

# MIU EFTERÅR 2021

# ALGORITMER OG DATASTRUKTURER

Heldagsseminar 1

TROELS BJERRE LUND



AARHUS  
UNIVERSITY

9/50

**9) Hvilken af nedenstående dækker bedst beskrivelsen af, hvad en algoritme er?**

- En form for seksuel aktivitet
- En serie af beregningsmodeller
- En metode til måling af smerte
- En kemisk betegnelse
- En form for kirkemaleri

## Planen for i dag

---

- 09:00-10:00: Introduktion og velkomst
- 10:00-12:00: Forelæsning afbrudt af en kaffepause
- 12:00-12:45: Frokost
- 12:45-14:00: Værktøj og praktiske øvelser
- 14:00-14:15: Pause
- 14:15-16:00: Teoretiske øvelser

# Undervisere på kurset

---

- Forelæser:  
Troels Bjerre Lund ( mig)  
Lektor i Datalog  
IT-Universitetet i København



- Hjælpelærer:  
Steffan Christ Sølvsten  
PhD studerende i Datalogi  
Aarhus Universitet



# Kursets læringsmål

---

- Formulere algoritmer og datastrukturer i form af pseudokode
- Analysere og sammenligne tid og pladsforbrug af algoritmer og datastrukturer
- Konstruere og implementere simple algoritmer ved hjælp af standard algoritme paradigmer
- Identificere og sammenligne datastrukturer til realisering af simple data abstraktioner
- Konstruere, implementere og evaluere datastrukturer og algoritmers ydeevne for simple algoritmiske problemer
- Beskrive hvordan algoritmer anvendes i virkelige systemer.
- Beskrive forskellige succeskriterier for algoritmer inklusive effektivitet, korrekthed, fairness, transparens og accountabilitet

## Faglige forudsætninger

---

### Formelle forudsætninger:

- Introduktion til Programmering eller tilsvarende
- Matematisk grundviden

Det her er et teori-kursus, men...

- Hvis du ikke er fortrolig med programmering, så kommer det til at føles som et programmeringskursus

Ligegyldigt hvor fortrolig du er med programmering:

- Øvelse gør mester: Kattis mf. (stay tuned)

# Kursusbogen og materialer

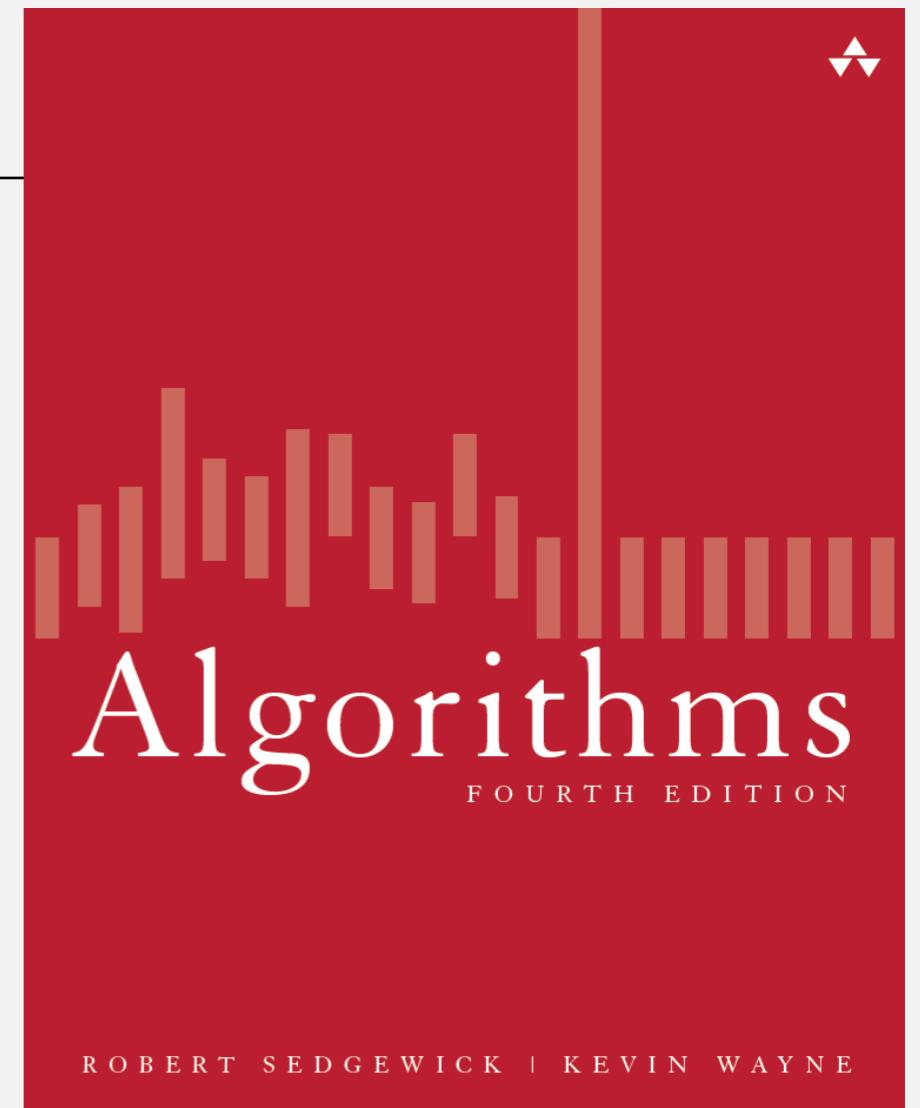
---

Bogens hjemmeside:

- <https://algs4.cs.princeton.edu/>

Links til:

- kodebiblioteker (algs4.jar)
- kildekode til alle programmer i bogen
- programmeringsopgaver (+autograding)
- forelæsninger over bogen af bogens forfattere



# Forløbig plan for emner

---

- 1. seminar (Fredag 10/9): Union-Find: S&W 1.5
- 2. seminar (Tirsdag 14/9): Stacks and Queues: S&W 1.3
- 3. seminar (Tirsdag 21/9): Analysis of Algorithms: S&W 1.4
- 4. seminar (Tirsdag 28/9): Priority Queues: S&W 2.4
- 5. seminar (Fredag 15/10): Sorting, Mergesort: S&W 2.1, 2.2
- 6. seminar (Tirsdag 19/10): Symbol Tables: S&W 3.1, 3.4
- 7. seminar (Tirsdag 26/10): Graphs, DFS, BFS: S&W 4.1, 4.2
- 8. seminar (Tirsdag 9/11): MST, Dijkstra: S&W 4.3, 4.4
- 9. seminar (Fredag 12/11): Strings, Radix Sort, Quicksort: S&W 5.1, 2.3
- 10. seminar (Tirsdag 16/11): Search Trees: S&W 3.2, 3.3, 5.2
- 11. seminar (Tirsdag 23/11): KMP: S&W 5.3
- 12. seminar (Fredag 3/12): Alternative successkriterier: noter og artikler

## En typisk (online) uge

---

- Jeg uploader ugens forelæsning ca. en uge i forvejen
- I ser forelæsningen, læser kapitlet, kigger på øvelserne
- I stiller spørgsmål løbende på Discord
- Vi mødes tirsdag klokken 15 på Discord:
  - Vi snakker kort i plenum om emnet for ugen
  - I går ud i virtuelle grupperum i små grupper og:
    - Diskuterer løsninger
    - Arbejder på de opgaver I ikke kunne
    - Hiver Steffan eller mig ind i jeres grupperum, når I skal bruge hjælp
  - Husk webcam og mikrofon

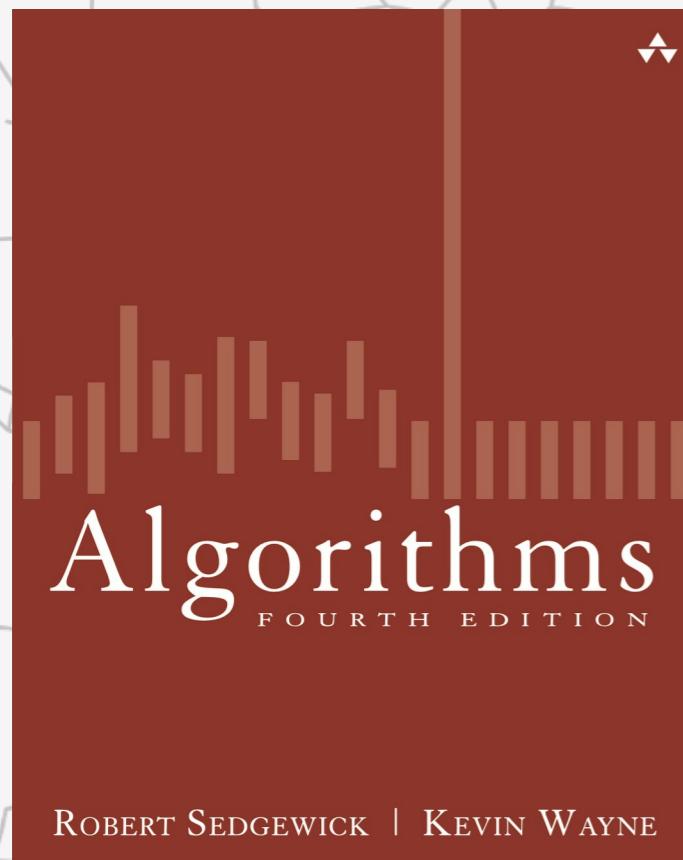
## Obligatoriske afleveringer

---

- 3 obligatoriske afleveringer
- Afleveringsfrist: mandagen efter hver af heldagsseminarerne
- Stilles 14 dage før afleveringsfristen
- 2 opgaver skal godkendes for at kunne gå til eksamen
- Opgaverne tæller med i eksamensresultatet

Fredag d. 17/12

- 20 minutters forberedelse
- 20 minutters mundtlig eksamen



## 1.5 FORÉN-OG-FIND

---

- ▶ *dynamisk sammenhæng*
- ▶ *hurtig find*
- ▶ *hurtig forening*
- ▶ *forbedringer*
- ▶ *anvendelser*

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

# Formål med dagens forelæsning (og kurset som helhed)

---

## Arbejdsgang for at udvikle en brugbar algoritme.

- Modellér problemet.
- Find på en algoritm, der løser det.
- Er den hurtig nok? Bruger den for meget hukommelse?
- Hvis ikke, find ud af hvorfor.
- Find løsninger, der håndterer årsagen.
- Gentag, indtil du er tilfreds.

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 1.5 FORÉN-OG-FIND

---

- ▶ *dynamisk sammenhæng*
- ▶ *hurtig find*
- ▶ *hurtig forening*
- ▶ *forbedringer*
- ▶ *anvendelser*

# Dynamisk sammenhæng (problembeskrivelse)

---

Givet en mængde af N elementer, understøttet to operationer:

- union(a, b): Forén to elementer.
- connected(a, b): Er to elementer forbundne (transitivt)?

forén 4 og 3

forén 3 og 8

forén 6 og 5

forén 9 og 4

forén 2 og 1

er 0 og 7 forbundne?

✗

er 8 og 9 forbundne?

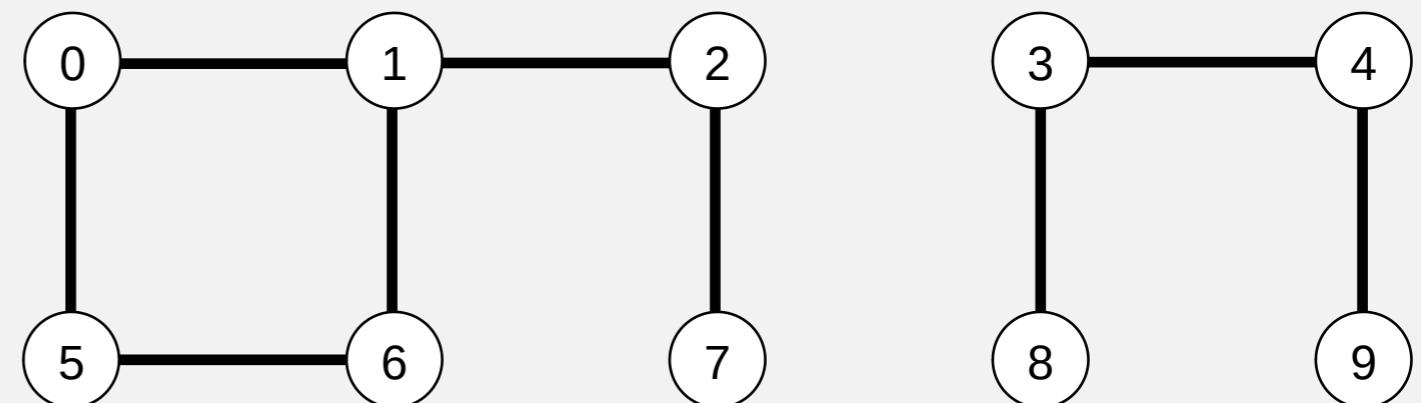
forén 5 og 0

forén 7 og 2

forén 6 og 1

forén 1 og 0

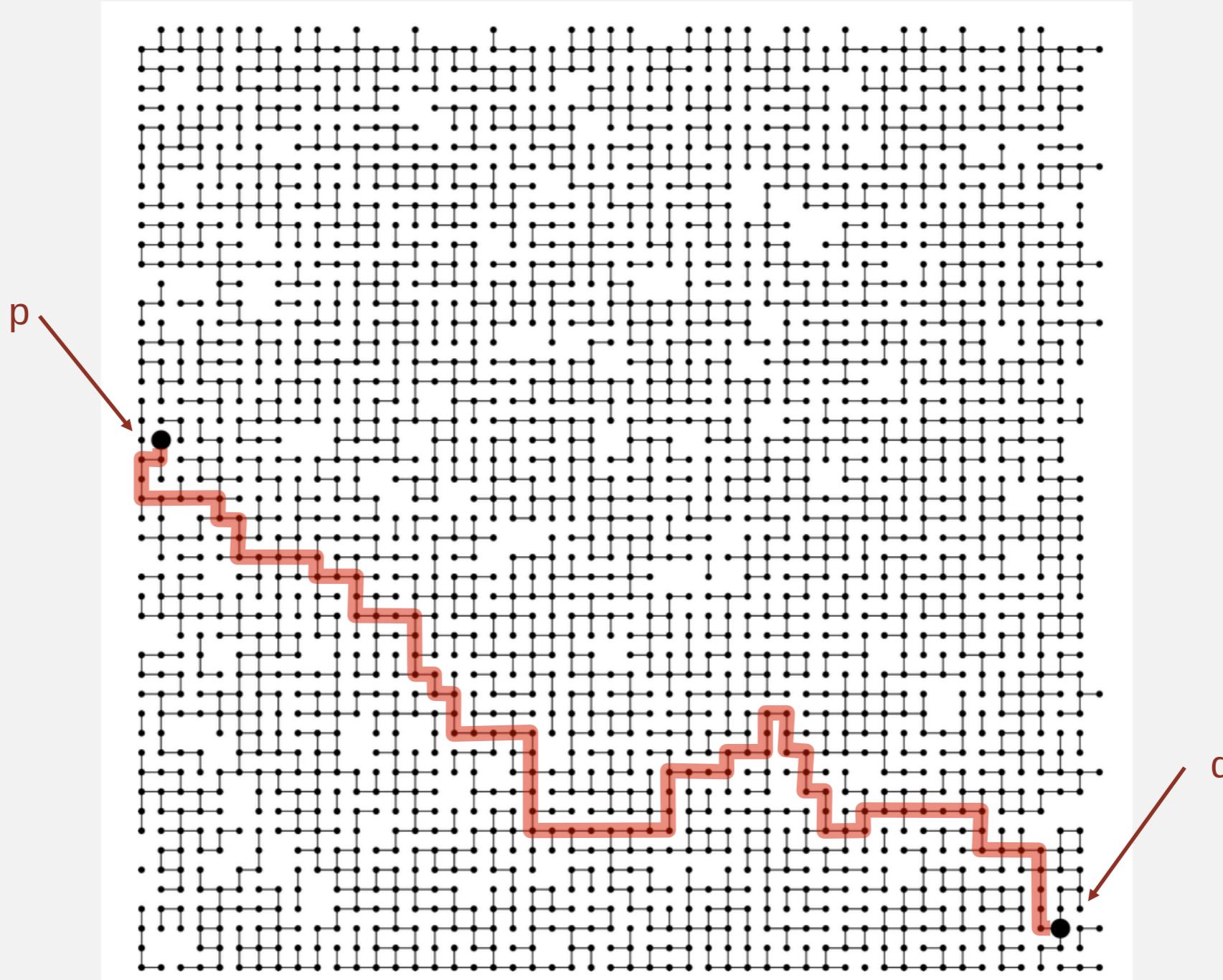
er 0 og 7 forbundne?



