Øvingsforelesning 3

TDT4120 - Algoritmer og datastrukturer

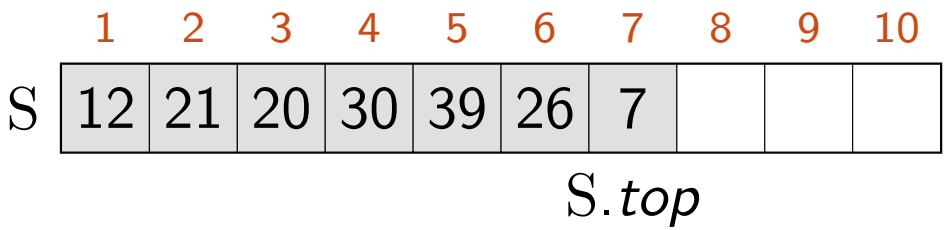
Øving 2

Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$

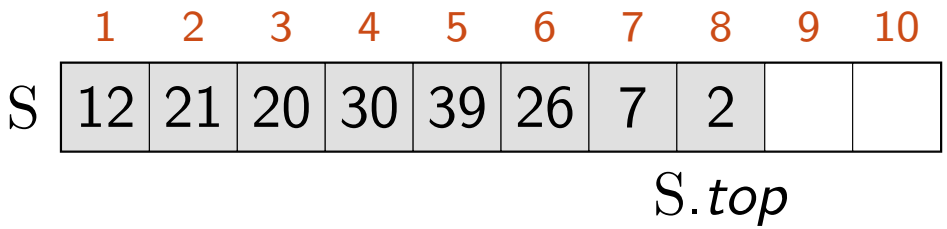
Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$



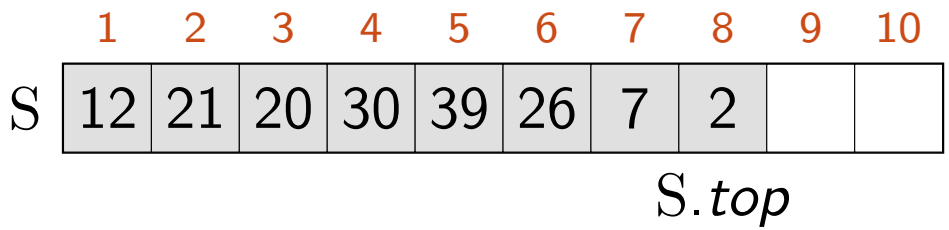
Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$



Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$



$\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7, 2 \rangle$

Stakker

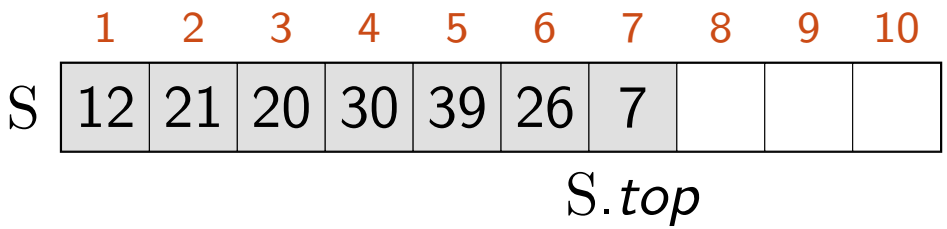
Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$

Oppgave 3: $\langle 12, 21, 20, 30, 39, 26, 7 \rangle \rightarrow \langle 12, 21, 8, 41, 39, 26 \rangle$

Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$

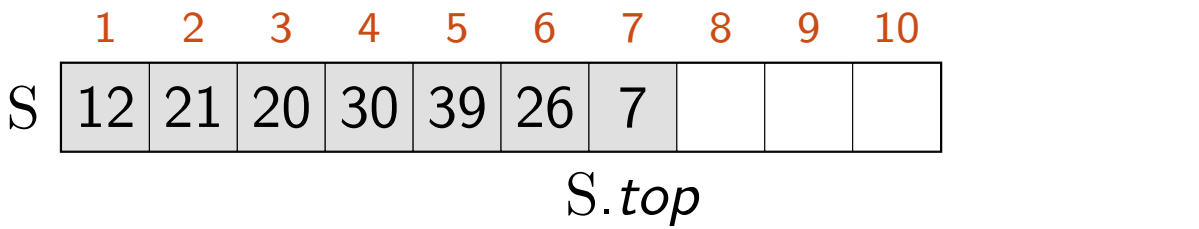
Oppgave 3: $\langle 12, 21, 20, 30, 39, 26, 7 \rangle \rightarrow \langle 12, 21, 8, 41, 39, 26 \rangle$

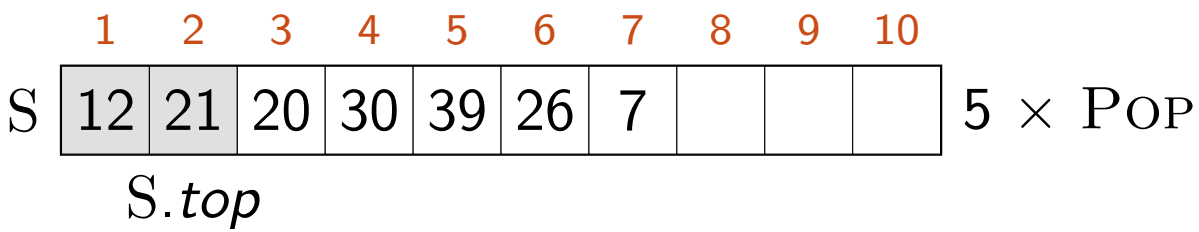


Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$

Oppgave 3: $\langle 12, 21, 20, 30, 39, 26, 7 \rangle \rightarrow \langle 12, 21, 8, 41, 39, 26 \rangle$



