Introduktion til datastrukturer

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

Datastrukturer

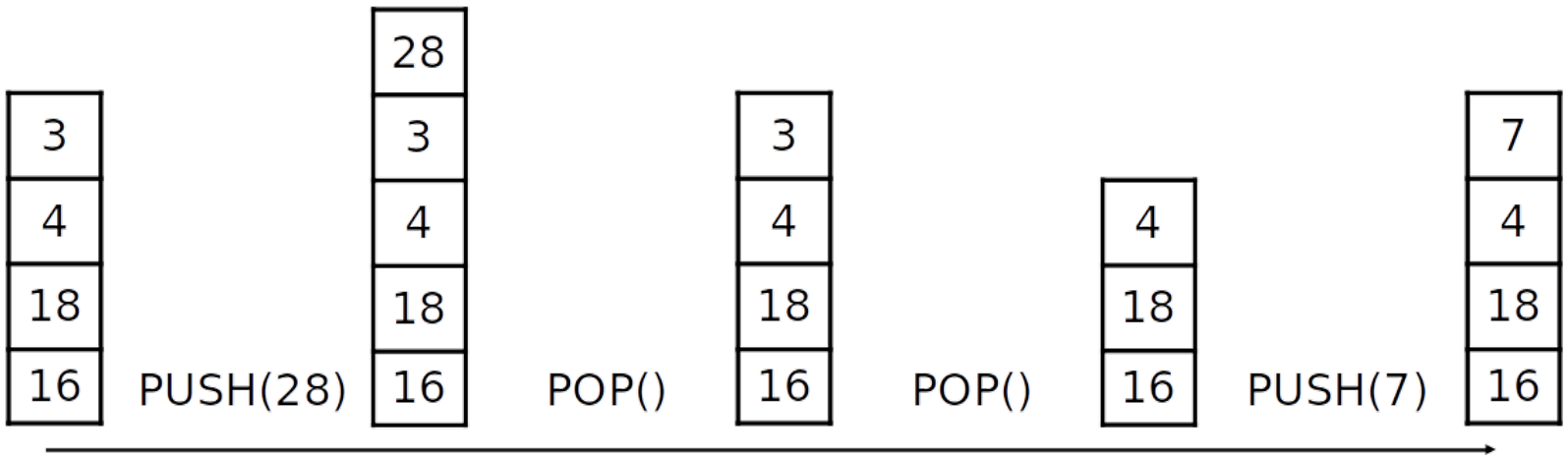
- Datastruktur: metode til at organisere data så det effektivt kan søges i, tilgås, ændres, ...
- Mål:
  - Hurtig,
  - Kompakt.
- Terminologi:
  - Abstrakt vs. konkret datastruktur
  - Dynamisk vs. statisk datastruktur

Introduktion til datastrukturer

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

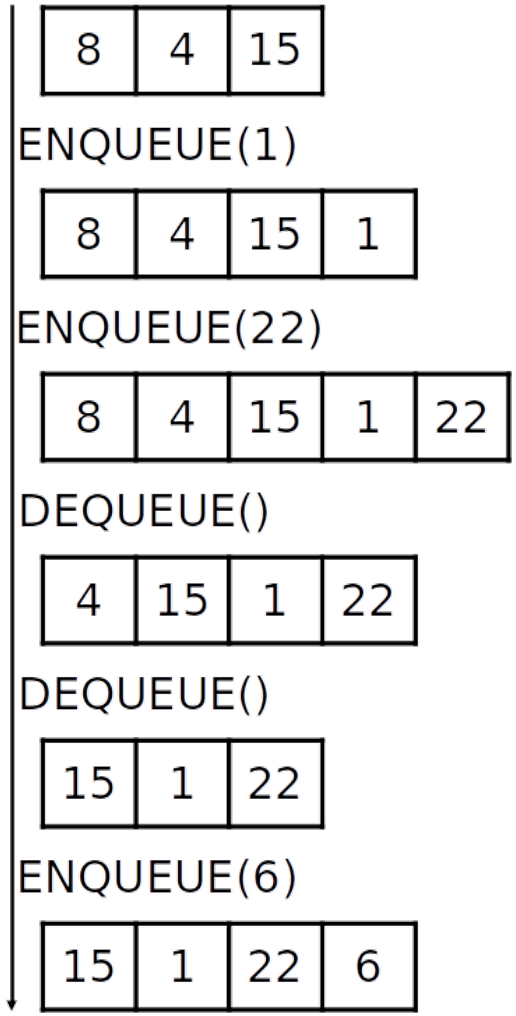
Stak (stack)