## Et større eksempel

---

Q. Er der en sti der forbinder  $p$  og  $q$  ?



A. JA!

# Modellering af elementerne

---

Konkrete anvendelser vil have forskellige typer af elementer.

- Pixler i et digitalt billede.
- Computere i et netværk.
- Venner i et socialt netværk.
- Transistorere på en computer chip.
- Elementer i en matematisk mængde.
- Variabelnavne i et Java program.

Ved programming er det praktisk at navngive elementer 0 til  $N - 1$ .

- Vi kan bruge heltal som array indeks.
- Ignorér detaljer der er irrelevante for kerneproblemets.



Hvis det virker uacceptabelt, så kan vi bruge en symbol tabel til at oversætte fra domænets elementer til heltal. (Kapitel 3)

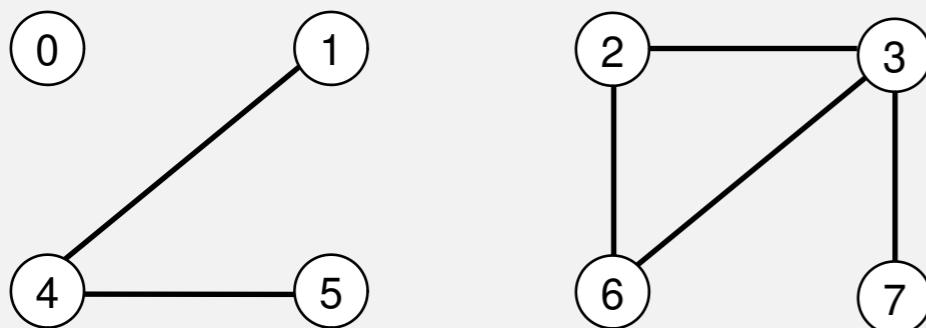
# Modellering af forbindelserne

---

Vi antager at "er forbundet med" er en ækvivalensrelation:

- Refleksiv:  $p$  er forbundet med sig selv  $p$ .
- Symmetrisk: hvis  $p$  is forbundet med  $q$ , så er  $q$  også forbundet med  $p$ .
- Transitiv: hvis  $p$  is forbundet med  $q$  og  $q$  er forbundet med  $r$ ,  
så er  $p$  også forbundet med  $r$ .

Sammenhængskomponent. Maksimal mængde af parvis forbundne elementer.



$$\{ 0 \} \{ 1 4 5 \} \{ 2 3 6 7 \}$$

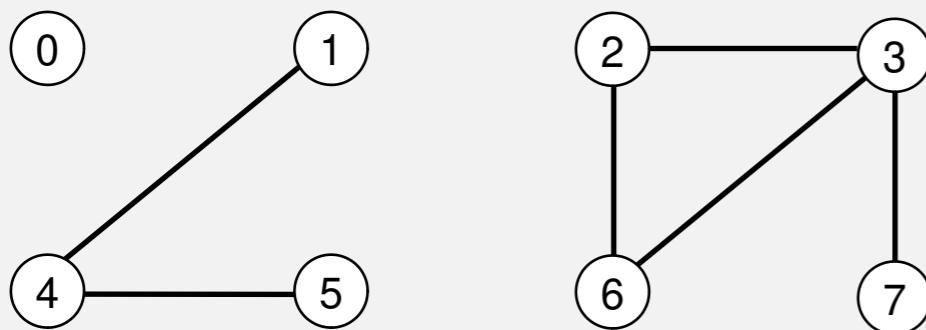
3 sammenhængskomponenter

# Implementering af operationerne

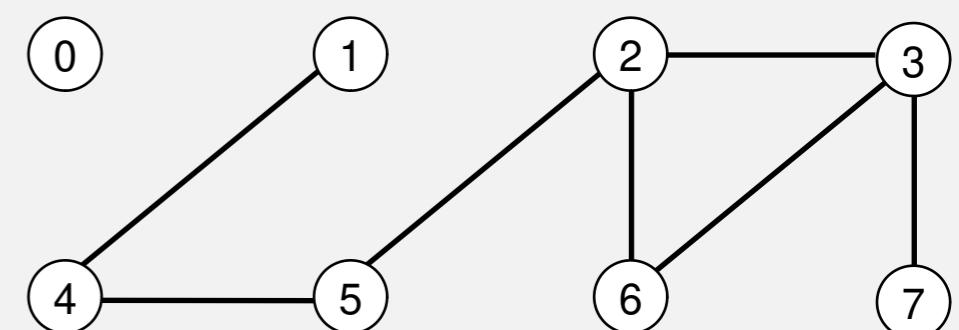
Find. I hvilket komponent er element  $p$ ?

Forbundne. Er elementene  $p$  og  $q$  i samme komponent?

Forén. Erstat komponenterne, der indeholder  $p$  og  $q$ , med deres forening.



union(2, 5)



{ 0 } { 1 4 5 } { 2 3 6 7 }

3 sammenhængskomponenter

{ 0 } { 1 2 3 4 5 6 7 }

2 sammenhængskomponenter

# Union-find datatype (API)

**Formål.** Design en effektiv datastruktur til forén-og-find.

- Antal elementer  $N$  kan være stor.
- Antal operationer  $M$  kan være stor.
- Forén og find operationerne kan komme i vilkårlig rækkefølge.

```
public class UF
```

```
UF(int N)
```

*opret forén-og-find datastruktur  
med  $N$  isolerede elementer (0 til  $N - 1$ )*

```
void union(int p, int q)
```

*forén  $p$  og  $q$*

```
int find(int p)
```

*komponent-representant for  $p$  (0 to  $N - 1$ )*

```
boolean connected(int p, int q)
```

*er  $p$  og  $q$  i samme komponent?*

```
public boolean connected(int p, int q)  
{ return find(p) == find(q); }
```

**1-linjes implementation af connected()**

# Klient til dynamisk sammenhæng

- Indlæs antal elementer  $N$  fra standard input.
- Gentag:
  - Indlæs par af heltal fra standard input
  - Hvis de endnu ikke er forenede, så forén dem og udskriv det par

```
public static void main(String[] args) {  
    int N = StdIn.readInt();  
    UF uf = new UF(N);  
    while (!StdIn.isEmpty()) {  
        int p = StdIn.readInt();  
        int q = StdIn.readInt();  
        if (!uf.connected(p, q)) {  
            uf.union(p, q);  
            StdOut.println(p + " " + q);  
        }  
    }  
}
```

% more tinyUF.txt

10

4 3

3 8

6 5

9 4

2 1

8 9

5 0

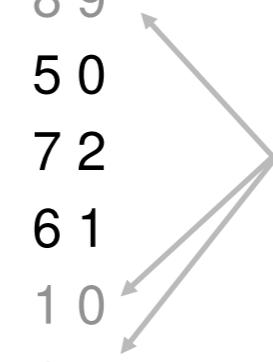
7 2

6 1

1 0

6 7

Allerede forenede



# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 1.5 FORÉN-OG-FIND

---

- ▶ *dynamisk sammenhæng*
- ▶ *hurtig find*
- ▶ *hurtig forening*
- ▶ *forbedringer*
- ▶ *anvendelser*

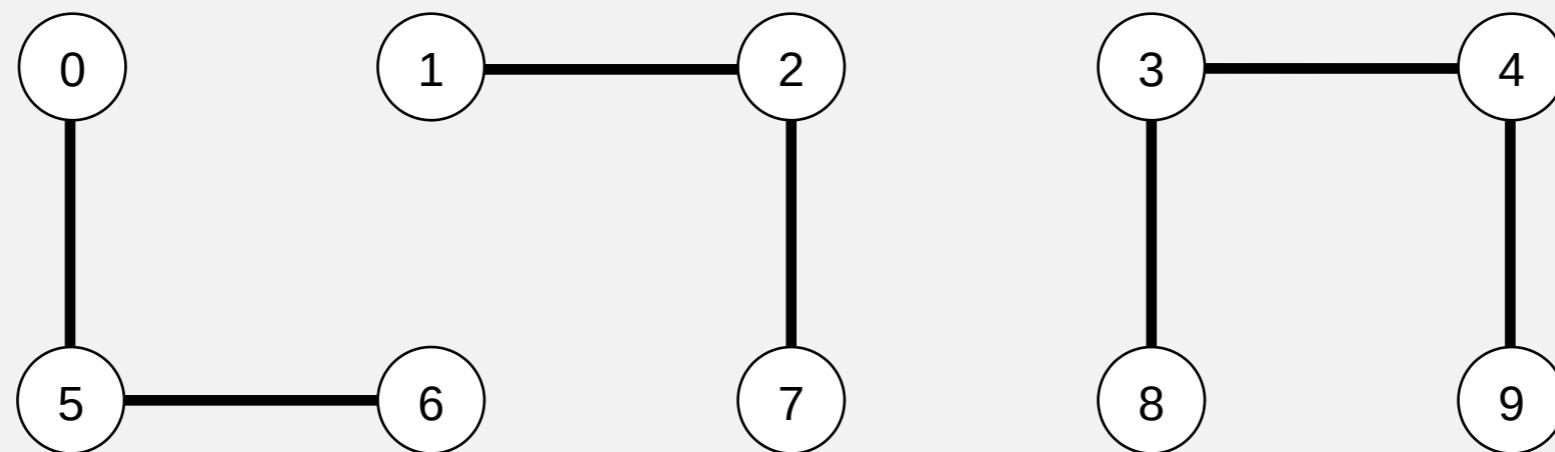
# Hurtig-find [ivrig tilgang]

---

## Datastruktur.

- Array af heltal  $\text{id}[]$  med længde  $N$ .
- Fortolkning:  $\text{id}[p]$  er entydig id for komponenten der indeholder  $p$ .

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	0	1	1	8	8	0	0	1	8	8
						0, 5 og 6 er forbundne 1, 2, og 7 er forbundne 3, 4, 8, og 9 er forbundne				



# Hurtig-find [ivrig tilgang]

## Datastruktur.

- Array af heltal  $\text{id}[]$  med længde  $N$ .
- Fortolkning:  $\text{id}[p]$  er entydig id for komponenten der indeholder  $p$ .

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	0	1	1	8	8	0	0	1	8	8

Find. Hvad er id for  $p$ ?

$\text{id}[6] = 0; \text{id}[1] = 1$

Forbundne. Har  $p$  og  $q$  samme id?

6 og 1 er ikke forbundne

Forén. For at forene komponenterne for  $p$  og  $q$ , erstat alle indgange der er lig med  $\text{id}[p]$  med  $\text{id}[q]$ .

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	1	1	1	8	8	1	1	1	8	8
	↑					↑	↑			