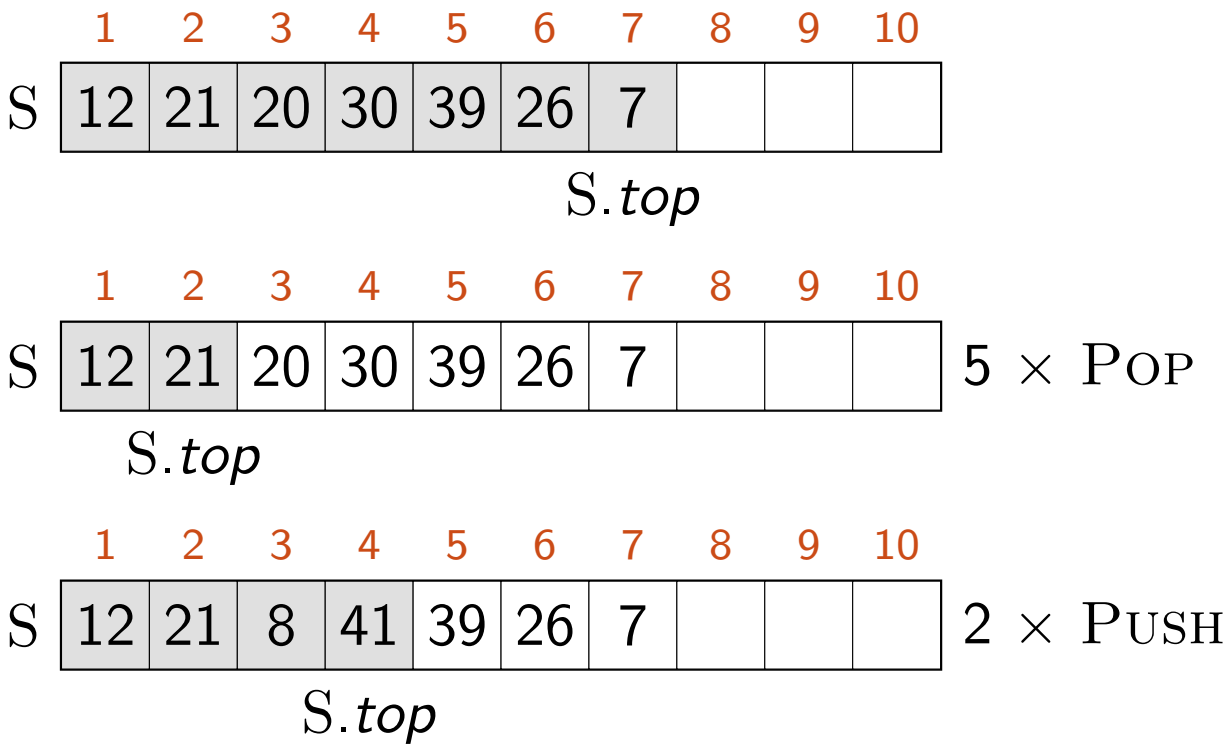


5 × Pop

Stakker

Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$

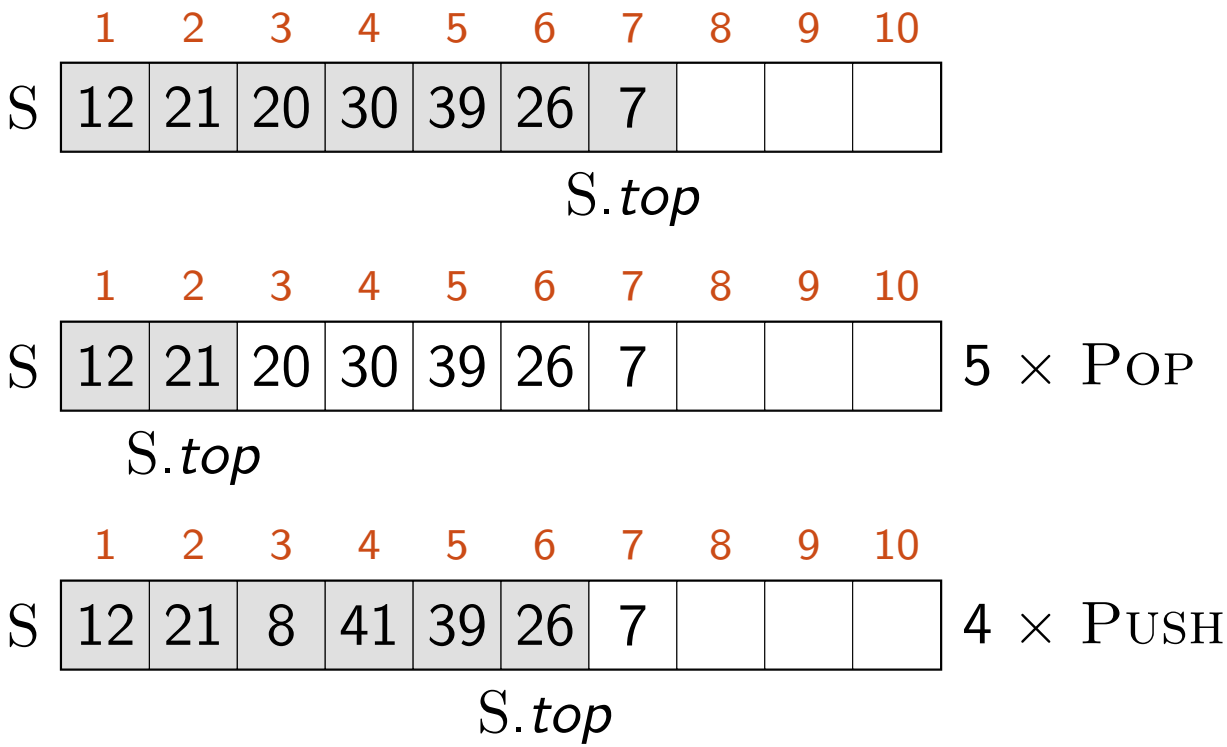
Oppgave 3: $\langle 12, 21, 20, 30, 39, 26, 7 \rangle \rightarrow \langle 12, 21, 8, 41, 39, 26 \rangle$



