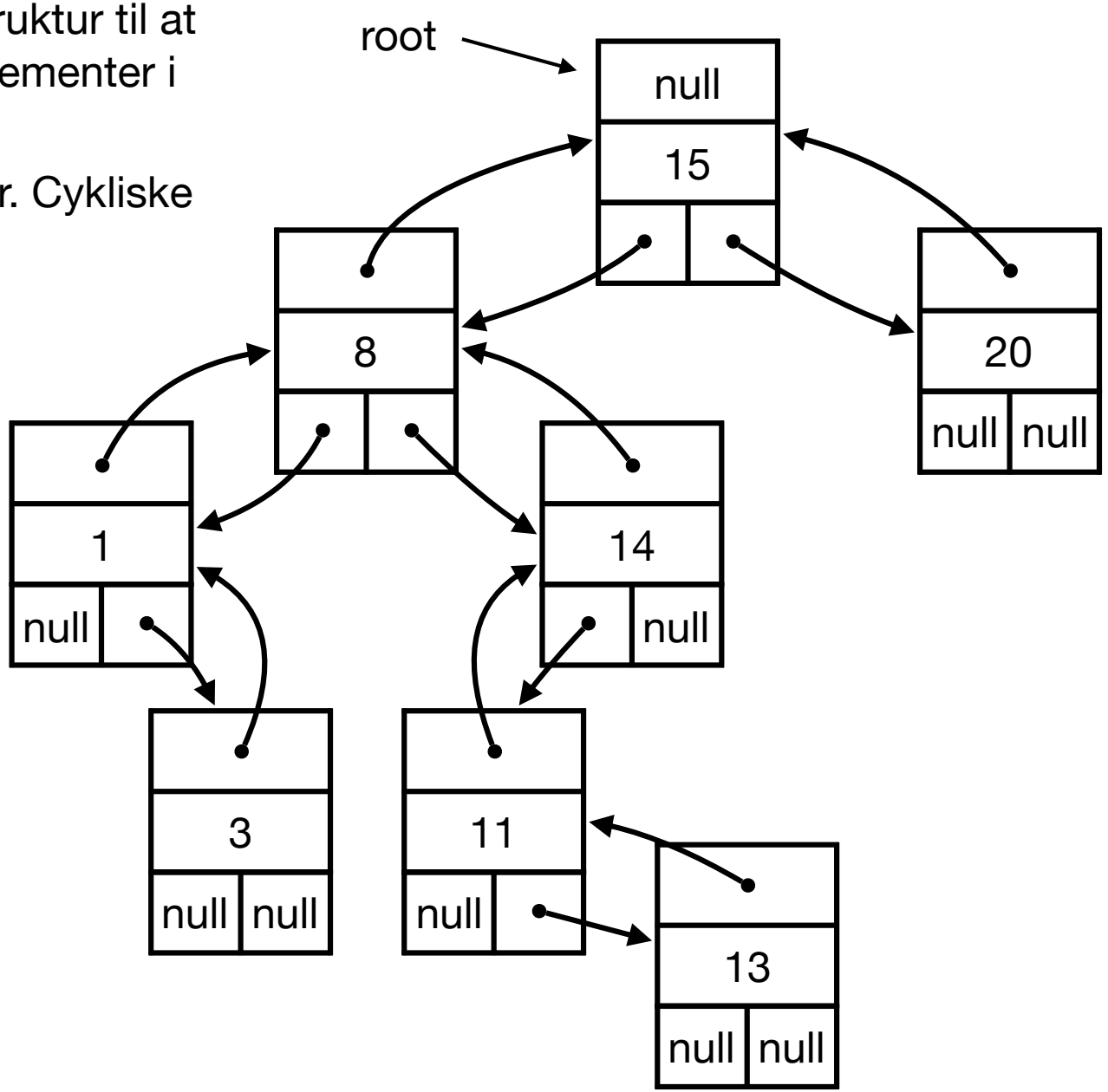
- Stak og kø implementeret med hægtet liste
- Stak.
  - Tid. PUSH, POP, ISEMPTY i Θ(1) tid.
  - Plads. Θ(n)

- Kø.
  - Tid. ENQUEUE, Dequeue, ISEMPTY i Θ(1) tid.
  - Plads. Θ(n)

Hægtede lister

- Hægtet liste. Fleksibel datastruktur til at vedligeholde en sekvens af elementer i lineær plads.
- Andre hægtede datastrukturer. Cykliske lister, træer, grafer, …





Introduktion til datastrukturer

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller

Stak med dynamisk tabel

- Udfordring. Kan vi implementere en stak effektivt med tabel(ler)?
  - Behøver vi fastsætte en øvre grænse på antallet af elementer?
  - Kan vi komme ned på lineær plads og konstant tid?