problem: muligvis mange værdier at ændre

# Hurtig-find demo

---

**union(4, 3)**

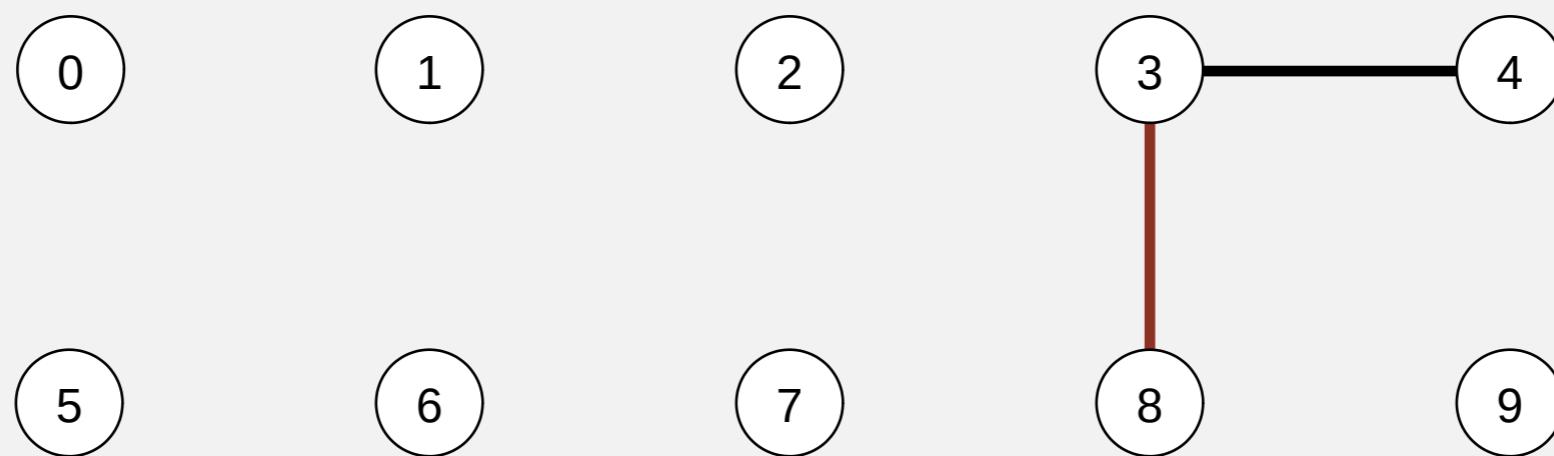


0	1	2	3	4	5	6	7	8	9	
id[]	0	1	2	3	3	5	6	7	8	9
	↑	↑								

# Hurtig-find demo

---

**union(3, 8)**



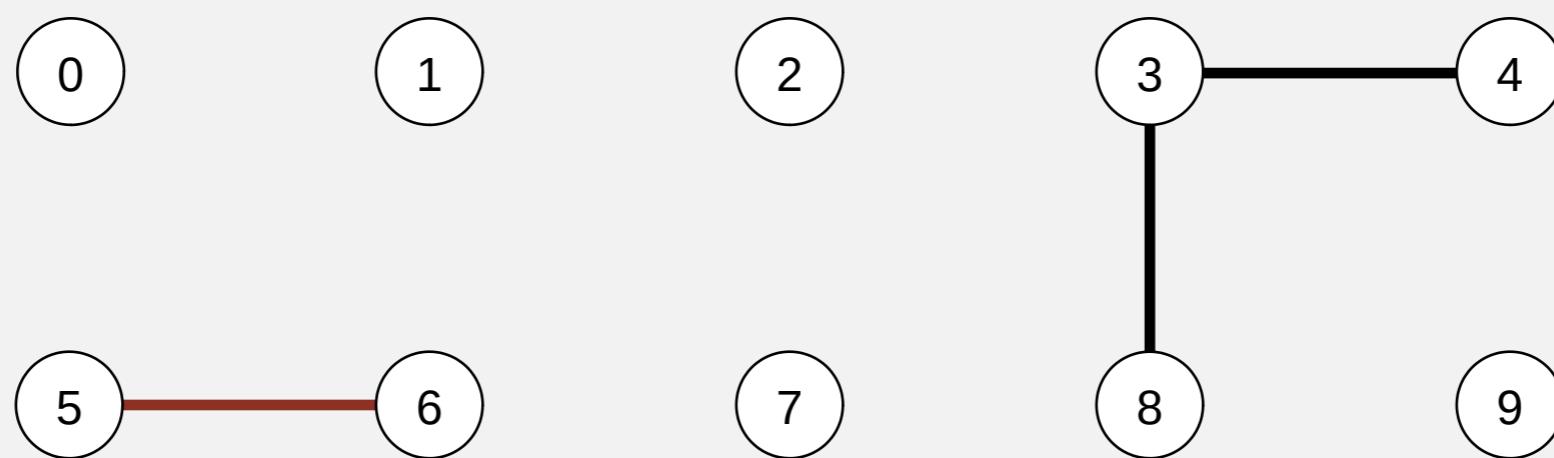
0	1	2	3	4	5	6	7	8	9
0	1	2	8	8	5	6	7	8	9

↑                      ↑

# Hurtig-find demo

---

**union(6, 5)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	<b>5</b>	7	8	9

↑      ↑

# Hurtig-find demo

**union(9, 4)**

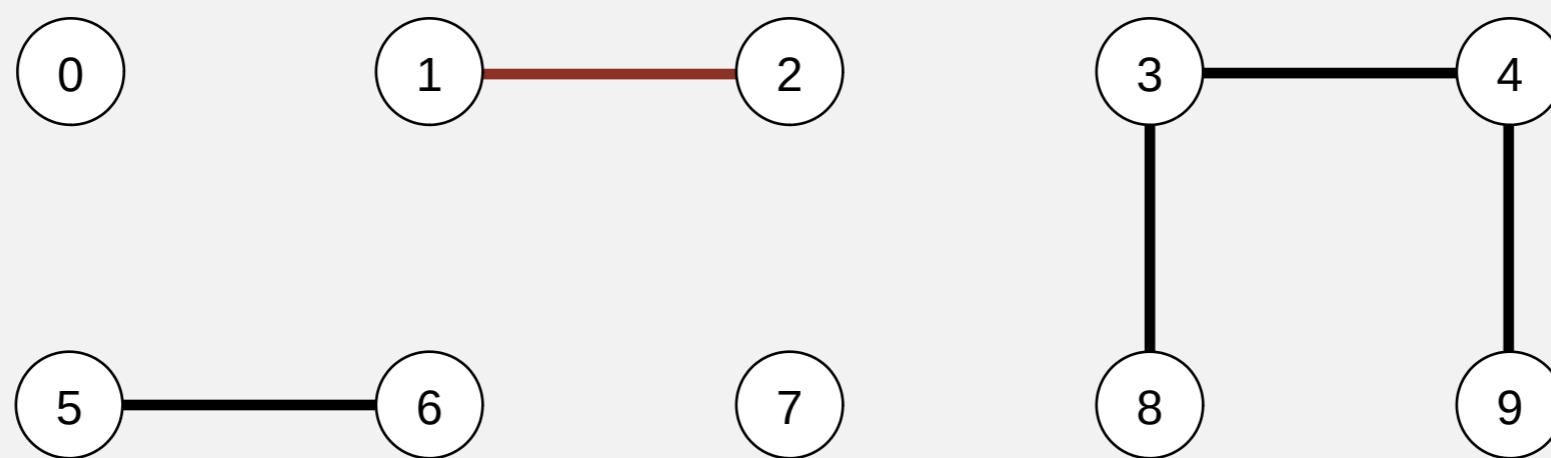


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	8	5	5	7	8	8

# Hurtig-find demo

---

**union(2, 1)**



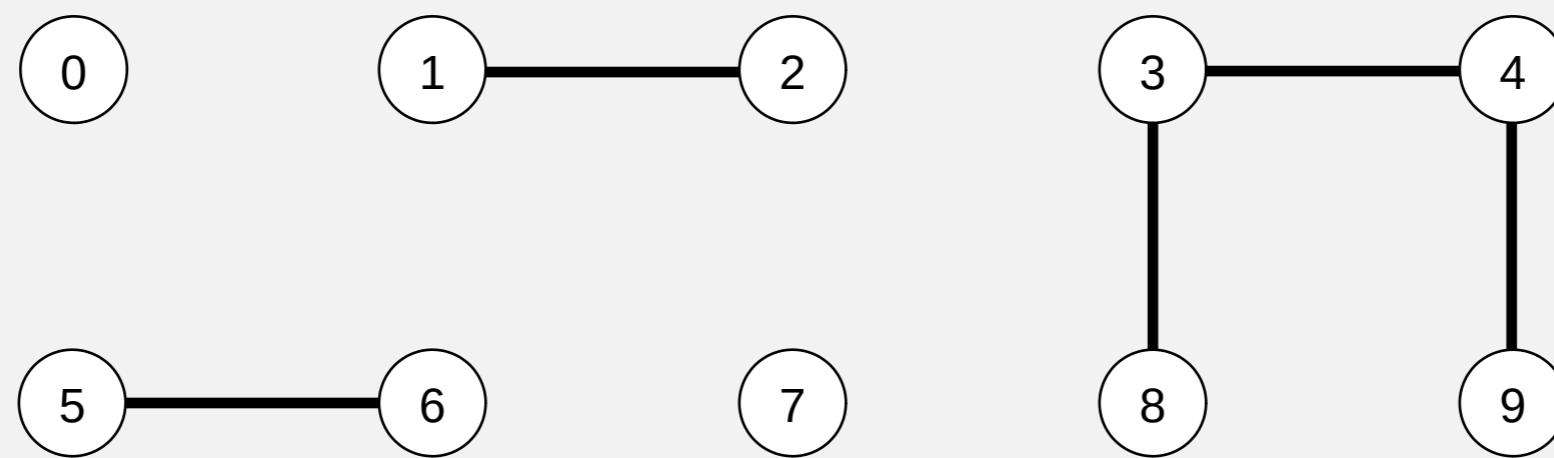
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

↑      ↑

# Hurtig-find demo

---

**connected(8, 9)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8

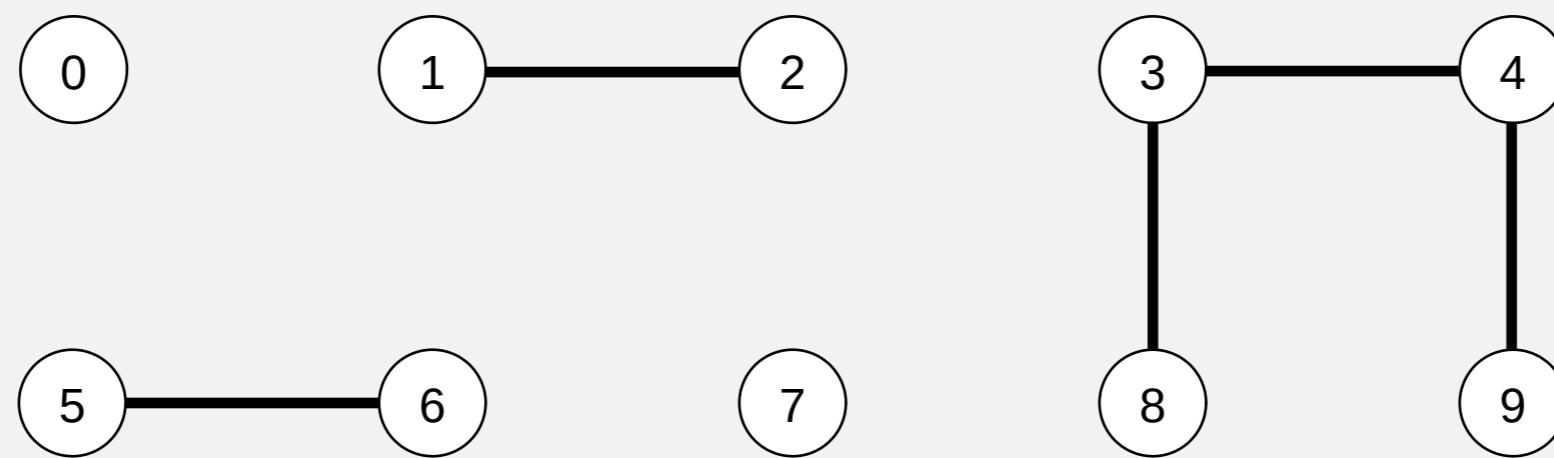


**already connected**

# Hurtig-find demo

---

**connected(5, 0)**

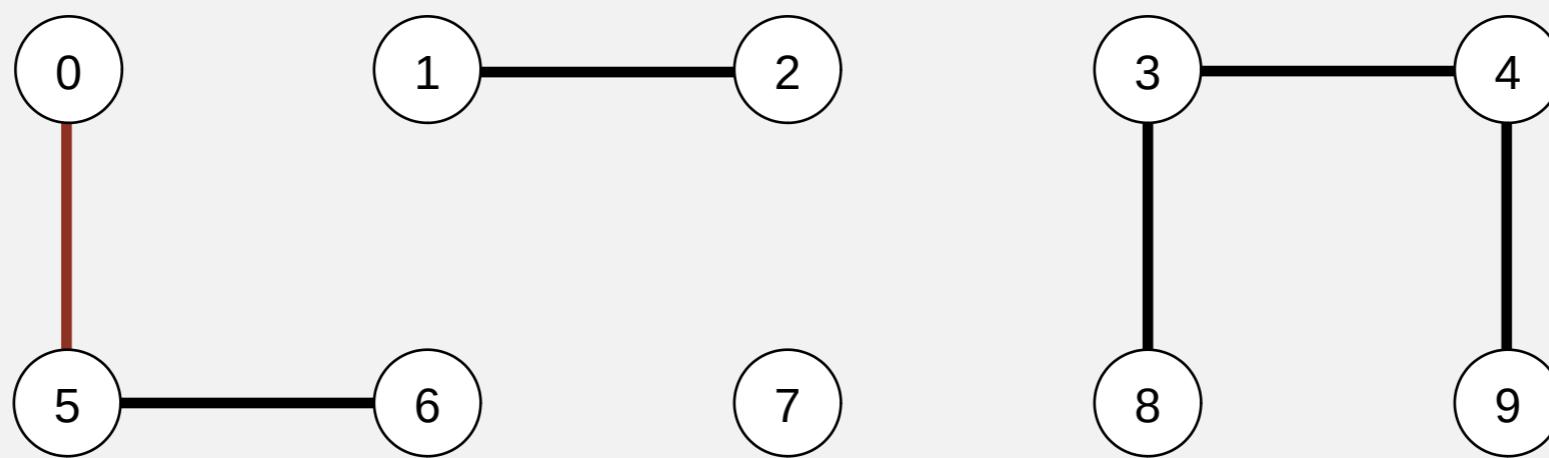


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	5	5	7	8	8
	↑					↑				

**not connected**

# Hurtig-find demo

**union(5, 0)**

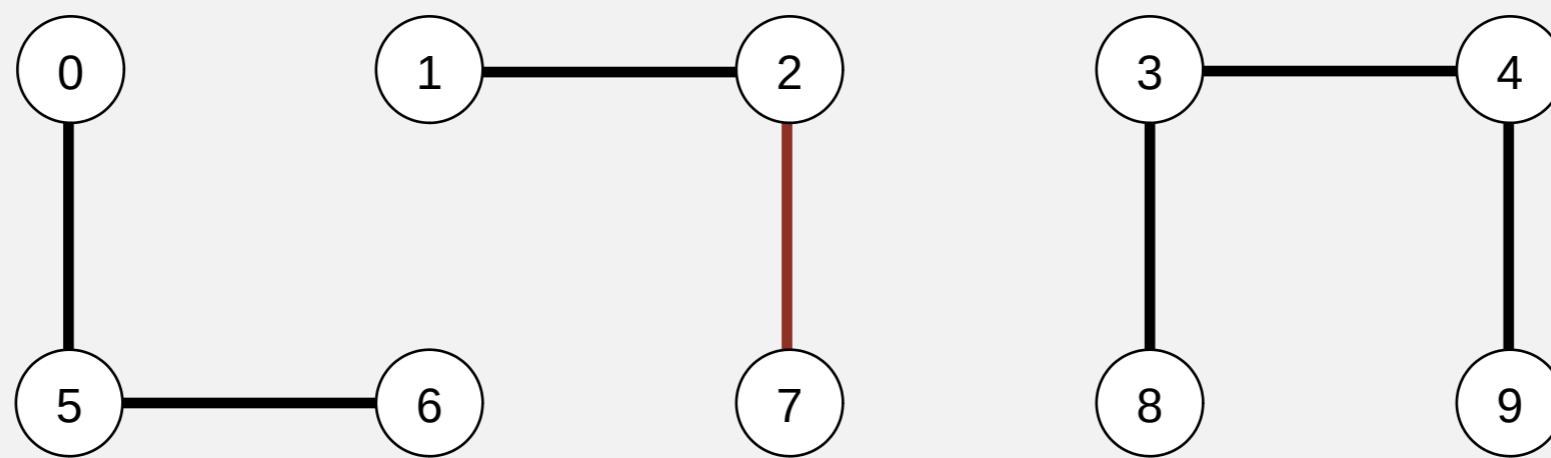


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	7	8	8

# Hurtig-find demo

---

**union(7, 2)**

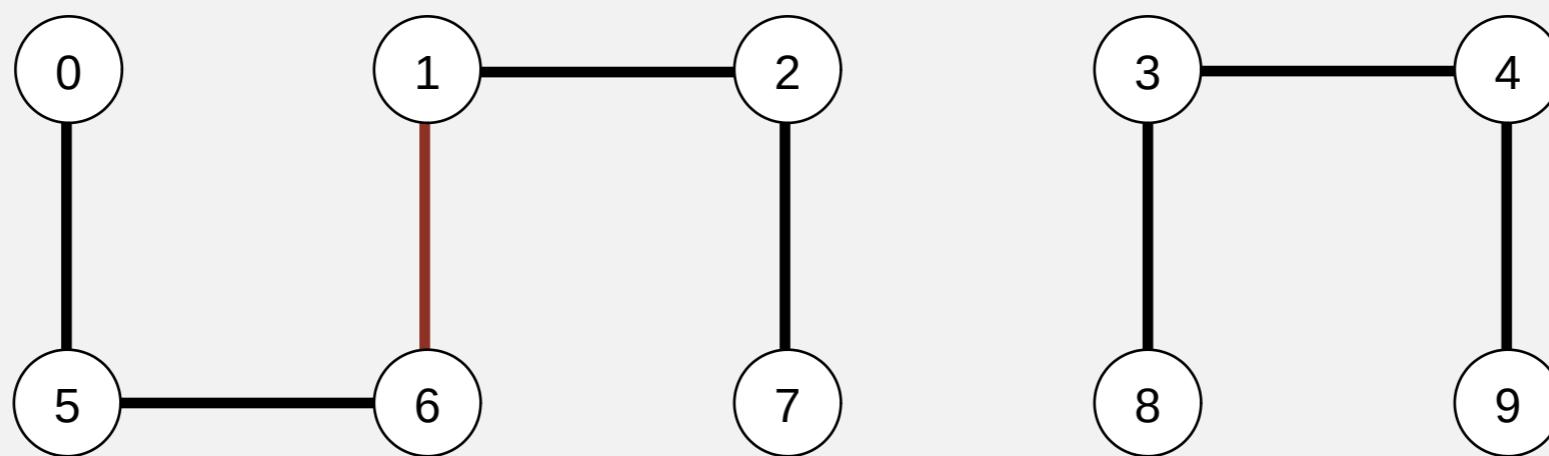


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	1	8	8	0	0	<b>1</b>	8	8

Red arrows point to the '1' at index 7 and the '1' at index 7 in the id[] array.

# Hurtig-find demo

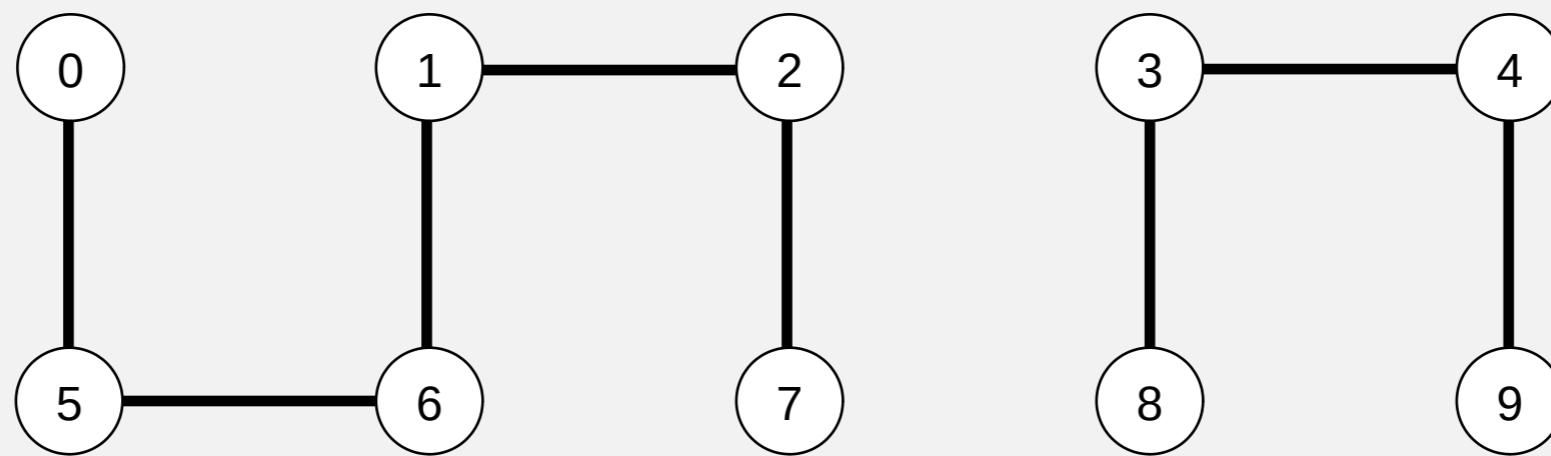
**union(6, 1)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	1	1	1	8	8	1	1	1	8	8

# Hurtig-find demo

---



0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	1	1	1	8	8	1	1	8	8

Live kode

# Hurtig-find er for laaaaangsom

---

Omkostningsmodel. Antal arrayadgange (læsning og skrivning).

algoritme	opret	forén	find	forbundne
<b>hurtig-find</b>	N	N	1	1

Størrelsesorden af antal arrayadgange

Forening er for dyr. Det kan koste op til  $N^2$  arrayadgange at behandle en sekvens med  $N$  foreningsoperationer på  $N$  elementer.