Stakker

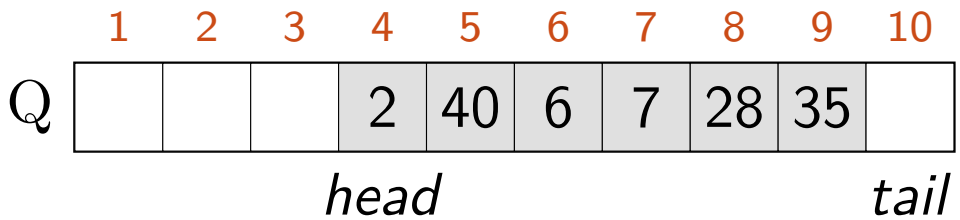
Oppgave 2: $\mathrm{S} = \langle 12, 21, 20, 30, 39, 26, 7 \rangle$, $\textsc{Push}(\mathrm{S}, 2)$

Oppgave 3: $\langle 12, 21, 20, 30, 39, 26, 7 \rangle \rightarrow \langle 12, 21, 8, 41, 39, 26 \rangle$

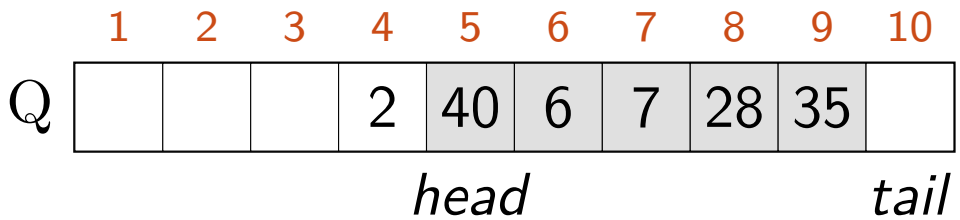


Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$

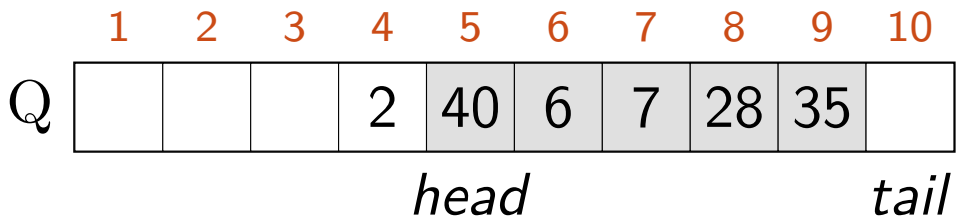
Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$



Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$



Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$



$\mathrm{Q} = \langle 35, 28, 7, 6, 40 \rangle$

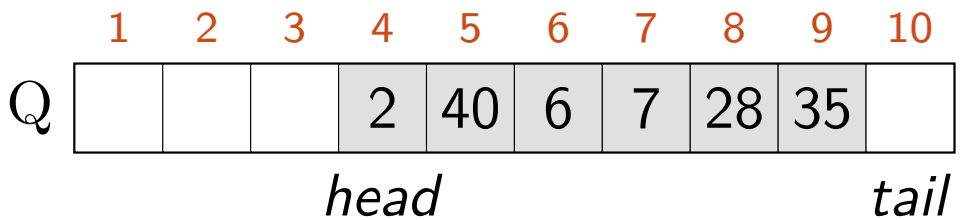
Køer

Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$

Oppgave 5: $\langle 35, 28, 7, 6, 40, 2 \rangle \rightarrow \langle 27, 8, 28, 6, 4, 2 \rangle$

Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$

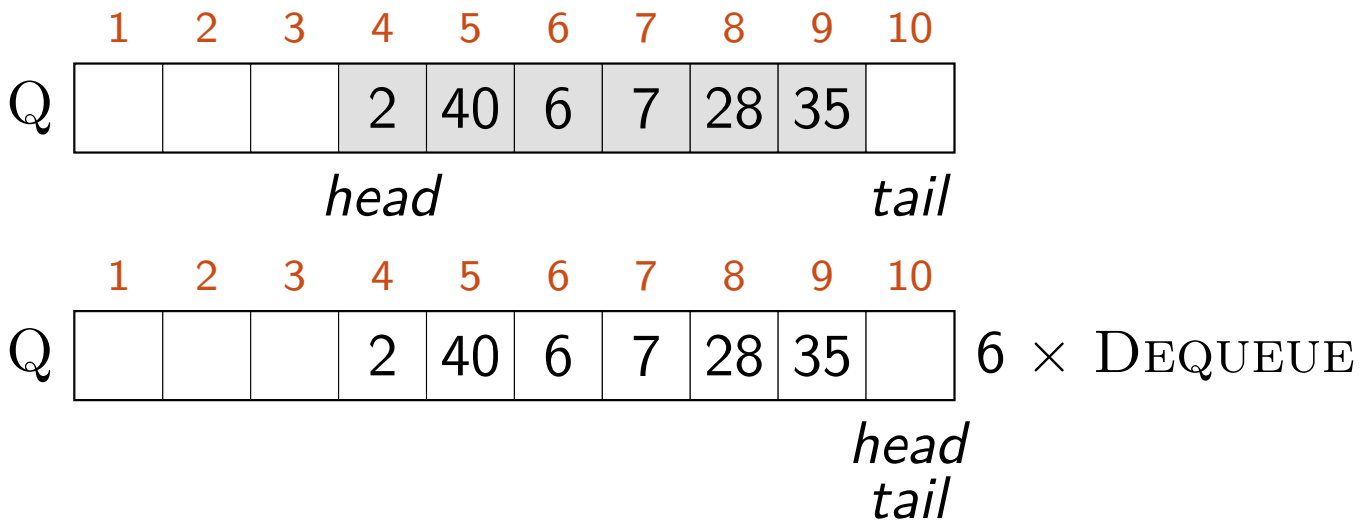
Oppgave 5: $\langle 35, 28, 7, 6, 40, 2 \rangle \rightarrow \langle 27, 8, 28, 6, 4, 2 \rangle$