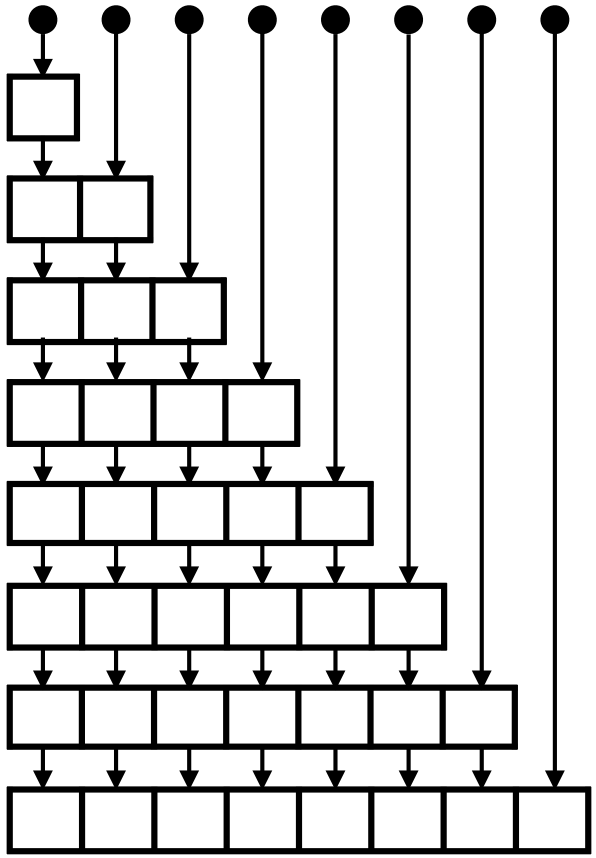
Dynamiske tabeller

- Mål.
  - Implementer en stak vha. tabel(ler) i $\Theta(n)$ plads for n elementer.
  - Så hurtige operationer som muligt.
  - Kun fokus på PUSH. Ignorer POP og ISEMPTY indtil videre.

- Løsning 1.
  - Start med tabel af størrelse 1.
- PUSH(x):
  - Opret ny tabel af størrelse +1.
  - Flyt alle elementer til ny tabel.
  - Slet gammel tabel.

Dynamiske tabeller

- PUSH(x):
  - Opret ny tabel af størrelse +1.
  - Flyt alle elementer til ny tabel.
  - Slet gammel tabel.



- Tid. Hvor meget tid tager n PUSH operationer?
  - PUSH i tager Θ(i) tid: byg tabel af størrelse i, flyt i-1, elementer og indsæt nyt element.
- Samlet tid. $1 + 2 + 3 + 4 + \ldots + n = \Theta(n^2)$
- Plads. Θ(n)

- Udfordring. Kan vi gøre noget smartere?

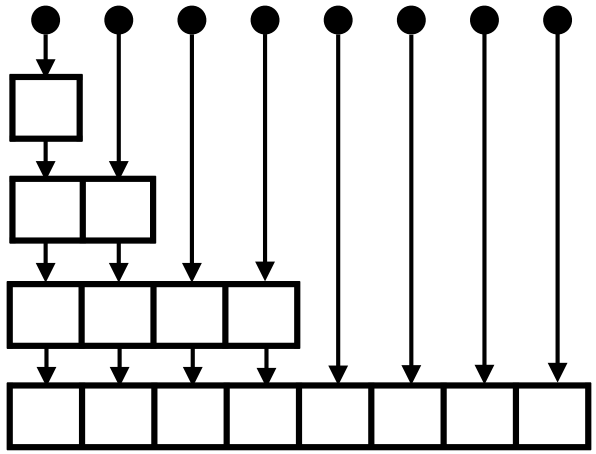
Dynamiske tabeller

- Ide. Kopier kun elementer en gang i mellem.

- Løsning 2.
  - Start med tabel af størrelse 1.
- PUSH(x):
  - Hvis tabel er fuld (antallet af elementer i stak er lig tabellens størrelse).
    - Opret ny tabel af dobbelt størrelse.
    - Flyt elementer over i ny tabel.
    - Slet gammel tabel.

Dynamiske tabeller

- PUSH(x):
  - Hvis tabel er fuld (antallet af elementer i stak er lig tabellens størrelse).
    - Opret ny tabel af dobbelt størrelse.
    - Flyt elementer over i ny tabel.
    - Slet gammel tabel.



- Tid. Hvor meget tid tager n PUSH operationer?
  - PUSH 2^k tager $\Theta(2^k)$ tid: byg tabel af størrelse 2^{k+1}, flyt 2^k elementer og indsæt nyt element.
  - Alle andre PUSH tager $\Theta(1)$ tid.
- Samlet tid. $1 + 2 + 4 + 8 + 16 + \ldots + 2^{\lfloor \log n \rfloor} + n = \Theta(n)$
- Plads. $\Theta(n)$

Dynamiske tabeller

- Stak med dynamisk tabel.
  - n PUSH operationer i Θ(n) tid og plads.
  - Kan udvides til n PUSH, POP og ISEMPTY operationer i Θ(n) tid.
- Køretiden er amortiseret Θ(1) per operation (een operation kan tage lang tid, men tiden for enhver sekvens af k operationer er Θ(k))
- Med snedige tricks kan man deamortisere løsning til Θ(1) værstefaldskøretid per operation.

- Kø med dynamisk tabel.
  - Samme resultater som stak.
- Global genobygning.
  - Dynamisk tabel er eksempel på global genopbygning (*global rebuilding*).
  - General teknik til at gøre statiske datastrukturer dynamiske.

Stakke og køer

Datastruktur	PUSH	POP	ISEMPTY	Plads
Tabel med kapacitet N	Θ(1)	Θ(1)	Θ(1)	Θ(N)
Hægtet liste	Θ(1)	Θ(1)	Θ(1)	Θ(n)
Dynamisk tabel med udvidelse	Θ(n)†	Θ(1)†	Θ(1)	Θ(n)
Dynamisk tabel med fordobling	Θ(1)†	Θ(1)†	Θ(1)	Θ(n)
Dynamisk tabel med deamortiseret fordobling	Θ(1)	Θ(1)	Θ(1)	Θ(n)

† = amortiseret

Introduktion til datastrukturer

- Datastrukturer
- Stakke og køer
- Hægtede lister
- Dynamiske tabeller