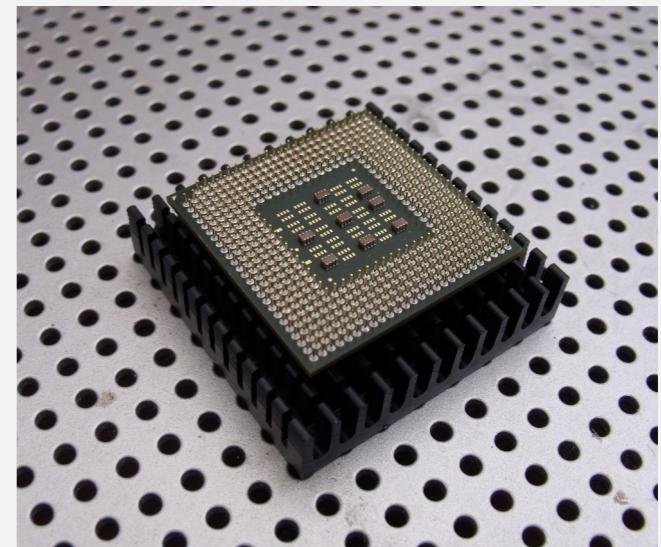
kvadratisk

# Kvadratiske algoritmer skalerer ikke

## Tommelfingerregler.

- $10^9$  operationer i sekundet.
- $10^9$  ord i hukommelsen.
- Tilgå alle ord på ca. 1 sekund.

Har været  
nogenlunde sandt  
siden 1950!

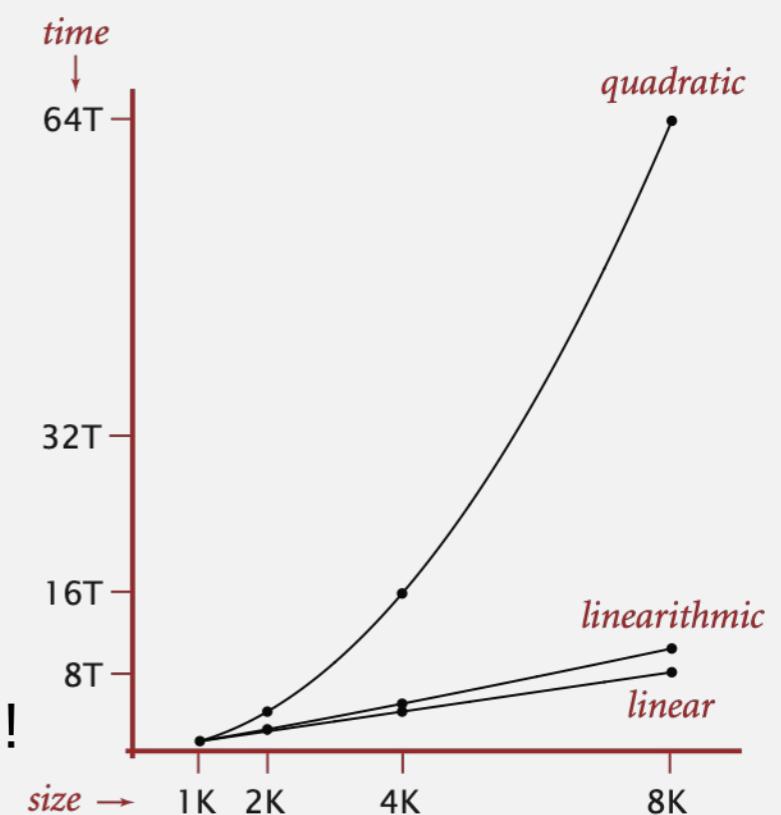


## Stort problem for quick-find.

- $10^9$  union operationer på  $10^9$  elementer.
- Hurtig-find kan bruge op mod  $10^{18}$  hukommelsesadgange.
- 30+ års beregningstid!

## Kvadratiske algoritmer skalerer ikke med teknologi.

- En ny computer er måske 10x hurtigere.
- Men, med 10x så meget hukommelse  $\Rightarrow$  kan arbejde på 10x så stort et problem.
- Med en kvadratisk algoritm tager det 10x så lang tid!



# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 1.5 FORÉN-OG-FIND

---

- ▶ *dynamisk sammenhæng*
- ▶ *hurtig find*
- ▶ *hurtig forening*
- ▶ *forbedringer*
- ▶ *anvendelser*

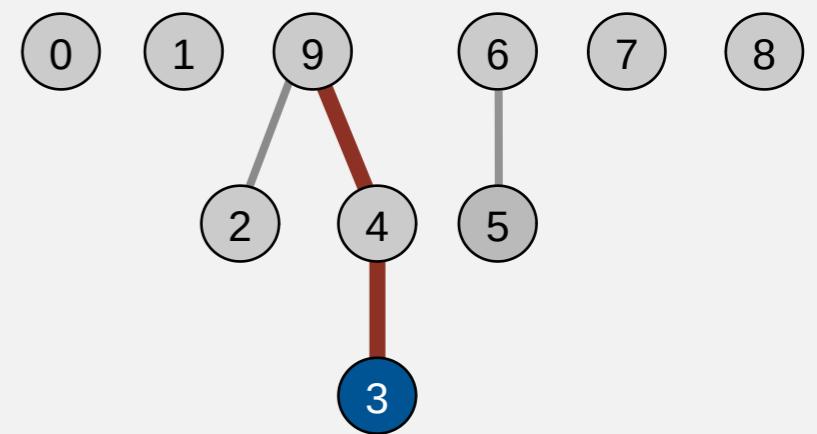
# Hurtig-forening [doven tilgang]

## Datastruktur.

- Array af heltal  $\text{id}[]$  med længde  $N$ .
- Fortolkning:  $\text{id}[i]$  er forælder af  $i$ .
- **Roden** af  $i$  er  $\text{id}[\text{id}[\text{id}[\dots\text{id}[i]\dots]]]$ .

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	0	1	9	4	9	6	6	7	8	9

Fortsæt indtil det ikke ændrer sig  
(algoritmen garanterer at det stopper)



Forælderen af 3 er 4

Forælderen af 4 er 9

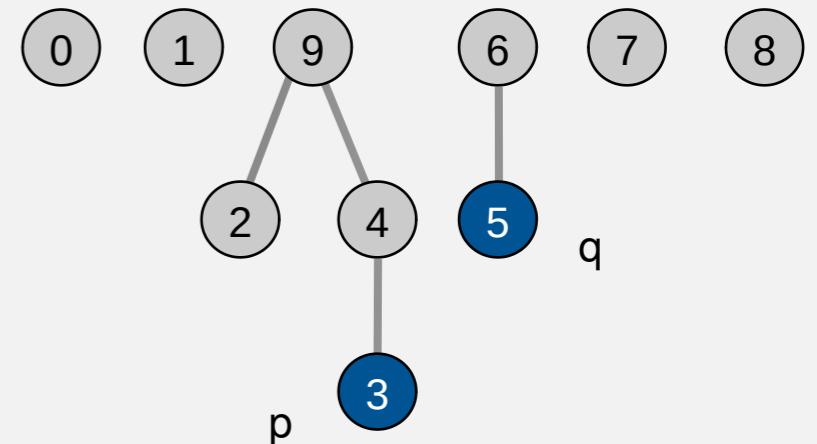
Roden af 3 er 9

# Hurtig-forening [doven tilgang]

## Datastruktur.

- Array af heltal  $\text{id}[]$  med længde  $N$ .
- Fortolkning:  $\text{id}[i]$  er forælder af  $i$ .
- Roden af  $i$  er  $\text{id}[\text{id}[\text{id}[\dots\text{id}[i]\dots]]]$ .

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	0	1	9	4	9	6	6	7	8	9



roden af 3 er 9

roden af 5 er 6

3 og 5 er ikke forenede

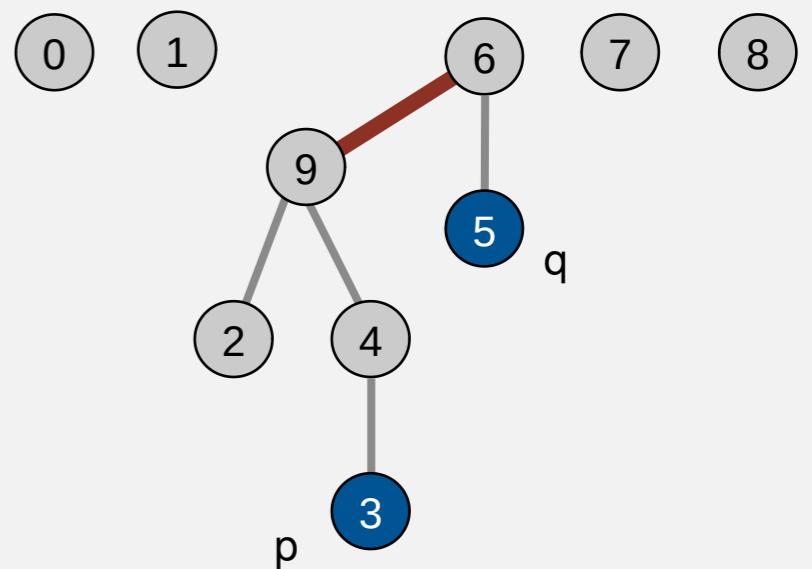
Find. Hvad er rodens af  $p$ ?

Forbundne. Har  $p$  og  $q$  samme rod?

Forén. For at forene komponenterne af  $p$  og  $q$ , skift forælderen af  $p$ 's rod til at være  $q$ 's rod.