Køer

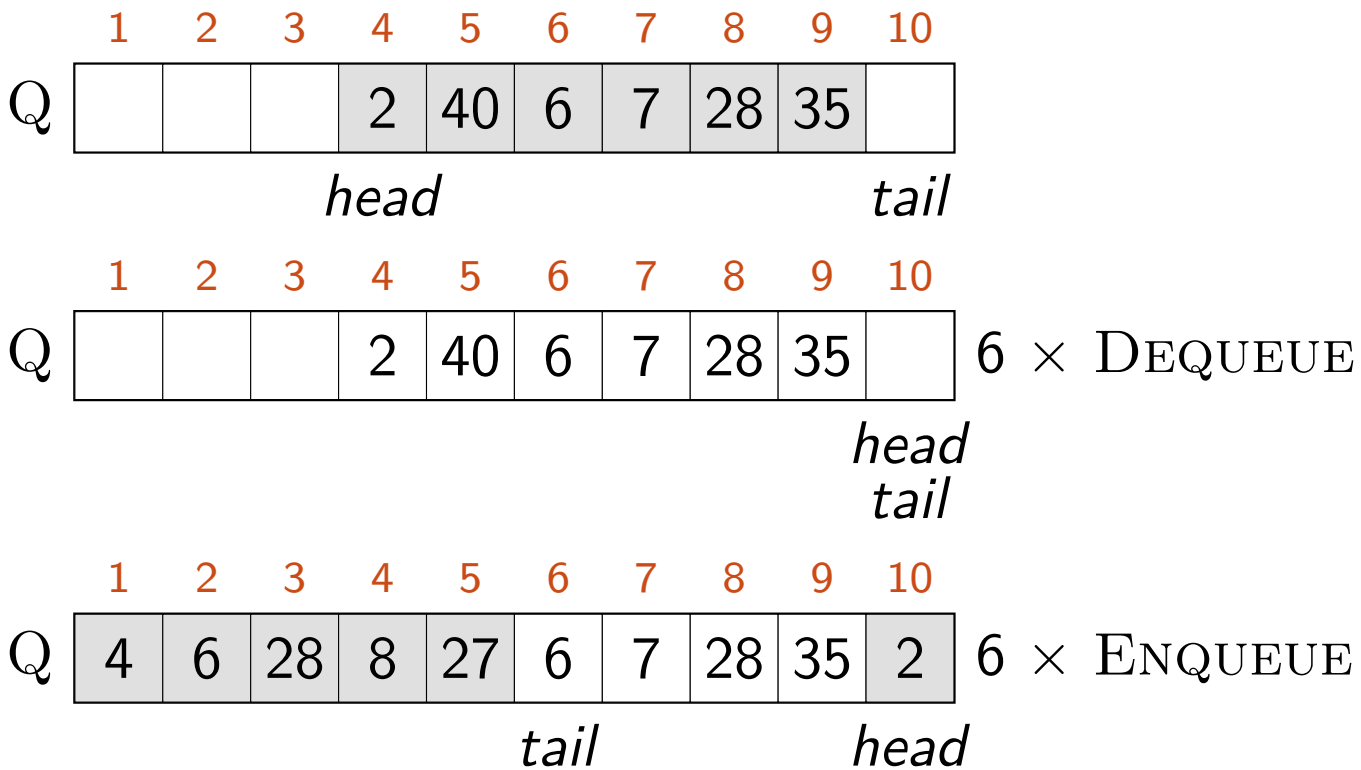
Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$

Oppgave 5: $\langle 35, 28, 7, 6, 40, 2 \rangle \rightarrow \langle 27, 8, 28, 6, 4, 2 \rangle$



Oppgave 4: $\mathrm{Q} = \langle 35, 28, 7, 6, 40, 2 \rangle$, $\textsc{Dequeue}(\mathrm{Q})$

Oppgave 5: $\langle 35, 28, 7, 6, 40, 2 \rangle \rightarrow \langle 27, 8, 28, 6, 4, 2 \rangle$

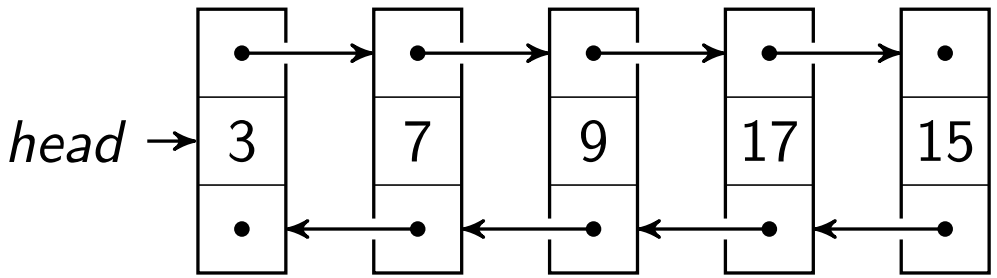


Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.

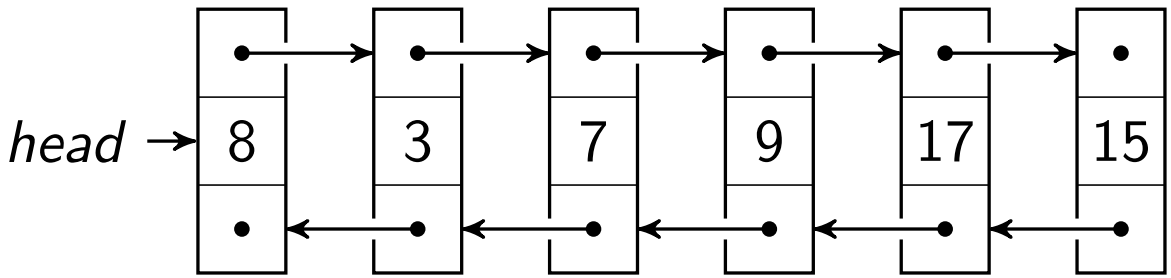
Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.



Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.



Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.

$\mathrm{L} = \langle 8, 3, 7, 9, 17, 15 \rangle$

Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.

Oppgave 8: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Search}(\mathrm{L}, 8)$

Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.

Oppgave 8: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Search}(\mathrm{L}, 8)$

$\textsc{List-Search}$ endrer ikke listen.

Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.

Oppgave 8: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Search}(\mathrm{L}, 8)$

Oppgave 9: Tidskompleksitet for sletting i lenkede lister

Lenket liste

Oppgave 7: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Insert}(\mathrm{L}, x)$, $x.key = 8$.

Oppgave 8: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, $\textsc{List-Search}(\mathrm{L}, 8)$

Oppgave 9: Tidskompleksitet for sletting i lenkede lister

- Enkelt-lenket - $\mathrm{O}(n)$ - Må søke igjennom listen etter elementet
- Dobbelt-lenket - $\mathrm{O}(1)$ - Kjenner både elementet før og etter

Oppgave 10: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, hvilke representasjoner stemmer?

Pekere og lenket liste

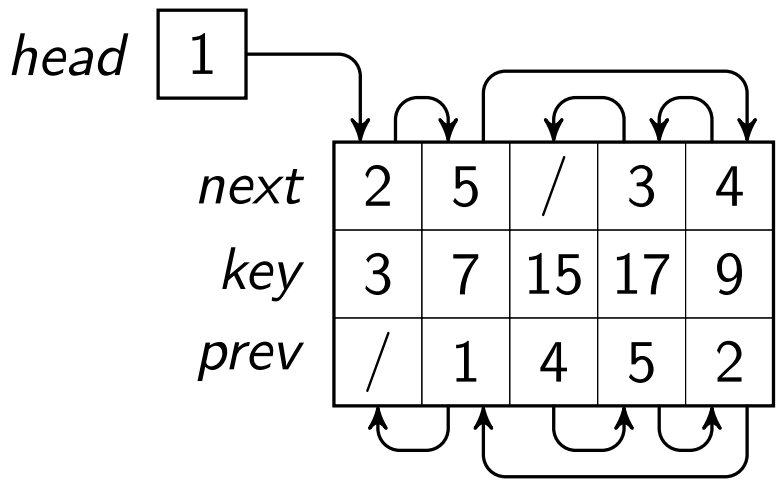
Oppgave 10: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, hvilke representasjoner stemmer?

$\mathit{next} = \langle 2, 5, /, 3, 4 \rangle$, $\mathit{key} = \langle 3, 7, 15, 17, 9 \rangle$, $\mathit{prev} = \langle /, 1, 4, 5, 2 \rangle$
$\mathit{head} = 1$.

Pekere og lenket liste

Oppgave 10: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, hvilke representasjoner stemmer?

$\mathit{next} = \langle 2, 5, /, 3, 4 \rangle$, $\mathit{key} = \langle 3, 7, 15, 17, 9 \rangle$, $\mathit{prev} = \langle /, 1, 4, 5, 2 \rangle$
$\mathit{head} = 1$.



Pekere og lenket liste

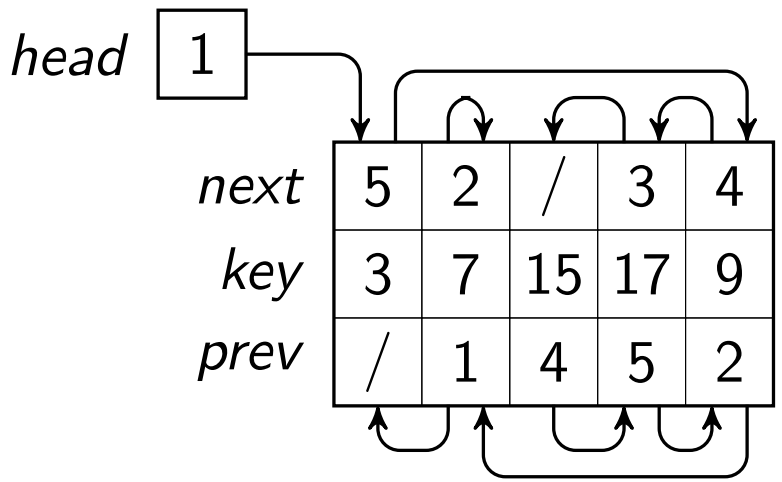
Oppgave 10: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, hvilke representasjoner stemmer?

$\mathit{next} = \langle \mathbf{5}, \mathbf{2}, /, 3, 4 \rangle$, $\mathit{key} = \langle 3, 7, 15, 17, 9 \rangle$, $\mathit{prev} = \langle /, 1, 4, 5, 2 \rangle$
$\mathit{head} = 1$.

Pekere og lenket liste

Oppgave 10: $\mathrm{L} = \langle 3, 7, 9, 17, 15 \rangle$, hvilke representasjoner stemmer?

$\mathit{next} = \langle \mathbf{5}, \mathbf{2}, /, 3, 4 \rangle$, $\mathit{key} = \langle 3, 7, 15, 17, 9 \rangle$, $\mathit{prev} = \langle /, 1, 4, 5, 2 \rangle$
$\mathit{head} = 1$.



Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og hente ut sist lagt til

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og hente ut sist lagt til
- Hashtabell - **Nei** - Holder ikke styr på rekkefølgen

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og hente ut sist lagt til
- Hashtabell - **Nei** - Holder ikke styr på rekkefølgen
- Kø - **Nei** - Kan ikke hente ut siste uten å tømme hele køen
- Stakk - **Ja** - $\mathrm{O}(1)$ for begge

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og hente ut sist lagt til
- Hashtabell - **Nei** - Holder ikke styr på rekkefølgen
- Kø - **Nei** - Kan ikke hente ut siste uten å tømme hele køen
- Stakk - **Ja** - $\mathrm{O}(1)$ for begge

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

Oppgave 12: Datastruktur for strøm mellom programmer

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

Oppgave 12: Datastruktur for strøm mellom programmer

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og fjerne og med kan legge til i en ende og ta fra den andre

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

Oppgave 12: Datastruktur for strøm mellom programmer

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og fjerne og med kan legge til i en ende og ta fra den andre
- Hashtabell - **Nei** - Holder ikke styr på rekkefølgen

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

Oppgave 12: Datastruktur for strøm mellom programmer

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og fjerne og med kan legge til i en ende og ta fra den andre
- Hashtabell - **Nei** - Holder ikke styr på rekkefølgen
- Kø - **Ja** - $\mathrm{O}(1)$ for begge, kan skrives og leses fra samtidig

Valg av datastruktur

Oppgave 11: Datastruktur for angrefunksjonalitet

Oppgave 12: Datastruktur for strøm mellom programmer

- Lenket liste - **Ja** - $\mathrm{O}(1)$ for å legge til og fjerne og med kan legge til i en ende og ta fra den andre
- Hashtabell - **Nei** - Holder ikke styr på rekkefølgen
- Kø - **Ja** - $\mathrm{O}(1)$ for begge, kan skrives og leses fra samtidig
- Stakk - **Nei** - Kan ikke hent ut neste element uten å tømme hele stakken

Hashfunksjoner - Teori

Oppgave 13: $x.key = m$, $h(m) = j$, h er en hashfunksjon

Hashfunksjoner - Teori

Oppgave 13: $x.key = m$, $h(m) = j$, h er en hashfunksjon

- x er elementet

Hashfunksjoner - Teori

Oppgave 13: $x.key = m$, $h(m) = j$, h er en hashfunksjon

- x er elementet
- m er nøkkelen

Hashfunksjoner - Teori

Oppgave 13: $x.key = m$, $h(m) = j$, h er en hashfunksjon

- x er elementet
- m er nøkkelen
- j er hashen

Hashfunksjoner - Teori

Oppgave 13: $x.key = m$, $h(m) = j$, h er en hashfunksjon

Oppgave 14: Kollisjoner i hashtabeller

Hashfunksjoner - Teori

Oppgave 13: $x.key = m$, $h(m) = j$, h er en hashfunksjon

Oppgave 14: Kollisjoner i hashtabeller

To eller flere ulike nøkler gir samme hashverdi.

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Fordeler nøklene omtrentlig uniformt over hashtabellen.

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Oppgave 16: Hvilke av definisjonene er hashfunksjoner?

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Oppgave 16: Hvilke av definisjonene er hashfunksjoner?

En hashfunksjon er en funksjon $\mathrm{U} \to \{0, 1, \ldots, m-1\}$

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Oppgave 16: Hvilke av definisjonene er hashfunksjoner?

En hashfunksjon er en funksjon $\mathrm{U} \to \{0, 1, \ldots, m-1\}$
$h(k)$ må alltid ta samme verdi for samme nøkkel.

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Oppgave 16: Hvilke av definisjonene er hashfunksjoner?

Oppgave 17: God hashfunksjon for lagring av fødselsnummer i en hashtabell med 65536 plasser.

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Oppgave 16: Hvilke av definisjonene er hashfunksjoner?

Oppgave 17: God hashfunksjon for lagring av fødselsnummer i en hashtabell med 65536 plasser.

Fødselsnummer er ikke uniformt fordelt.

Gode hashfunksjoner

Oppgave 15: Hva er en god hashfunksjon?

Oppgave 16: Hvilke av definisjonene er hashfunksjoner?

Oppgave 17: God hashfunksjon for lagring av fødselsnummer i en hashtabell med 65536 plasser.

Fødselsnummer er ikke uniformt fordelt.

- **Divisjonsmetoden:** De siste 16-bitsene (4-5 siste sifferene)
- **Multiplikasjonsmetoden:** En bedre fordeling av verdiene

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Kjøretiden i verste tilfellet endrer seg ofte basert på inputet.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Hvis $n \neq \frac{m}{3}$ kan vi utføre vanlig innsetting.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Hvis $n \neq \frac{m}{3}$ kan vi utføre vanlig innsetting.

Hvis $n = \frac{m}{3}$ må vi lage en ny hashtabell og kopier over verdiene.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Hvis $n \neq \frac{m}{3}$ kan vi utføre vanlig innsetting.

Hvis $n = \frac{m}{3}$ må vi lage en ny hashtabell og kopier over verdiene.

$$c_i = \begin{cases} 4i + 1 & \text{hvis } i \text{ kan skrives på formen } 3^k \\ 1 & \text{ellers} \end{cases}$$

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

$$c_i = \begin{cases} 4i + 1 & \text{hvis } i \text{ kan skrives på formen } 3^k \\ 1 & \text{ellers} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leqslant n + \sum_{j=0}^{\lfloor \log_3 n \rfloor} 4 \cdot 3^k$$

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

$$c_i = \begin{cases} 4i + 1 & \text{hvis } i \text{ kan skrives på formen } 3^k \\ 1 & \text{ellers} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leqslant n + \sum_{j=0}^{\lfloor \log_3 n \rfloor} 4 \cdot 3^k \leqslant n + 4 \cdot 3n$$

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

$$c_i = \begin{cases} 4i + 1 & \text{hvis } i \text{ kan skrives på formen } 3^k \\ 1 & \text{ellers} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leqslant n + \sum_{j=0}^{\lfloor \log_3 n \rfloor} 4 \cdot 3^k \leqslant n + 4 \cdot 3n = 13n$$

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

$$c_i = \begin{cases} 4i + 1 & \text{hvis } i \text{ kan skrives på formen } 3^k \\ 1 & \text{ellers} \end{cases}$$

$$\sum_{i=1}^{n} c_i \leqslant n + \sum_{j=0}^{\lfloor \log_3 n \rfloor} 4 \cdot 3^k \leqslant n + 4 \cdot 3n = 13n$$

Amortisert kjøretid på $O(1)$.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Oppgave 20: Amortisert kjøretid for innsetting i en sortert lenket liste.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Oppgave 20: Amortisert kjøretid for innsetting i en sortert lenket liste.