- Stak: Vedligehold en dynamisk sekvens (stakken) S af elementer under følgende operationer:
  - PUSH(x): tilføj element x til S
  - POP(): fjern og returner det senest tilføjede (nyeste) element fra S.
  - ISEMPTY(): Er S tom? (sand/falsk.)



Kø (queue)

- Kø. Vedligehold en dynamisk sekvens (køen) Q af elementer under følgende operationer:
  - ENQUEUE(x): tilføj elementet x til Q.
  - DEQUEUE(): fjern og returner det tidligst tilføjede (ældste) element fra Q.
  - ISEMPTY(): Er Q tom? (sand/falsk.)

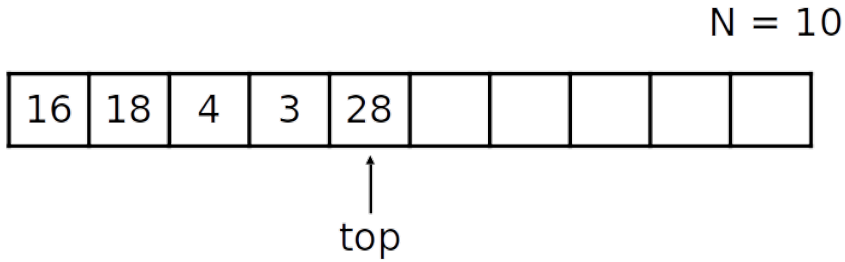


Anvendelser

- Stakke:
  - Parsing
  - Funktionskald
  - Backtracking
- Køer:
  - Skedulering af processer
  - Buffering
  - Bredde-først søgning på grafer

Implementation af stak ved hjælp af array

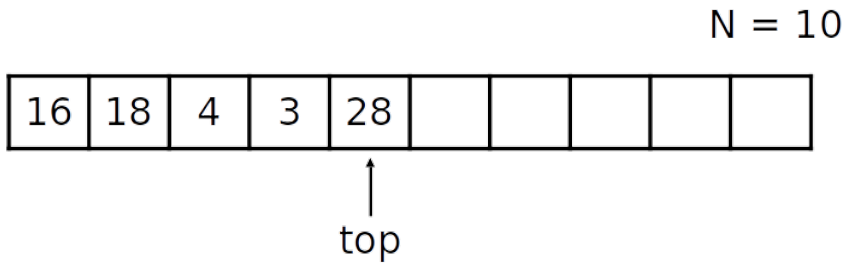
- Stak. Stak med kapacitet N.
- Datastruktur.
  - Array `S[0..N-1]`.
  - Index `top` i arrayet `S`.



- Operationer
  - PUSH(x): Tilføj x på plads `top+1` i S og forøg `top` med 1.
  - POP(): Returner `S[top]` og formindsk `top` med 1.
  - ISEMPTY(): Er `top` lig med tallet -1? (sand/falsk.)
  - Check for overløb og underløb.
- Eksempel

  $N = 10$ PUSH(2) PUSH(1) POP PUSH(7)