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	0	1	9	4	9	6	6	7	8	6

Kun én værdi skal ændres



# Hurtig-forening demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Hurtig-forening demo

---

**union(4, 3)**

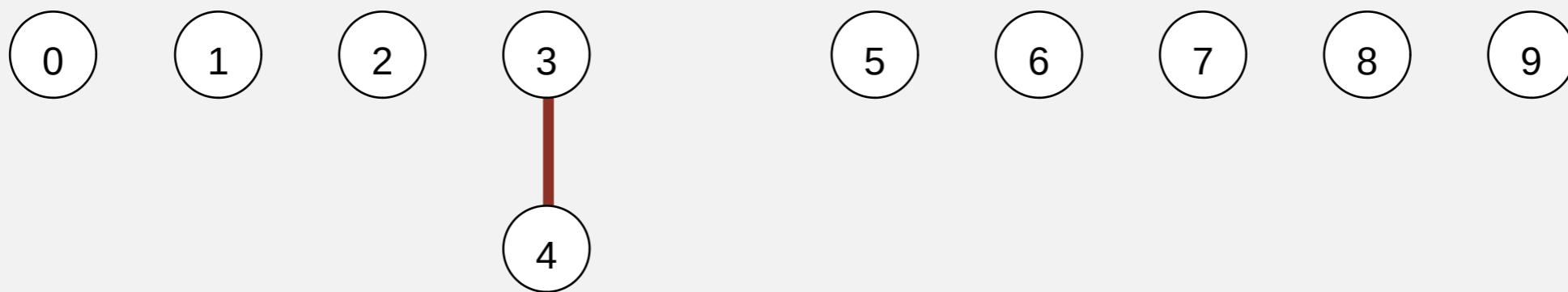


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Hurtig-forening demo

---

**union(4, 3)**



0	1	2	3	4	5	6	7	8	9	
id[]	0	1	2	3	3	5	6	7	8	9

# Hurtig-forening demo

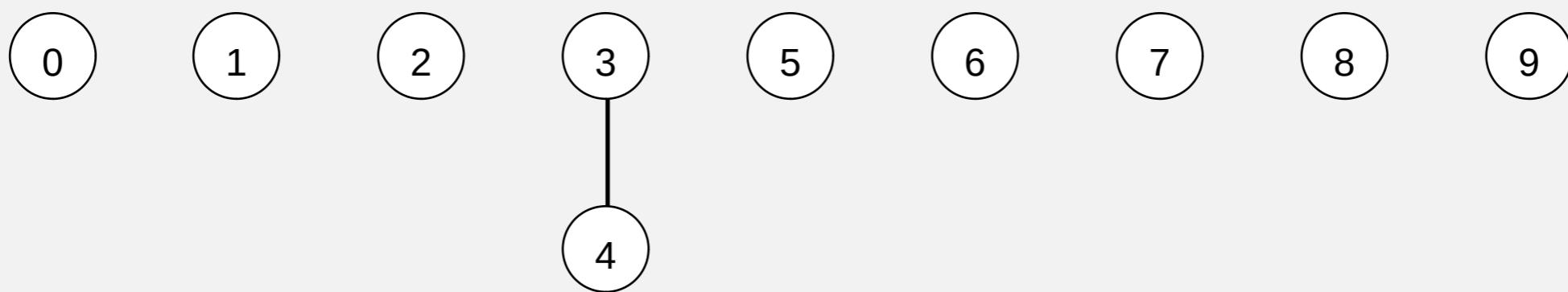


	0	1	2	3	4	5	6	7	8	9
<b>id</b>	0	1	2	3	3	5	6	7	8	9

# Hurtig-forening demo

---

**union(3, 8)**

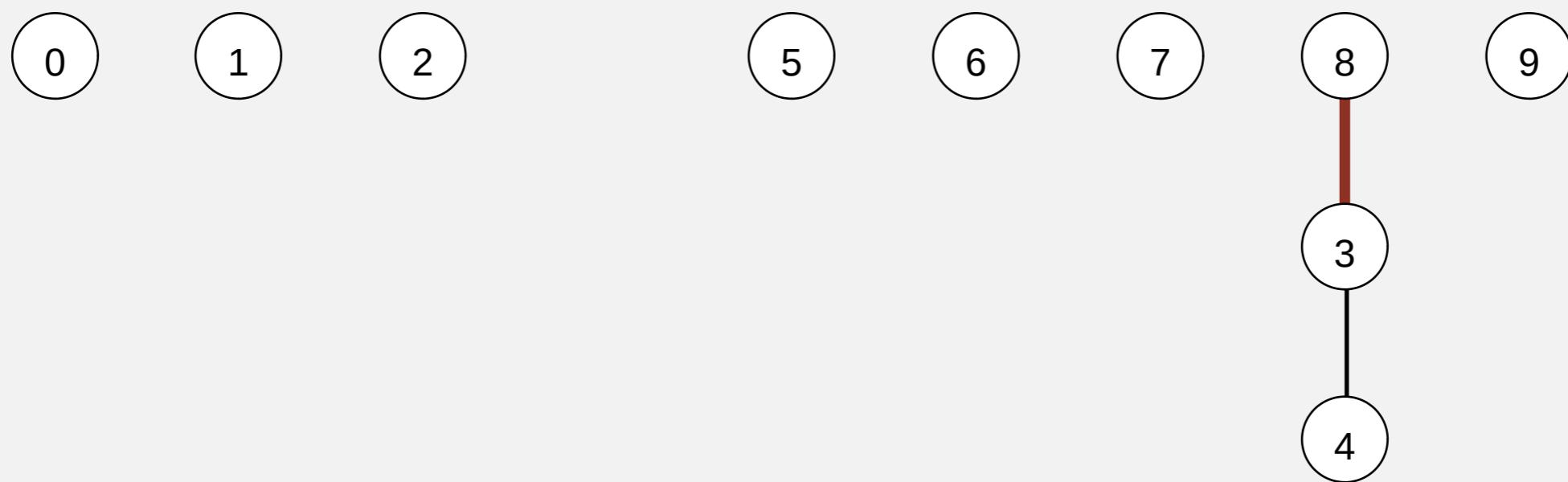


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	3	5	6	7	8	9

# Hurtig-forening demo

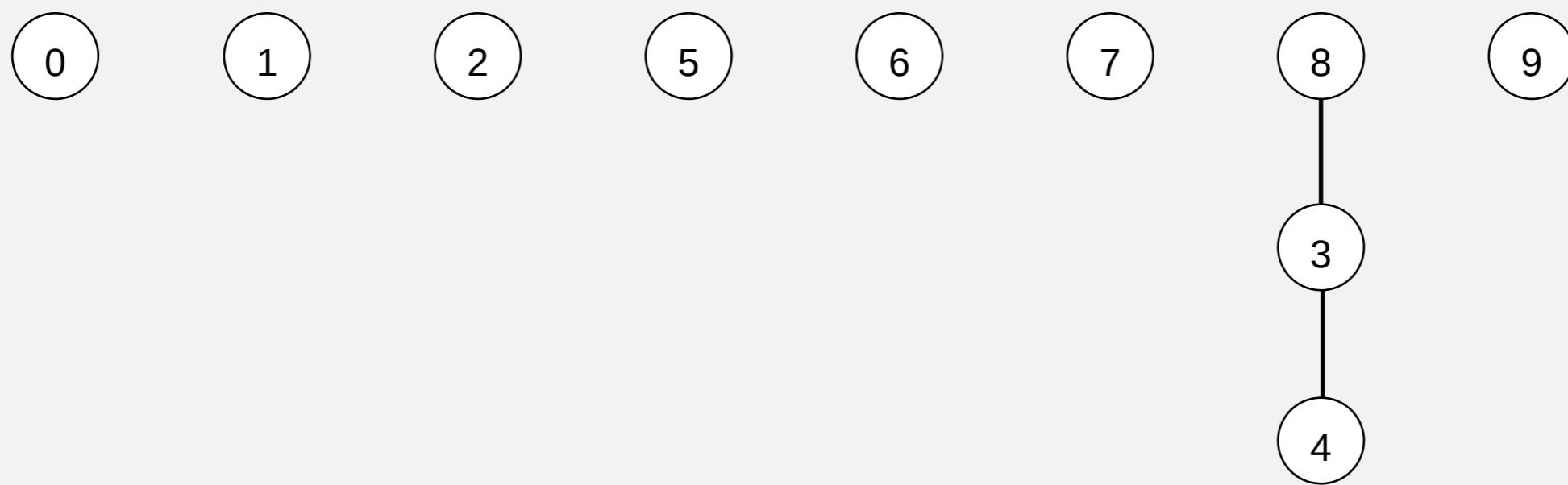
---

**union(3, 8)**



0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	6	7	8	9

# Hurtig-forening demo

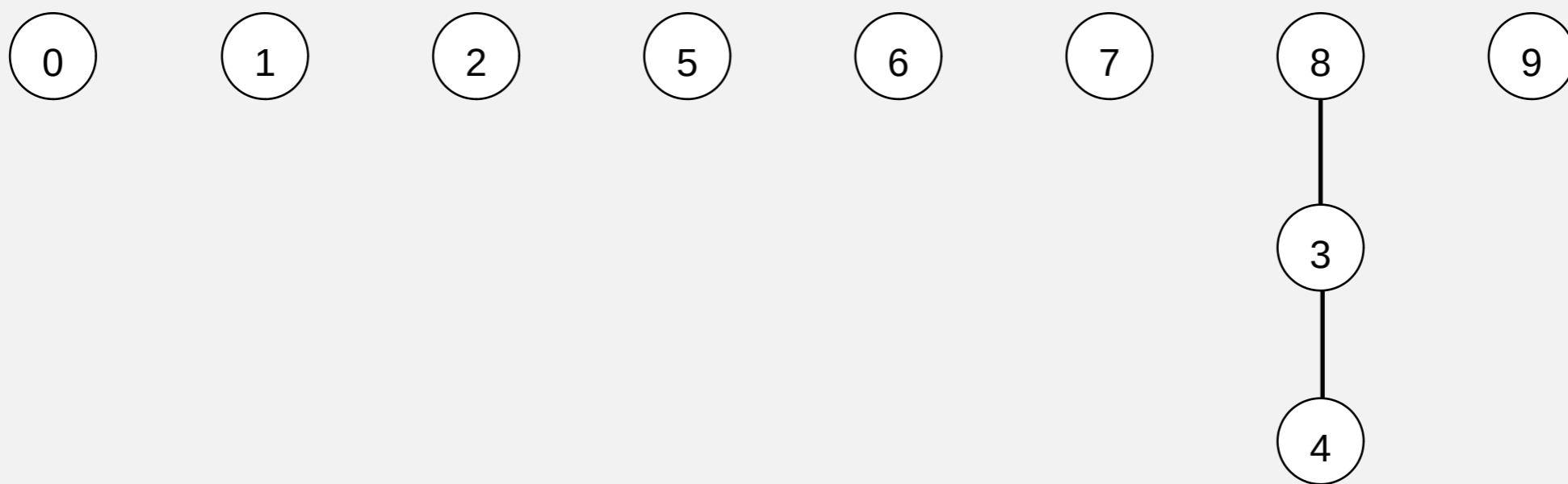


id

# Hurtig-forening demo

---

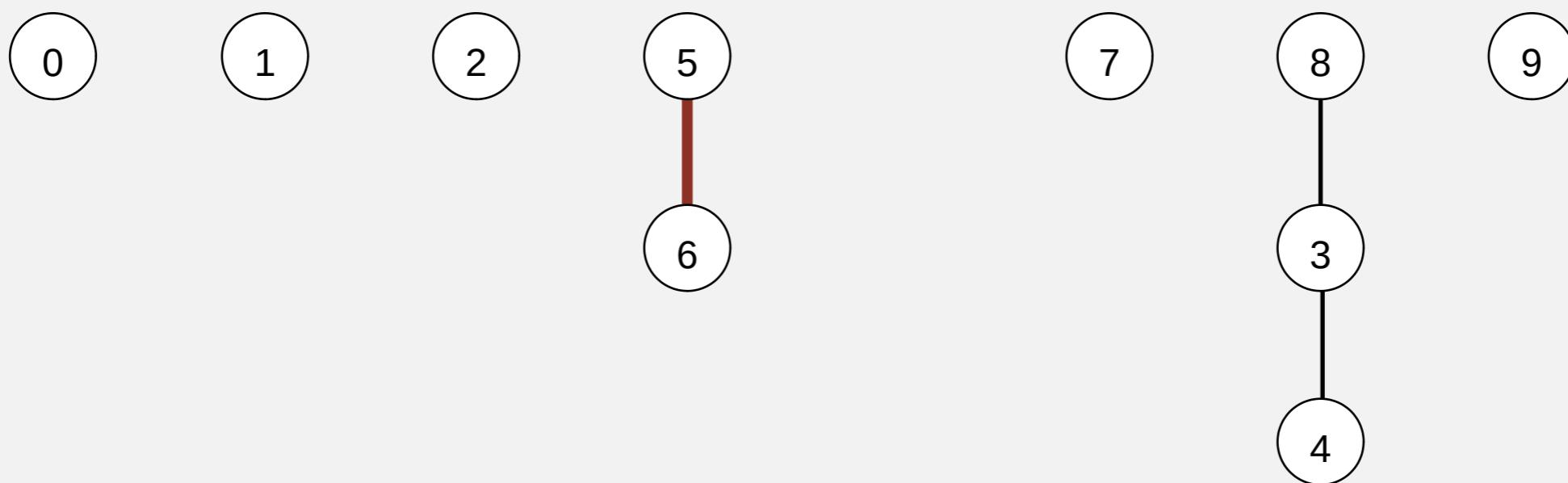
**union(6, 5)**



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	8	3	5	6	7	8	9

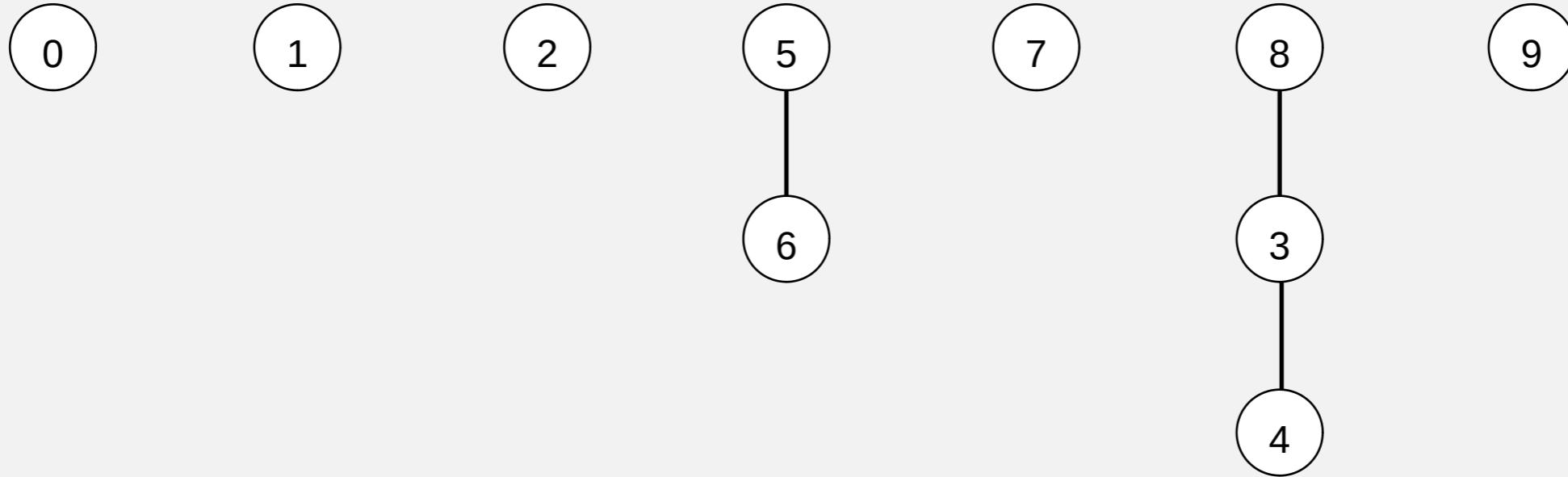
# Hurtig-forening demo

**union(6, 5)**



**id** [ ]

# Hurtig-forening demo



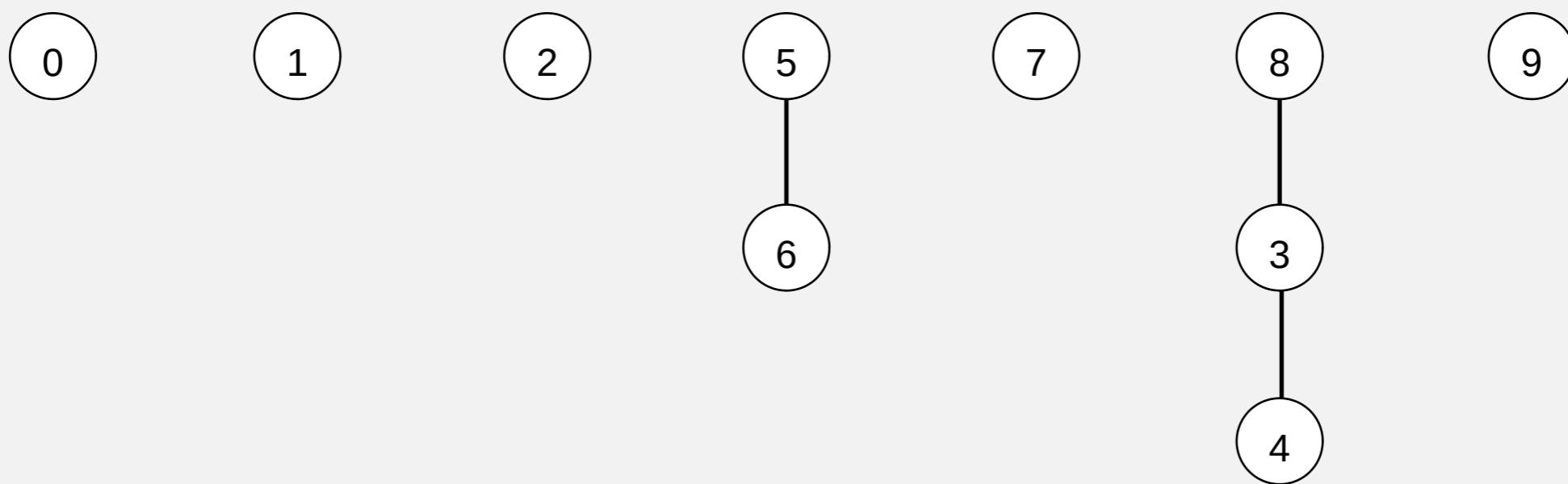
**id** □

0	1	2	8	3	5	5	7	8	9
---	---	---	---	---	---	---	---	---	---