I verste tilfellet må vi for hver insetting iterere til slutten av listen.

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Oppgave 20: Amortisert kjøretid for innsetting i en sortert lenket liste.

I verste tilfellet må vi for hver insetting iterere til slutten av listen.

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} i$$

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Oppgave 20: Amortisert kjøretid for innsetting i en sortert lenket liste.

I verste tilfellet må vi for hver insetting iterere til slutten av listen.

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} i = \frac{n^2}{2} + \frac{n}{2}$$

Amortisert analyse

Oppgave 18: Hvorfor er amortisert analyse ofte bedre enn kjøretid i verste tilfellet?

Oppgave 19: Amortisert kjøretid for innsetting i dynamisk hashtabell med utvidelse når lastfaktoren er $\frac{1}{3}$.

Oppgave 20: Amortisert kjøretid for innsetting i en sortert lenket liste.

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} i = \frac{n^2}{2} + \frac{n}{2}$$

Amortisert kjøretid på $O(n)$.

Labyrintutforskning

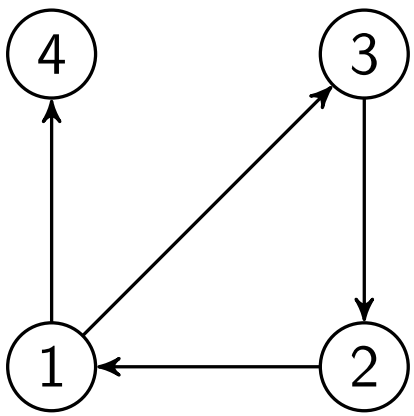
```
CAN-ESCAPE(M, ℓ)
 1  let S be a new stack
 2  PUSH(S, ℓ)
 3  while not STACK-EMPTY(S)
 4        ℓ = POP(S)
 5        ℓ_new = EXPLORE(M, ℓ)
 6        if IS-GOAL(ℓ_new)
 7              return TRUE
 8        for direction in POSSIBLE-DIRECTIONS(M, ℓ_new)
 9              let ℓ_copy be a new location
10              ℓ_copy.x = ℓ_new.x
11              ℓ_copy.y = ℓ_new.y
12              ℓ_copy.direction = direction
13              PUSH(S, ℓ_copy)
14  return FALSE
```

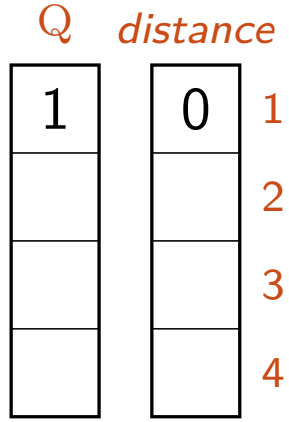
Minimalt antall hopp

- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, *a*, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, *b*
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg *b* til i køen

Minimalt antall hopp

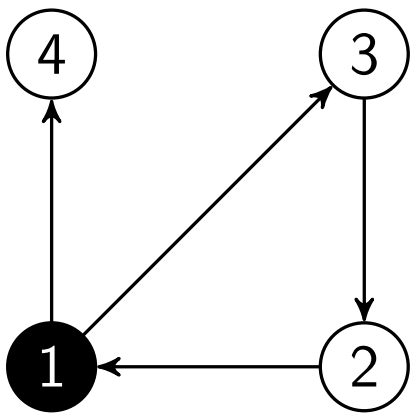
- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen

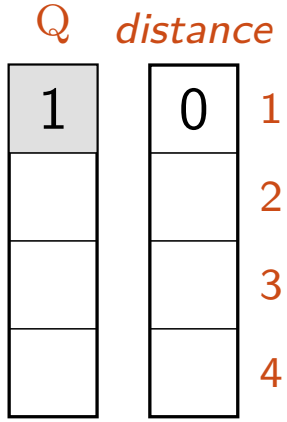




Minimalt antall hopp

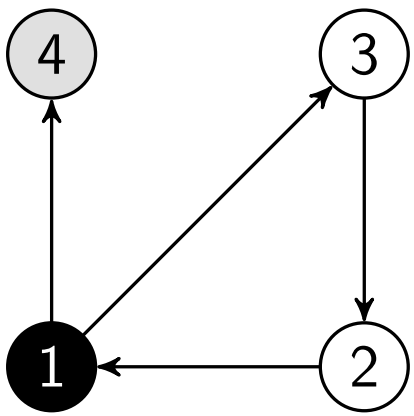
- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen

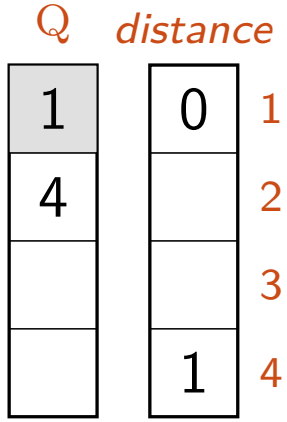




Minimalt antall hopp

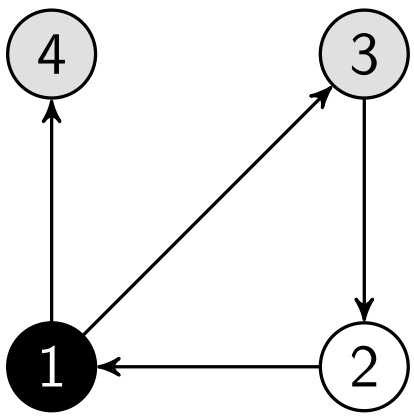
- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen





Minimalt antall hopp

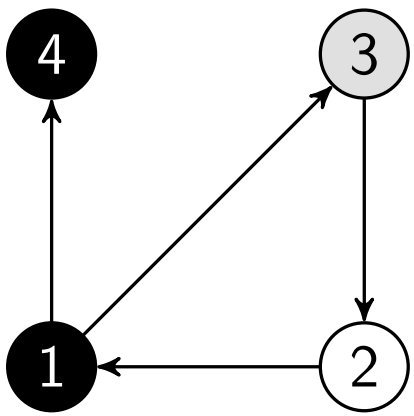
- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen



Q	*distance*	
1	0	1
4		2
3	1	3
	1	4

Minimalt antall hopp

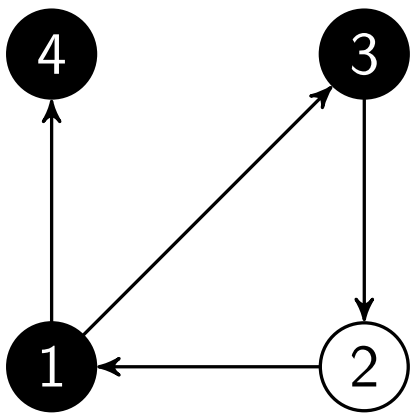
- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen



Q	*distance*	
1	0	1
4		2
3	1	3
	1	4

Minimalt antall hopp

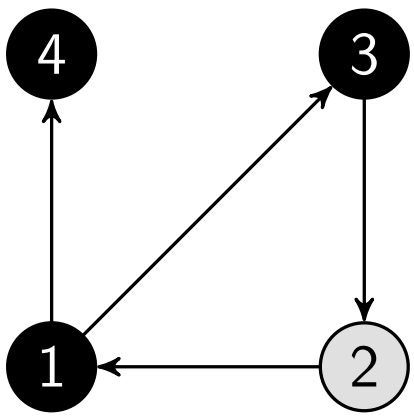
- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen



Q	*distance*	
1	0	1
4		2
3	1	3
	1	4

Minimalt antall hopp

- Besøker nodene i tur og orden.
- Starter med node $\mathrm{A}[j]$ i køen.
- For hver node, a, som besøkes
  - Fargelegg noden sort
  - For hver referanse til en hvit node, b
    - Hvis dette er $\mathrm{A}[k]$ er vi ferdig
    - Fargelegg noden grå
    - Sett $b.distance = a.distance + 1$
    - Legg b til i køen



Q	*distance*	
1	0	1
4	2	2
3	1	3
2	1	4

Poengtavlen - Oppgave 6

```
class Queue:

    def __init__(self, max_size):
        self.head = 0
        self.tail = 0
        self.array = [0] * max_size
        self.max_size = max_size

    def enqueue(self, value):
        self.array[self.tail] = value
        self.tail = (self.tail + 1) % self.max_size

    def dequeue(self):
        value = self.array[self.head]
        self.head = (self.head + 1) % self.max_size
        return value
```

Poengtavlen - Oppgave 6

Kan bruke standard listemetoder for å få en bedre poengsum.

```
class Queue:

    def __init__(self, max_size):
        self.array = []

    def enqueue(self, value):
        self.array.append(value)

    def dequeue(self):
        return self.array.pop(0)
```

Merk: $\mathrm{O}(n)$ for DEQUEUE

Poengtavlen - Oppgave 6

Kan optimalisere ved å fjerne lagring i objektet.

```
class Queue:

    def __init__(self, max_size):
        array = []
        self.enqueue = array.append
        self.dequeue = lambda: array.pop(0)

    def enqueue(self, value):
        pass

    def dequeue(self):
        pass
```

Merk: $\mathrm{O}(n)$ for DEQUEUE

Kan videre bruke functools for å gjøre dequeue raskere

Poengtavlen - Oppgave 6

Grunnet for store minnegrenser har man en veldig rask løsning.

```
class Queue:

    def __init__(self, max_size):
        array = []
        self.enqueue = array.append
        self.dequeue = array.__iter__().__next__

    def enqueue(self, value):
        pass

    def dequeue(self):
        pass
```

Merk: Fungerer ikke til vanlig, da man bruker mer og mer minne

Poengtavlen - Oppgave 22

Noen mulige måter:

- Sortering ved innsetting (*insertion sort*) - $\mathrm{O}(n^2)$ gjennomsnitt.
- Sortering ved utvelgelse (*selection sort*) - $\mathrm{O}(n^2)$ gjennomsnitt.
- Flettesortering (*merge sort*) - $\mathrm{O}(n \lg n)$ i alle tilfeller.
- Quicksort - $\mathrm{O}(n \lg n)$ gjennomsnitt.
- Radikssortering - $\mathrm{O}(nk)$ i alle tilfeller ($k = \lg x$, $x = \max(\mathrm{A})$).
- ...

Poengtavlen - Oppgave 22

Raskeste studentkode

```
def sort(stack1, stack2, stack3):
    radix = 0
    minval = +1000000 # Must have LSB = 0
    maxval = -1000000
    while radix < int.bit_length(maxval-minval):
        mask = 1 << radix
        while not stack1.empty():
            value = stack1.pop()
            minval = min(minval, value) & ~1 # Set LSB to 0
            maxval = max(maxval, value)

            if (value-minval) & mask:
                stack2.push(value)
            else:
                stack3.push(value)

        while not stack2.empty():
            stack1.push(stack2.pop())
        while not stack3.empty():
            stack1.push(stack3.pop())
        radix += 1
```