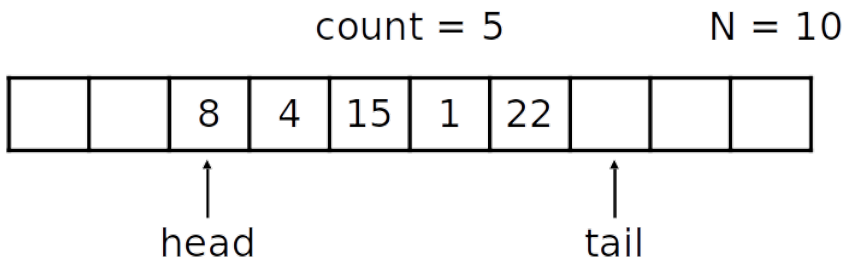
Implementation af stak ved hjælp af array



- Tid
  - PUSH i $\Theta(1)$ tid,
  - POP i $\Theta(1)$ tid,
  - ISEMPTY i $\Theta(1)$ tid.
- Plads
  - $\Theta(N)$ plads.
- Ulemper
  - Kapacitet N kendt fra start.
  - Spilder plads når $|S| \ll N$.

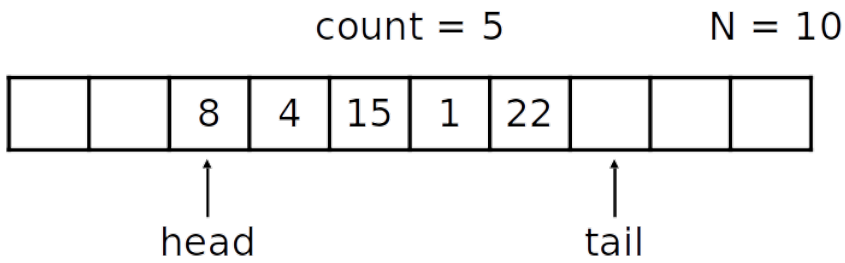
Implementation af kø ved hjælp af array

- Kø. Kø med kapacitet N.
- Datastruktur.
  - Array `Q[0..N-1]`.
  - Index `head` i arrayet `Q`.
  - Index `tail` i arrayet `Q`.
  - Tæller `count` (tal mellem 0 og N.)



- Operationer
  - ENQUEUE(x): Tilføj x på plads `tail` i `Q`, forøg `count` og forøg `tail` cyklisk med 1.
  - DEQUEUE(): Returner `Q[head]` og opdater count og opdater `head` cyklisk.
  - ISEMPTY(): Er `count` lig med 0? (sand/falsk.)
  - Check for overløb og underløb.
- Eksempel: ENQUEUE(7) ENQUEUE(34) ENQUEUE(59) DEQUEUE()

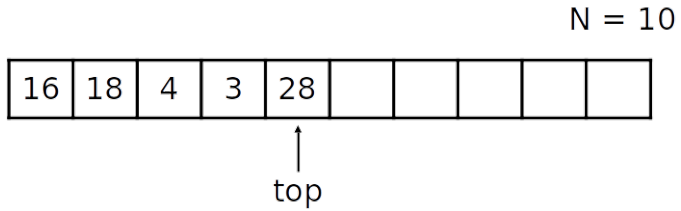
Implementation af kø ved hjælp af array

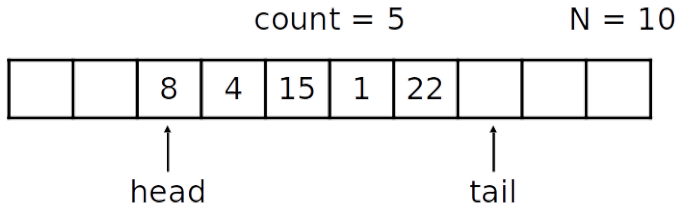


- Tid
  - ENQUEUE i $\Theta(1)$ tid,
  - DEQUEUE i $\Theta(1)$ tid,
  - ISEMPTY i $\Theta(1)$ tid.
- Plads
  - $\Theta(N)$ plads.
- Ulemper
  - Kapacitet N kendt fra start.
  - Spilder plads når $|S| \ll N$.