# Hurtig-forening demo

---

**union(9, 4)**

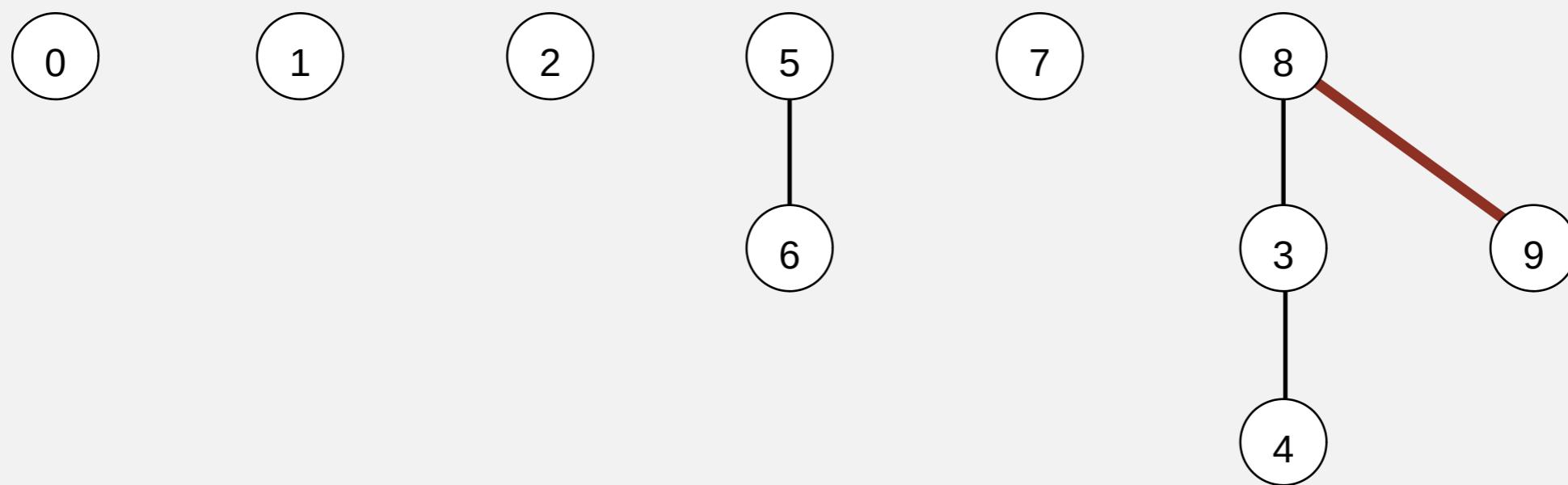


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	9

# Hurtig-forening demo

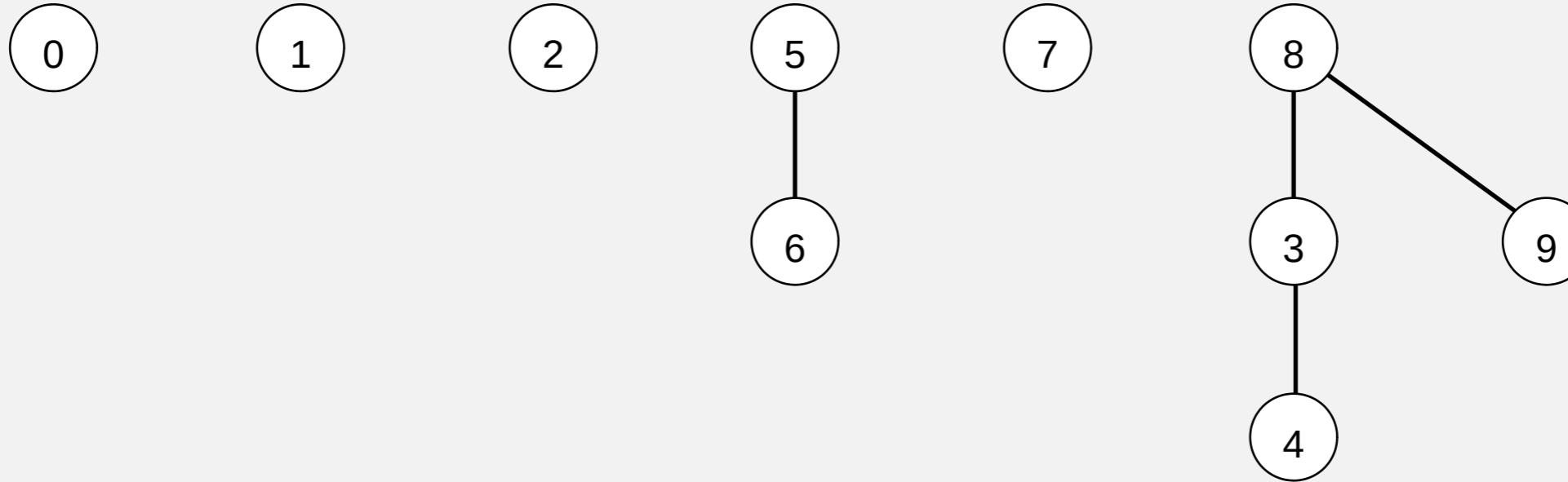
---

**union(9, 4)**



0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	8

# Hurtig-forening demo



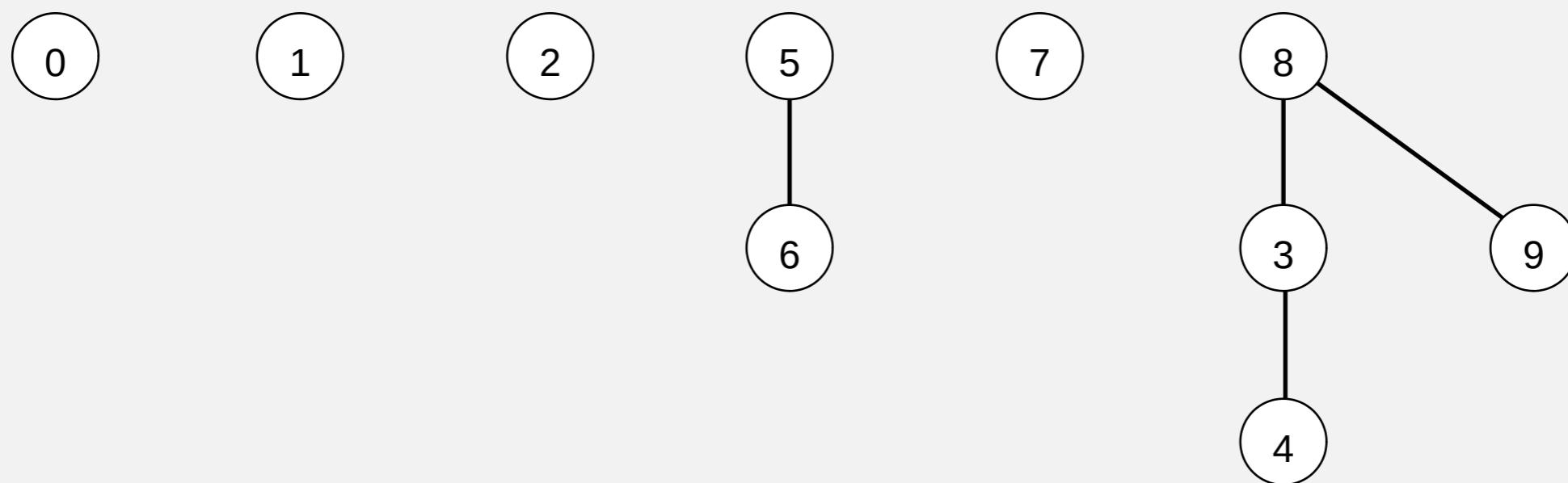
**id** □

0	1	2	8	3	5	5	7	8	8
---	---	---	---	---	---	---	---	---	---

# Hurtig-forening demo

---

**union(2, 1)**

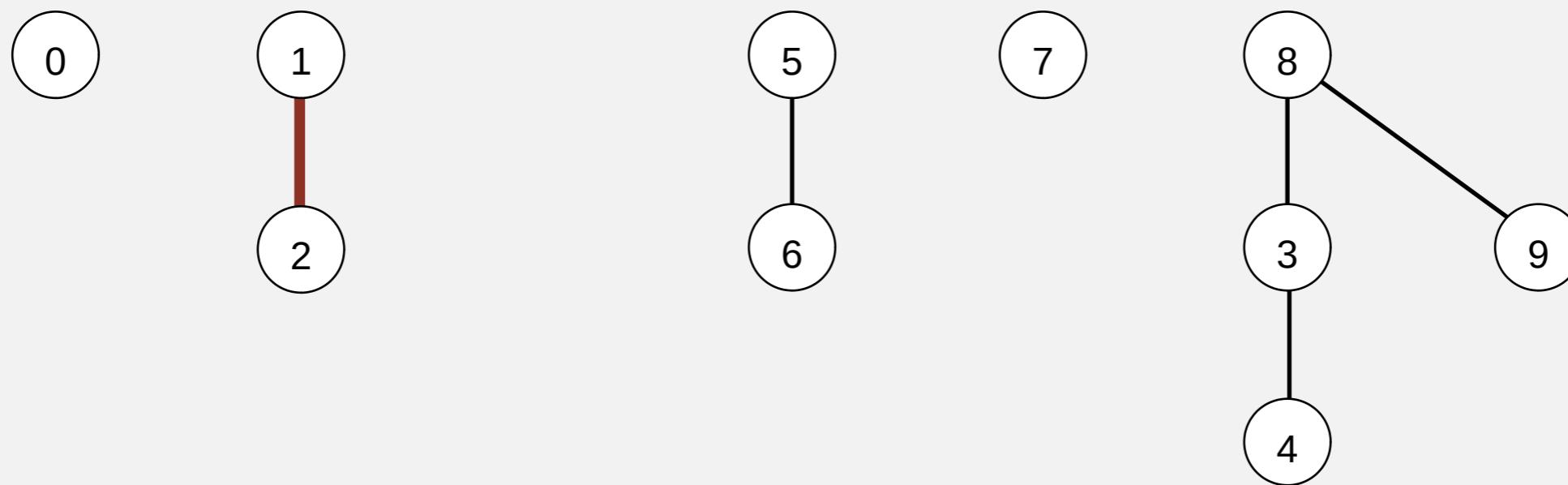


0	1	2	3	4	5	6	7	8	9
id[]					0	1	2	8	8

# Hurtig-forening demo

---

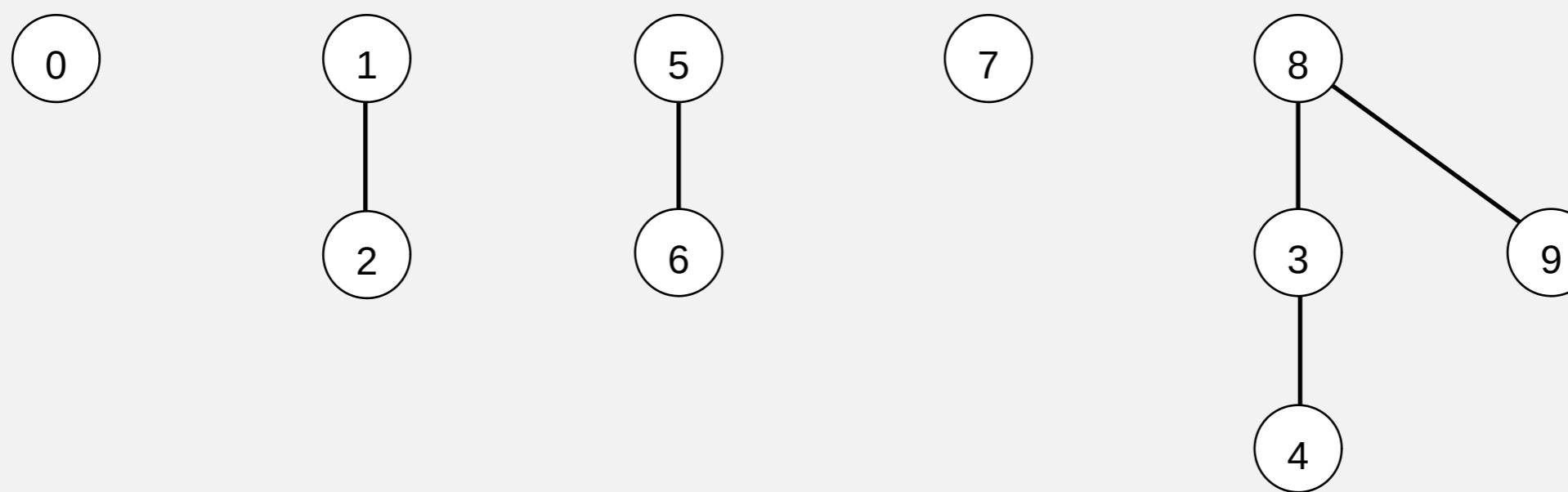
**union(2, 1)**



0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Hurtig-forening demo

---

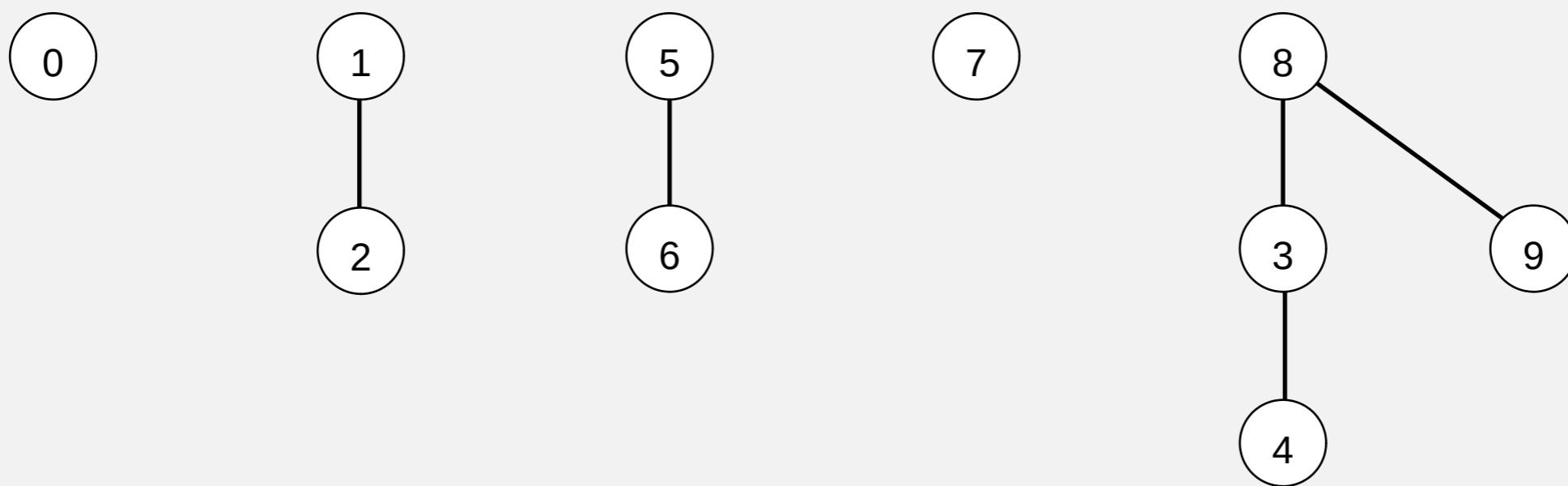


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Hurtig-forening demo

---

connected(8, 9)

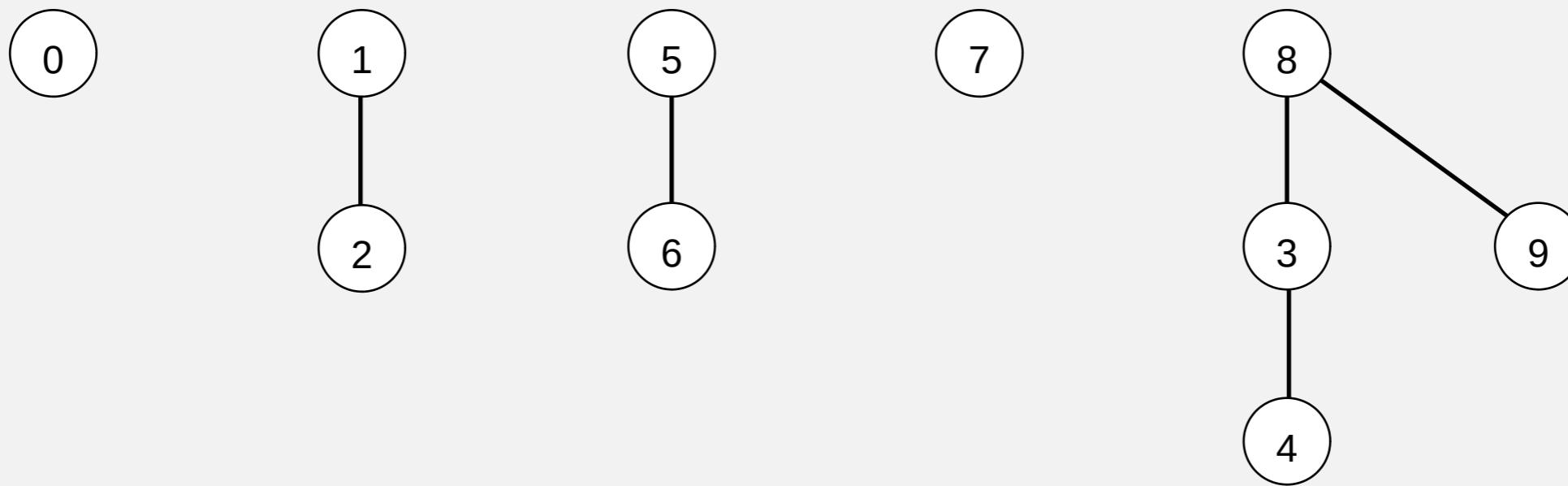


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Hurtig-forening demo

# connected(5, 4)

X



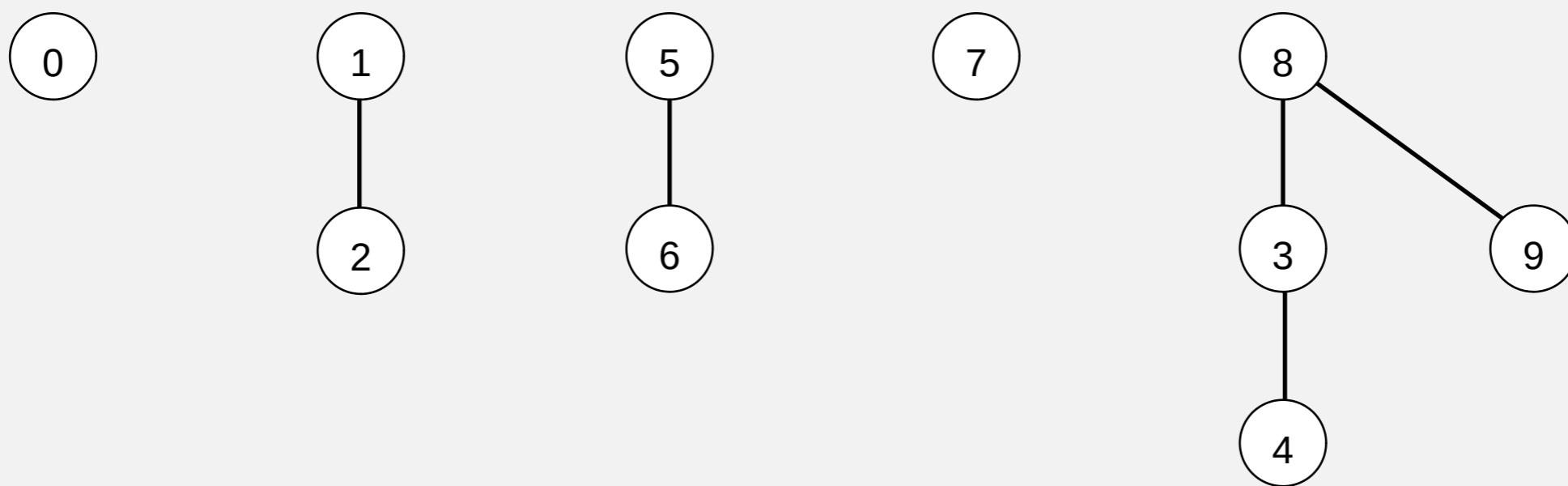
**id**

0	1	1	8	3	5	5	7	8	8
---	---	---	---	---	---	---	---	---	---

# Hurtig-forening demo

---

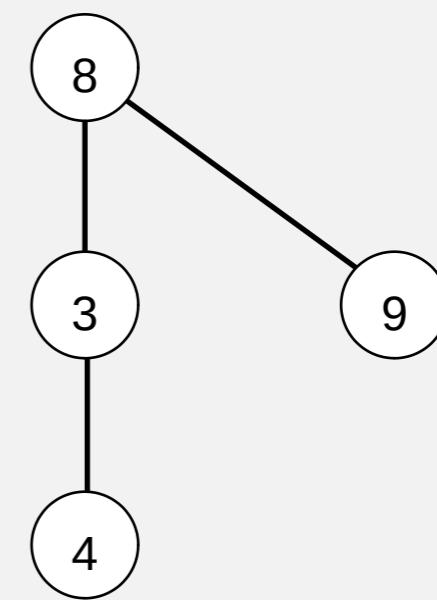
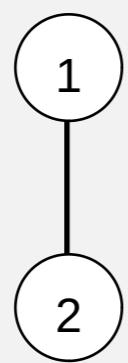
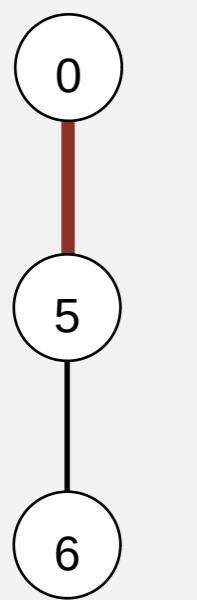
**union(5, 0)**



0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

# Hurtig-forening demo

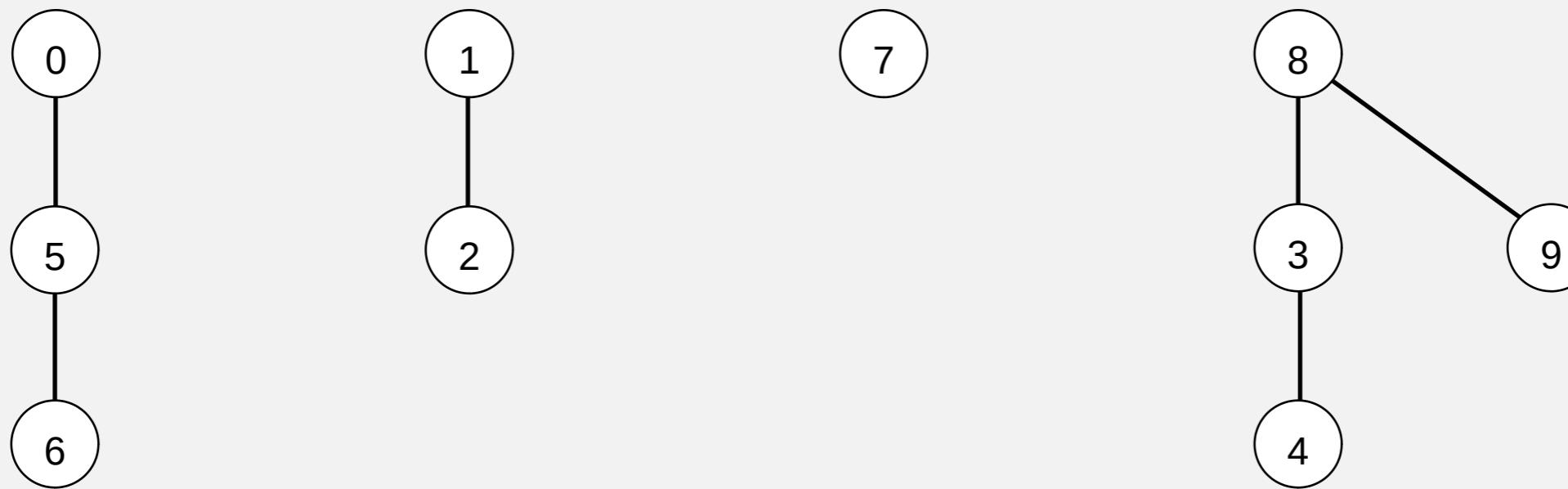
**union(5, 0)**



id

# Hurtig-forening demo

---

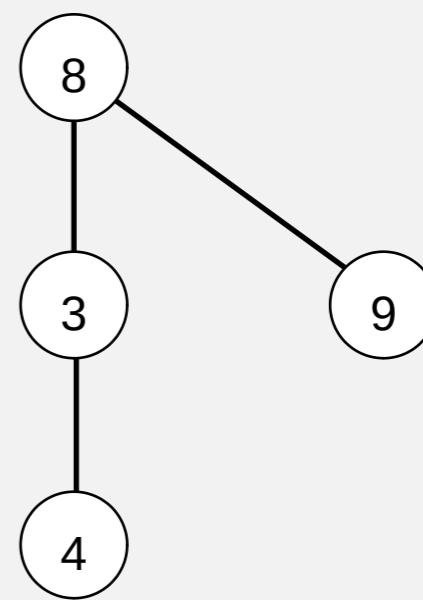
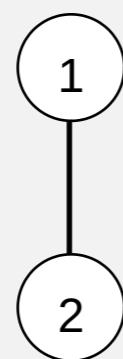


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

# Hurtig-forening demo

---

**union(7, 2)**

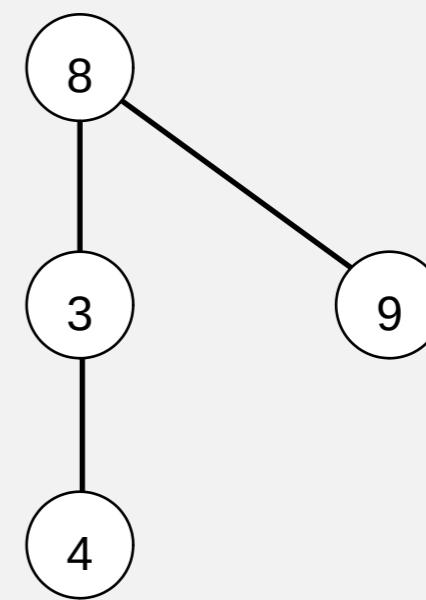
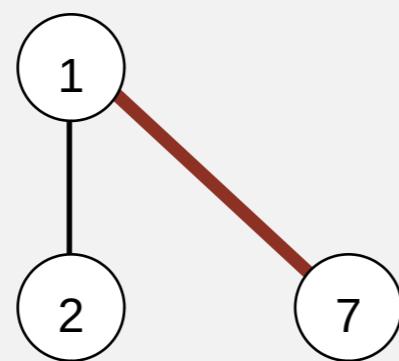