Stakke og køer

- Stak.
  - Tid: PUSH, POP, ISEMPTY i $\Theta(1)$ tid,
  - Plads: $\Theta(N)$, hvor N er kapaciteten.
- Kø.
  - Tid: ENQUEUE, DEQUEUE, ISEMPTY i $\Theta(1)$ tid,
  - Plads: $\Theta(N)$, hvor N er kapaciteten.





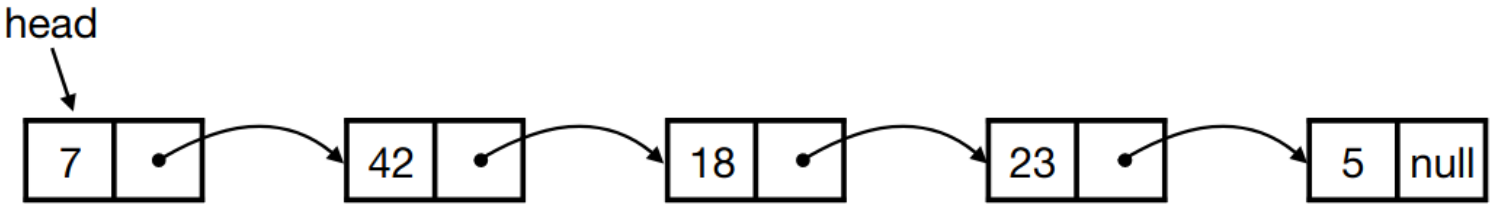
- Udfordring: Kan vi komme ned på lineær plads og samme tid?

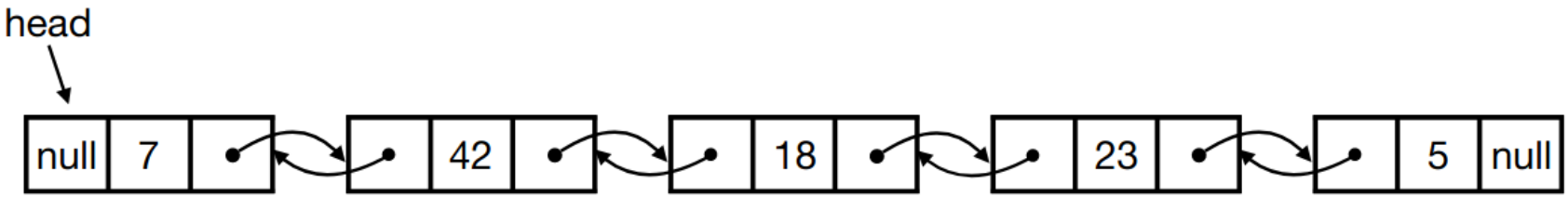
Introduktion til datastrukturer

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

Hægtede lister

- 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 eller fjerne elementer eller sammenhængende dele af elementer.
  - Dobbelt-hægtede eller enkelt-hægtede.



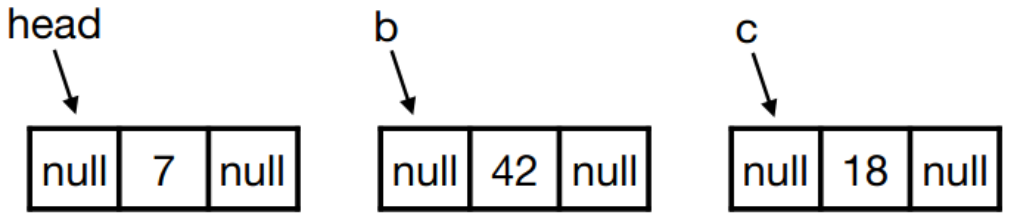


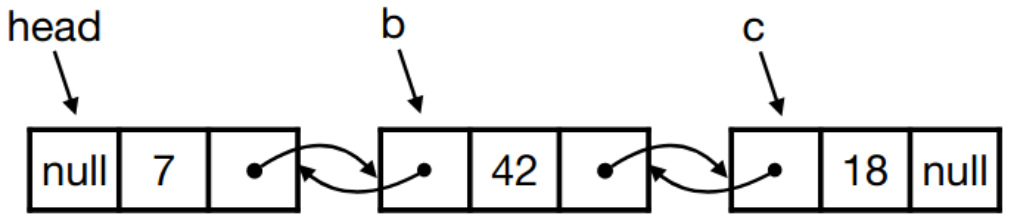
Hægtede lister – i Java

- Dobbelthægtede 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 – operationer

- Operationer:
  - SEARCH($\texttt{head}, x$): Returnér knude med værdi x i listen, eller `null` hvis den ikke findes.
  - INSERT($\texttt{head}, x$): Indsæt knuden x i starten af listen, returner ny head.
  - DELETE($\texttt{head}, x$) Slet knude x i listen.

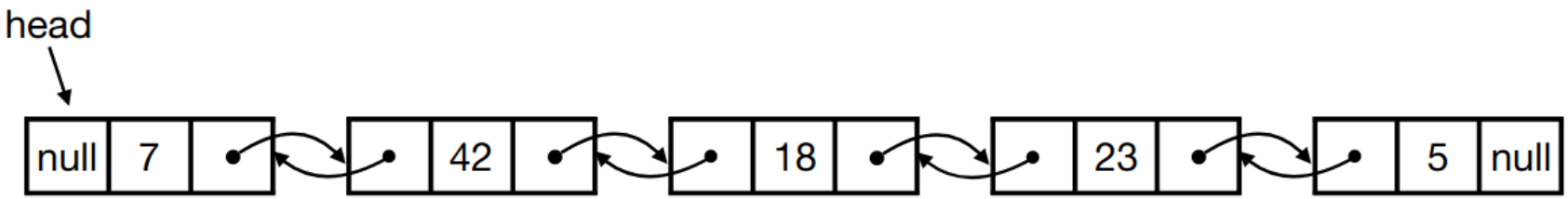
Hægtede lister – operationer i Java

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

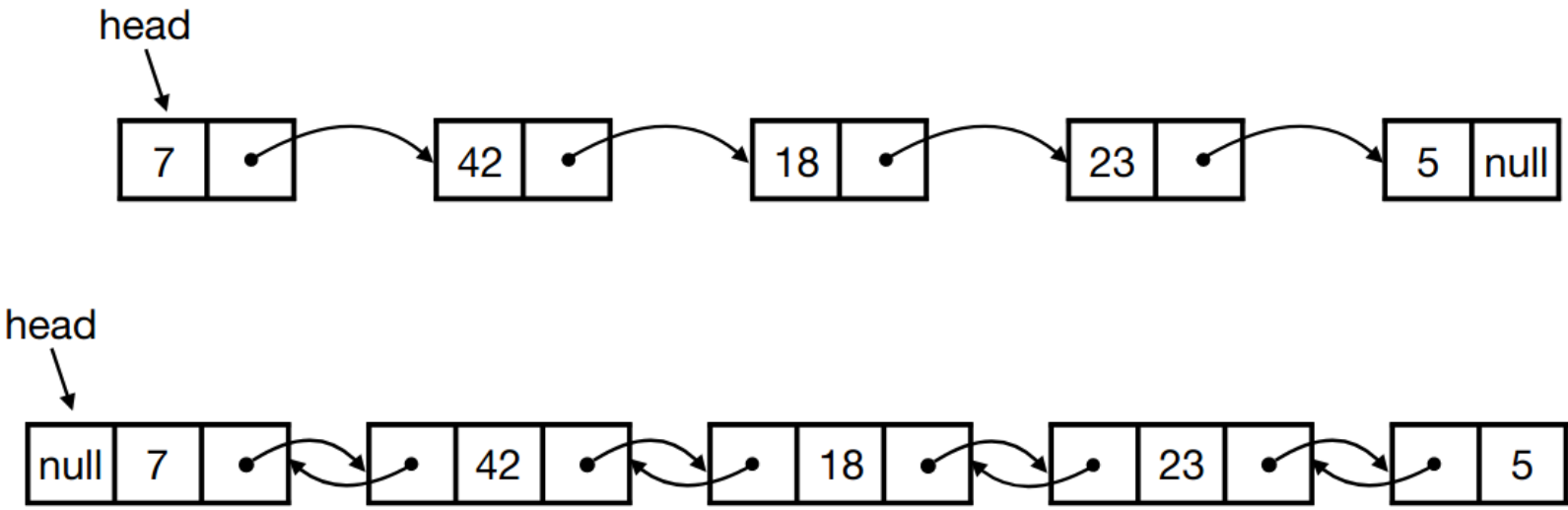
```
Node Delete(Node head, Node x) {
    if (x.prev != null)
        x.prev.next = x.next;
    else head = x.next;
    if (x.next != null)
        x.next.prev = x.prev;
    return head;
}
```

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

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



- Tid
  - SEARCH: $\Theta(n)$ tid.
  - INSERT: $\Theta(1)$ tid.
  - DELETE: $\Theta(1)$ tid.
- Plads
  - $\Theta(n)$ plads.

Stakke og køer – implementation via hægtede lister

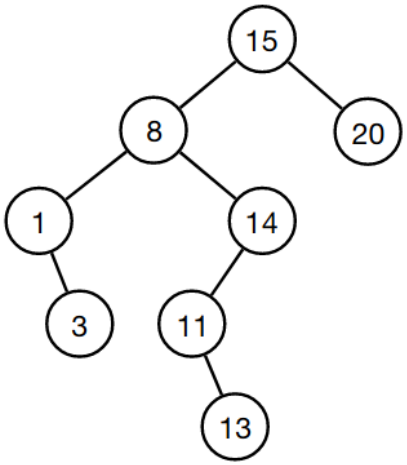
- Opgave: Implementere stakke og køer ved brug af hægtede lister.
- Stak En dynamisk sekvens (stakken) S under
  - PUSH(x): tilføj element x,
  - POP(): fjern og returner det nyeste element i S,
  - ISEMPTY(): er stakken tom? (sand/falsk.)
- Kø En dynamisk sekvens (køen) Q under
  - ENQUEUE(x): tilføj et nyt element til Q,
  - DEQUEUE(): fjern og returner det ældste element i Q,
  - ISEMPTY(): er køen tom? (sand/falsk.)

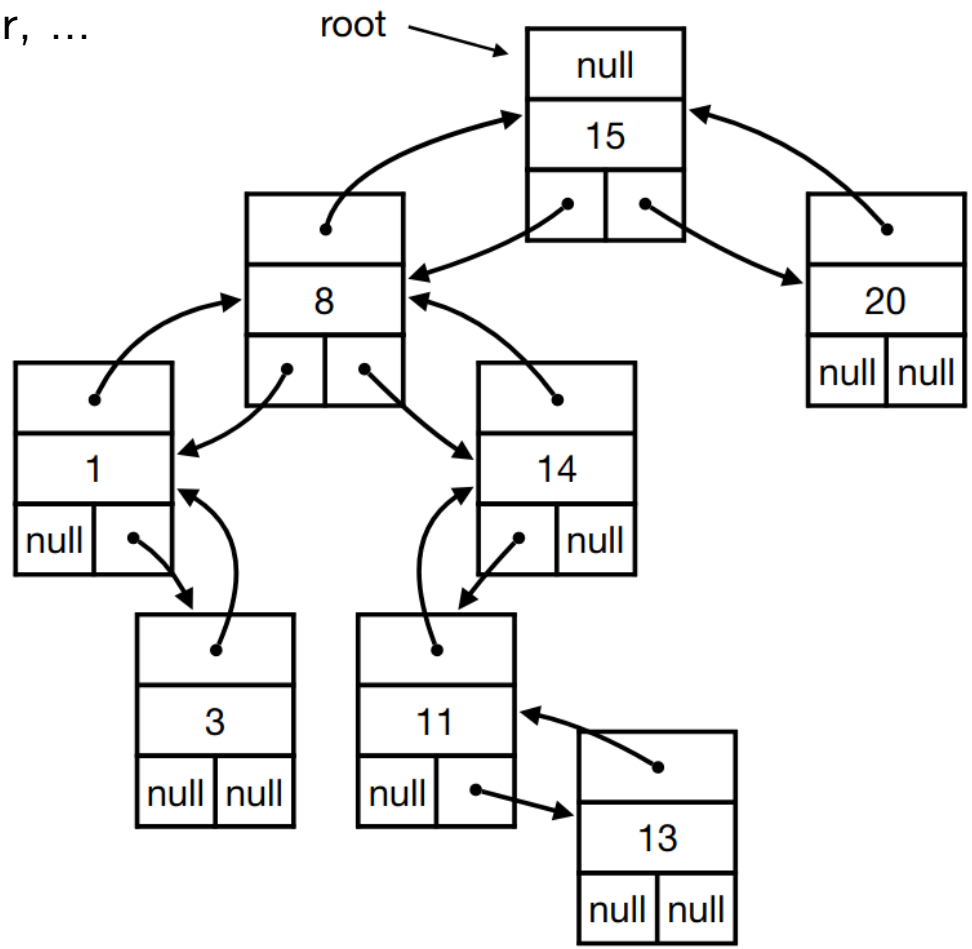
Stakke og køer

- Stak eller kø med n elementer implementeret via hægtede lister.
- Stak.
  - Tid: PUSH, POP, ISEMPTY i $\Theta(1)$ tid,
  - Plads: $\Theta(N)$.
- Kø.
  - Tid: ENQUEUE, DEQUEUE, ISEMPTY i $\Theta(1)$ tid,
  - Plads: $\Theta(n)$.

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 arrays

Stak implementeret via dynamisk array

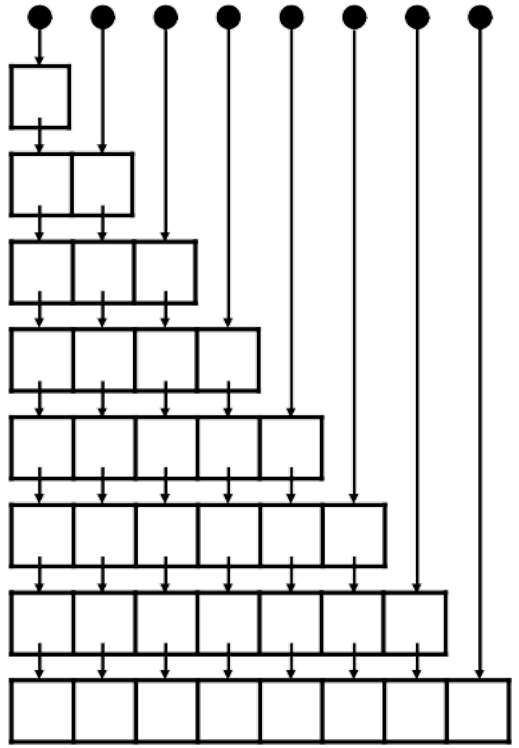
- Udfordring: Kan vi implementere en stak effektivt med et array/arrays?
  - Behøver vi fastsætte en øvre grænse på antallet af elementer?
  - Kan vi komme ned på lineær plads og konstant tid?

Dynamisk array

- Mål:
  - Implementer stak ved brug af arrays i $\Theta(n)$ plads, hvor n er antallet af elementer.
  - Hurtige operationer.
  - Først: fokus på PUSH.
- Løsning 1
  - Start med array af størrelse 1.
  - PUSH(x):
    - opret nyt array af størrelse $+ 1$
    - flyt alle elementer over i det nye array
    - slet gammelt array.

Dynamisk array – løsning 1

- Start med array af størrelse 1.
- PUSH(x):
  - opret nyt array af størrelse + 1
  - flyt alle elementer over i det nye array
  - slet gammelt array.
- Tid: samlet tid for n gange PUSH?
  - Det i'te PUSH tager $\Theta(i)$ tid: Bygge et array af størrelse i og flytte $i-1$ elementer.
  - Samlet tid: $1+2+3+4+5+\ldots+n = \Theta(n^2)$.
- Plads: $\Theta(n)$.
- Udfordring: smartere løsning?

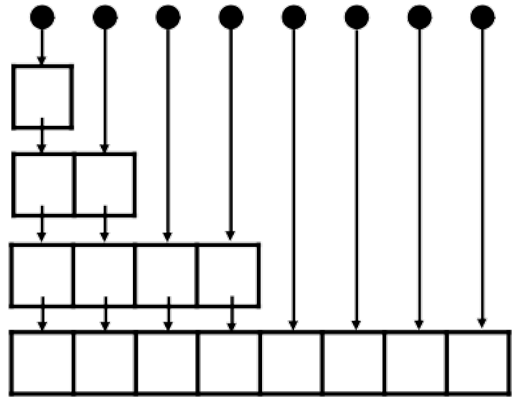


Dynamisk array – løsning 2

- Ide: Kun kopiere en gang imellem.
- Løsning 2.
- start med array af størrelse 1.
- PUSH(x):
  - Hvis arrayet er fyldt op,
    - Opret nyt array i dobbelt størrelse.
    - Flyt elementer til det nye array.
    - Slet gammelt array.
  - Tilføj x til arrayet (dette er der plads til).

Dynamisk array – løsning 2

- Start med array af størrelse 1.
- PUSH(x):
  - Opret nyt array i dobbelt størrelse.
  - Flyt elementer til det nye array.
  - Slet gammelt array.
- Tid: samlet tid for n gange PUSH?
  - Det 2^j'te PUSH tager $\Theta(2^j)$ tid: Bygge et array af størrelse 2^{j+1} og flytte 2^j elementer.
  - Alle andre PUSH: $\Theta(1)$ tid.
  - Samlet tid:
    $1 + 2 + 4 + 8 + \ldots + 2^{\log_2 n} + n = \Theta(n)$.
- Plads: $\Theta(n)$.



Dynamiske arrays

- Stak med dynamisk array.
  - n PUSH operationer i $\Theta(n)$ tid og plads.
  - Kan udviddes til n PUSH, POP, ISEMPTY-operationer i $\Theta(n)$ tid.
  - Køretiden er amortiseret $\Theta(1)$ per operation. Det betyder, at selv om *en* operation kan tage lang tid, vil enhver sekvens af k operationer tage $\Theta(k)$ samlet tid.
  - Med snedige tricks kan løsningen deamortiseres til $\Theta(1)$ værstefaldskøretid per operation.
- Kø med dynamisk array.
  - Samme resultater som for stak.
- Global genobygning.
  - Dynamisk tabel er eksempel på global genopbygning (global rebuilding).
  - General teknik til at gøre statiske datastrukturer dynamiske.

Stakke og kører

Datastruktur	PUSH	POP	ISEMPTY	Space
Array med kapacitet N	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(N)$
Hægtet liste	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
Dynamisk array med simpel udvidelse	$\Theta(n)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)$	$\Theta(n)$
Dynamisk array med fordobling	$\Theta(1)^\dagger$	$\Theta(1)^\dagger$	$\Theta(1)$	$\Theta(n)$

†: amortiseret.

Introduktion til datastrukturer

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