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

**id[]**

# Hurtig-forening demo

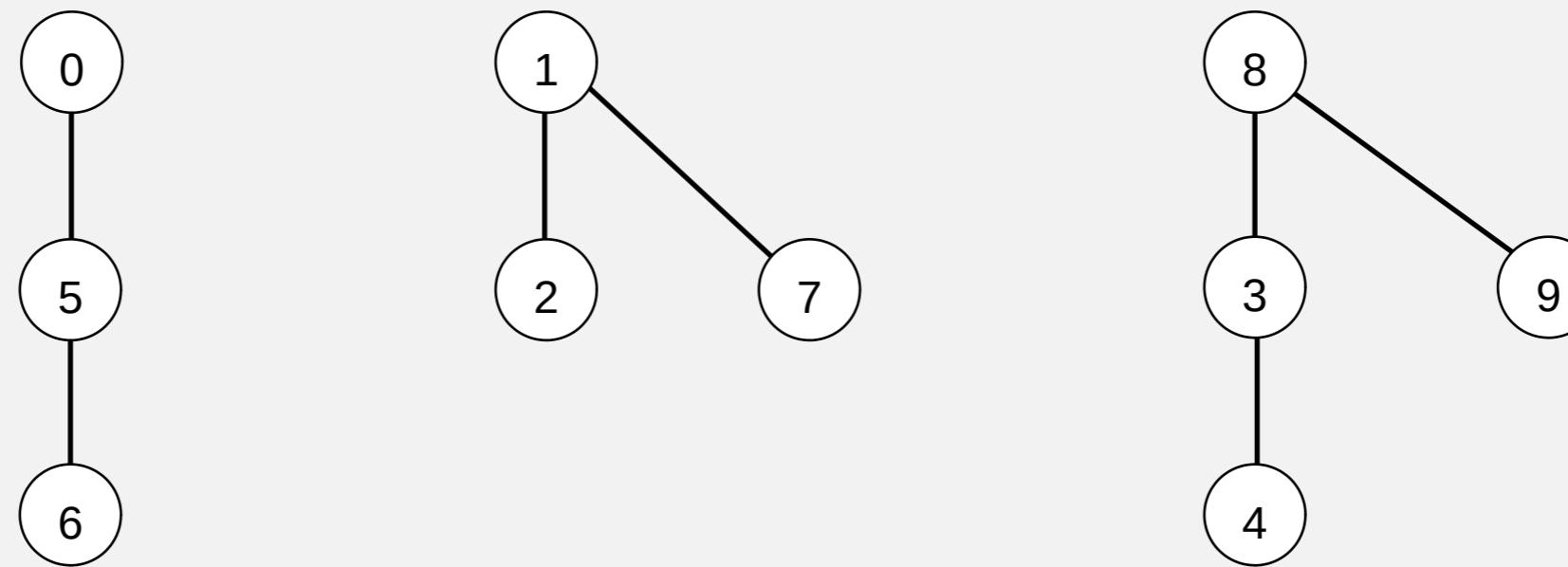
**union(7, 2)**



0      1      2      3      4      5      6      7      8      9

**id**

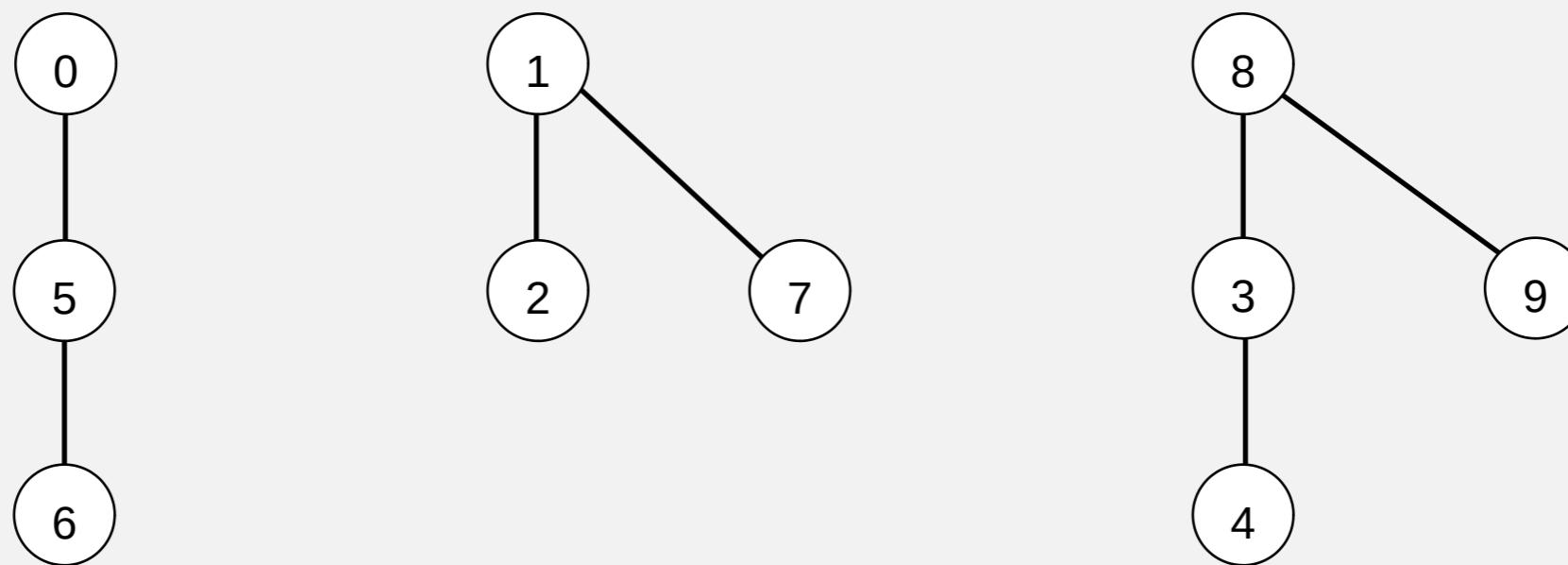
# Hurtig-forening demo



id

# Hurtig-forening demo

**union(6, 1)**

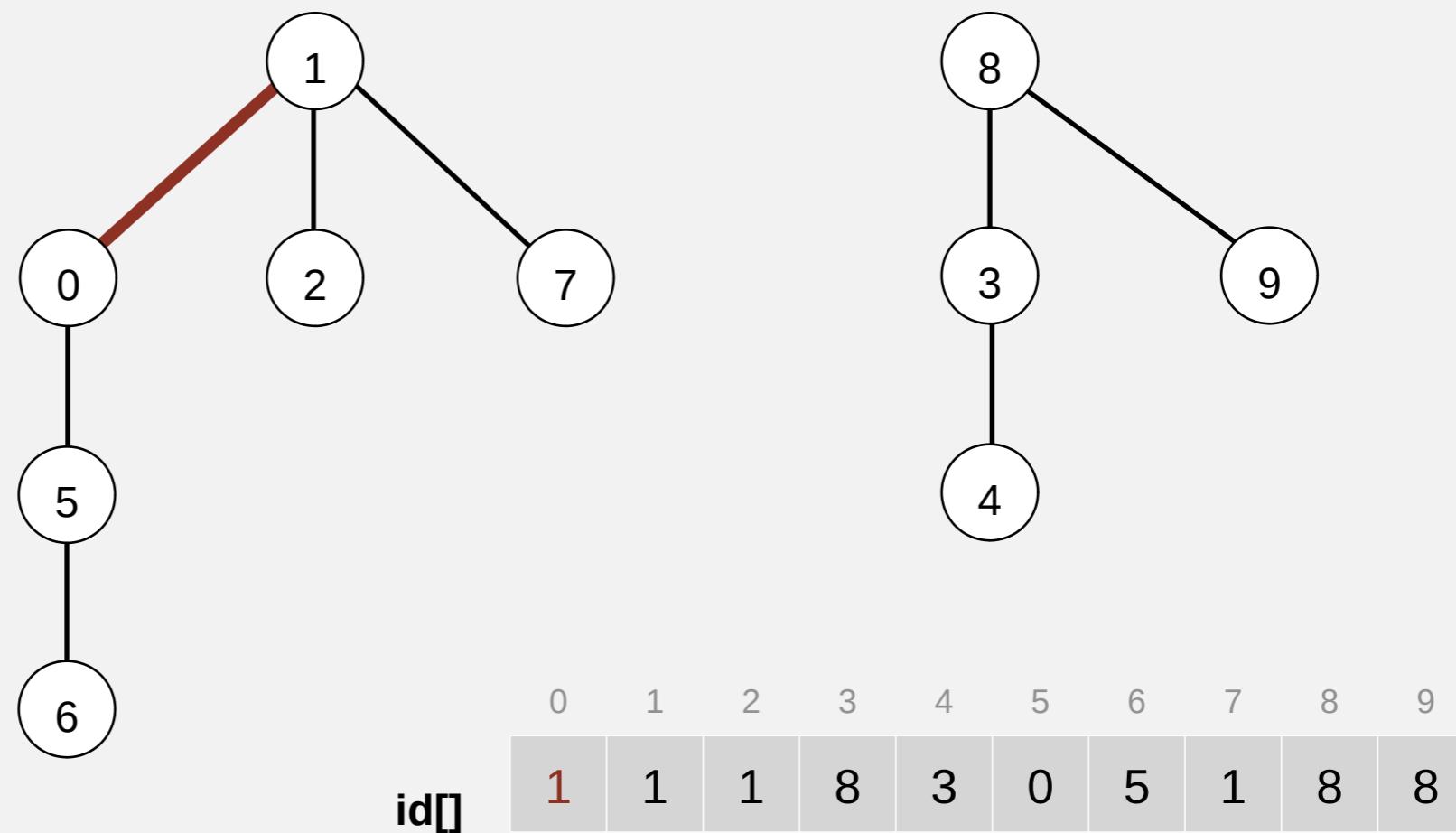


id

# Hurtig-forening demo

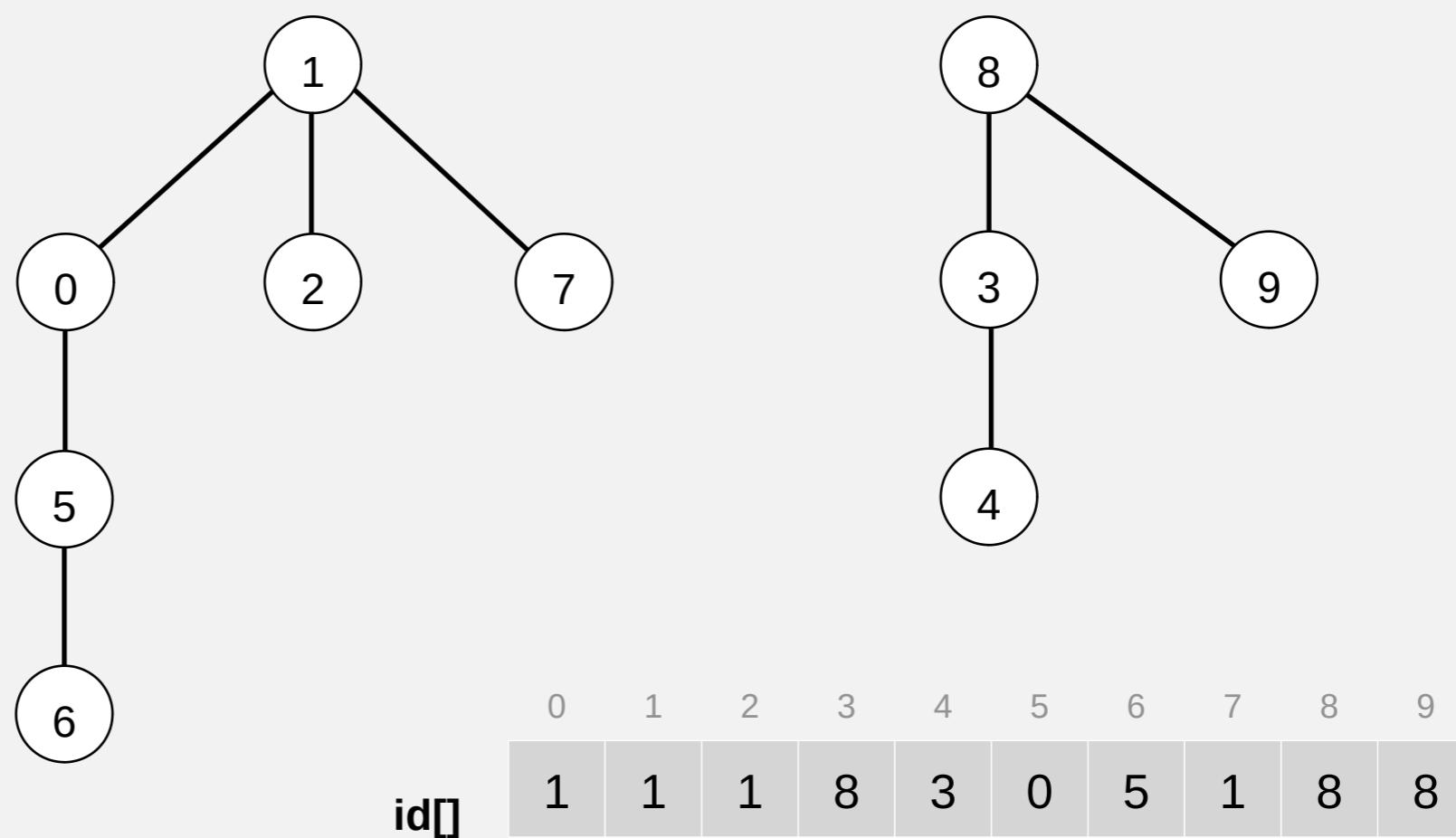
---

**union(6, 1)**



# Hurtig-forening demo

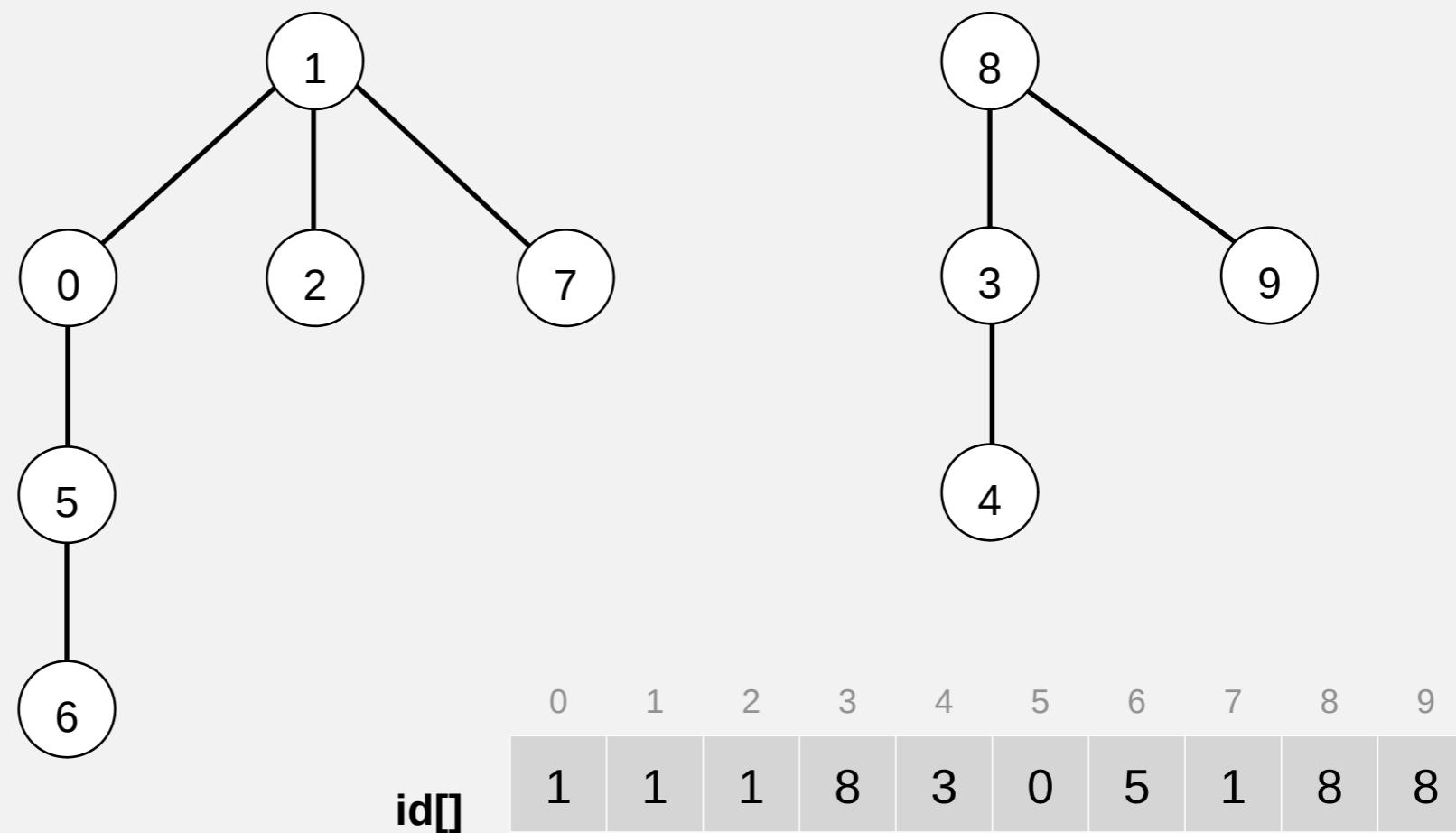
---



# Hurtig-forening demo

---

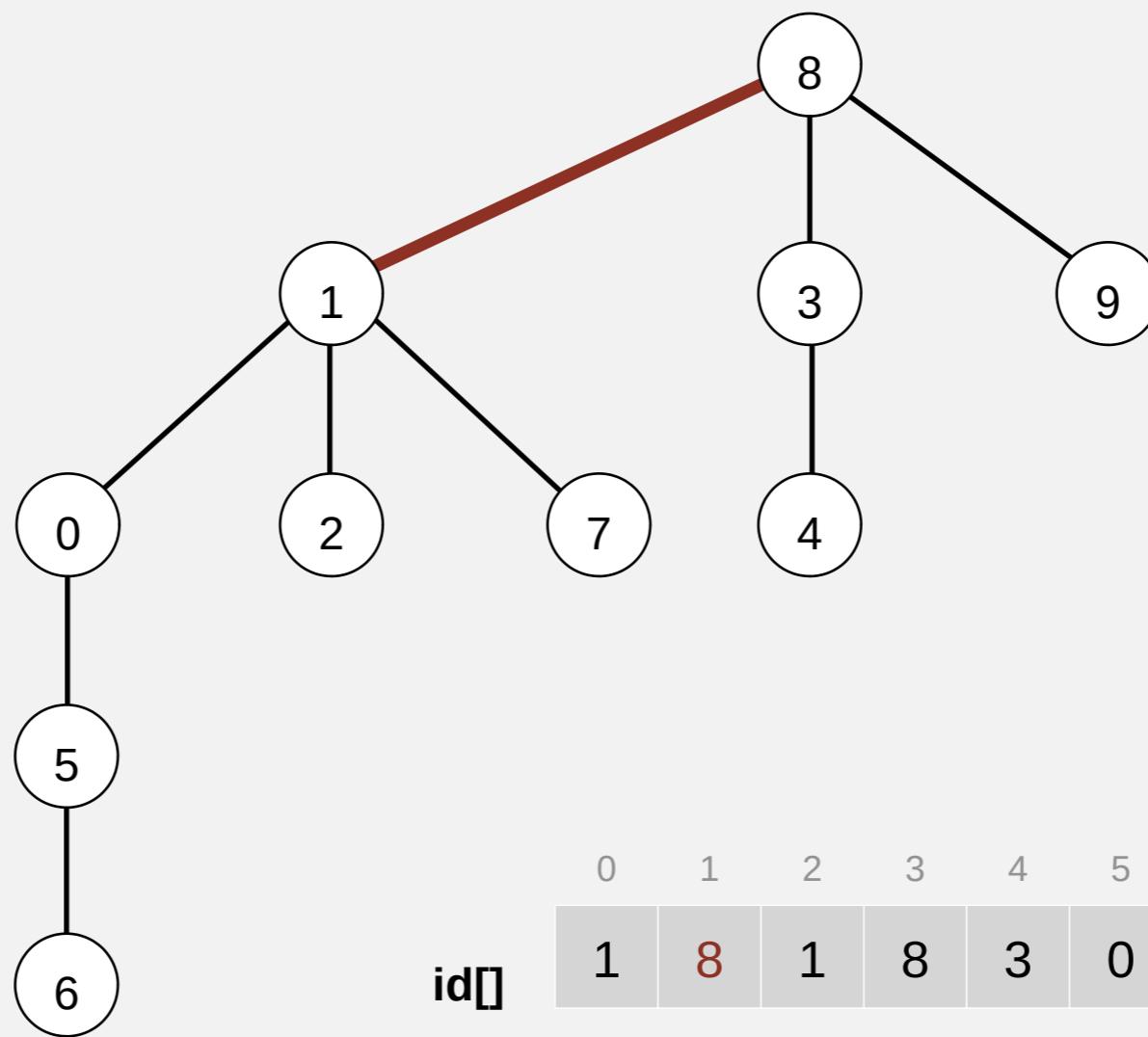
**union(7, 3)**



# Hurtig-forening demo

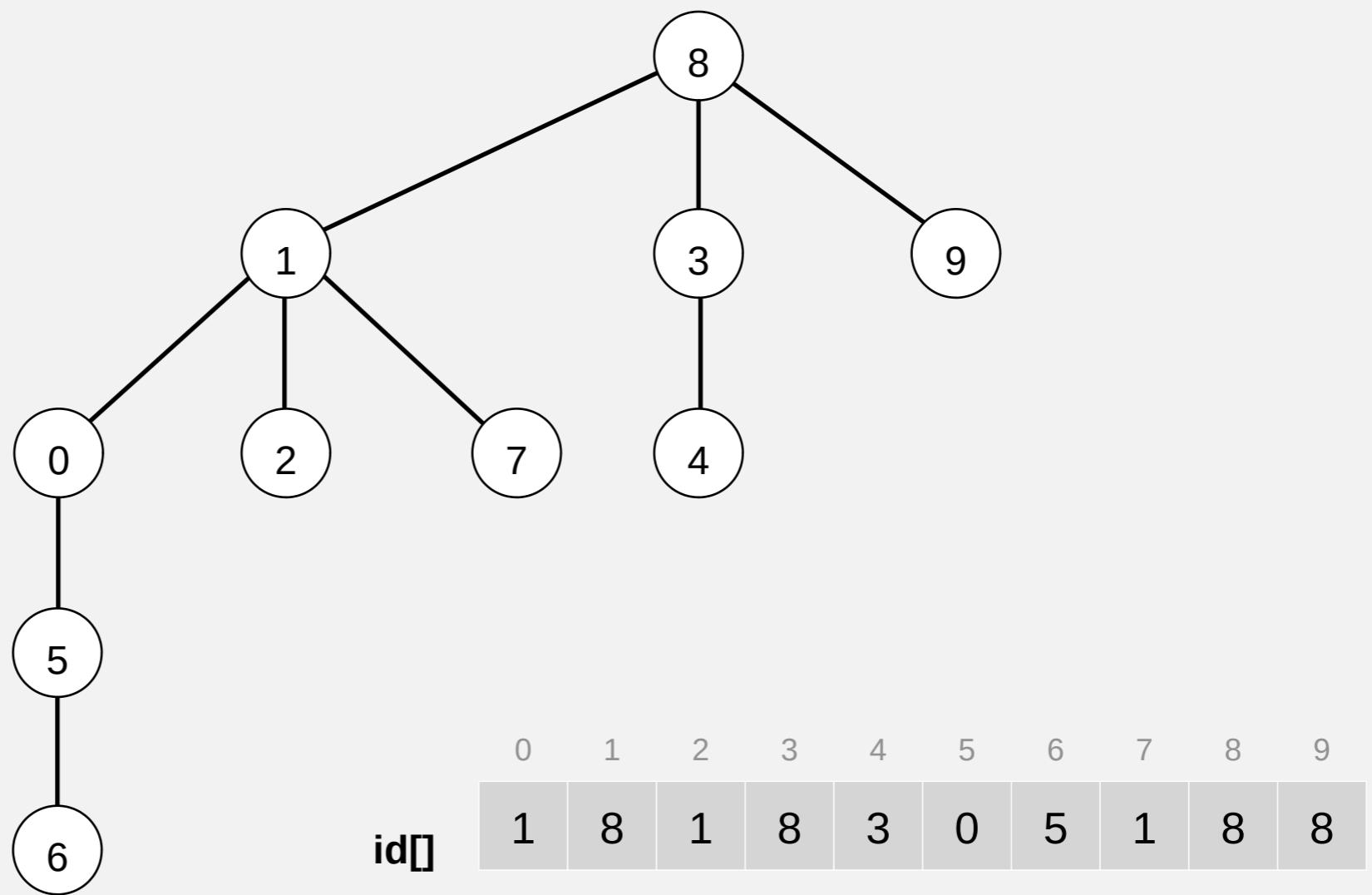
---

**union(7, 3)**



# Hurtig-forening demo

---



Live kode

# Quick-union is also too slow

Omkostningsmodel. Antal arrayadgange (læsning og skrivning).

algoritme	opret	forén	find	forbundne
<b>hurtig-find</b>	N	N	1	1
<b>hurtig-forening</b>	N	N <sup>†</sup>	N	N

← værste tilfælde

† inkluderer omkostning af at finde rødder

## Hurtig-find defekt.

- Forening er for dyr ( $N$  arrayadgange).
- Træerne er flade, men det er dyrt at holde dem flade.

## Hurtig-forening defekt.

- Træerne kan blive ret høje.
- Find (og dermed også forén) er for dyr (op til  $N$  arrayadgange).

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 1.5 FORÉN-OG-FIND

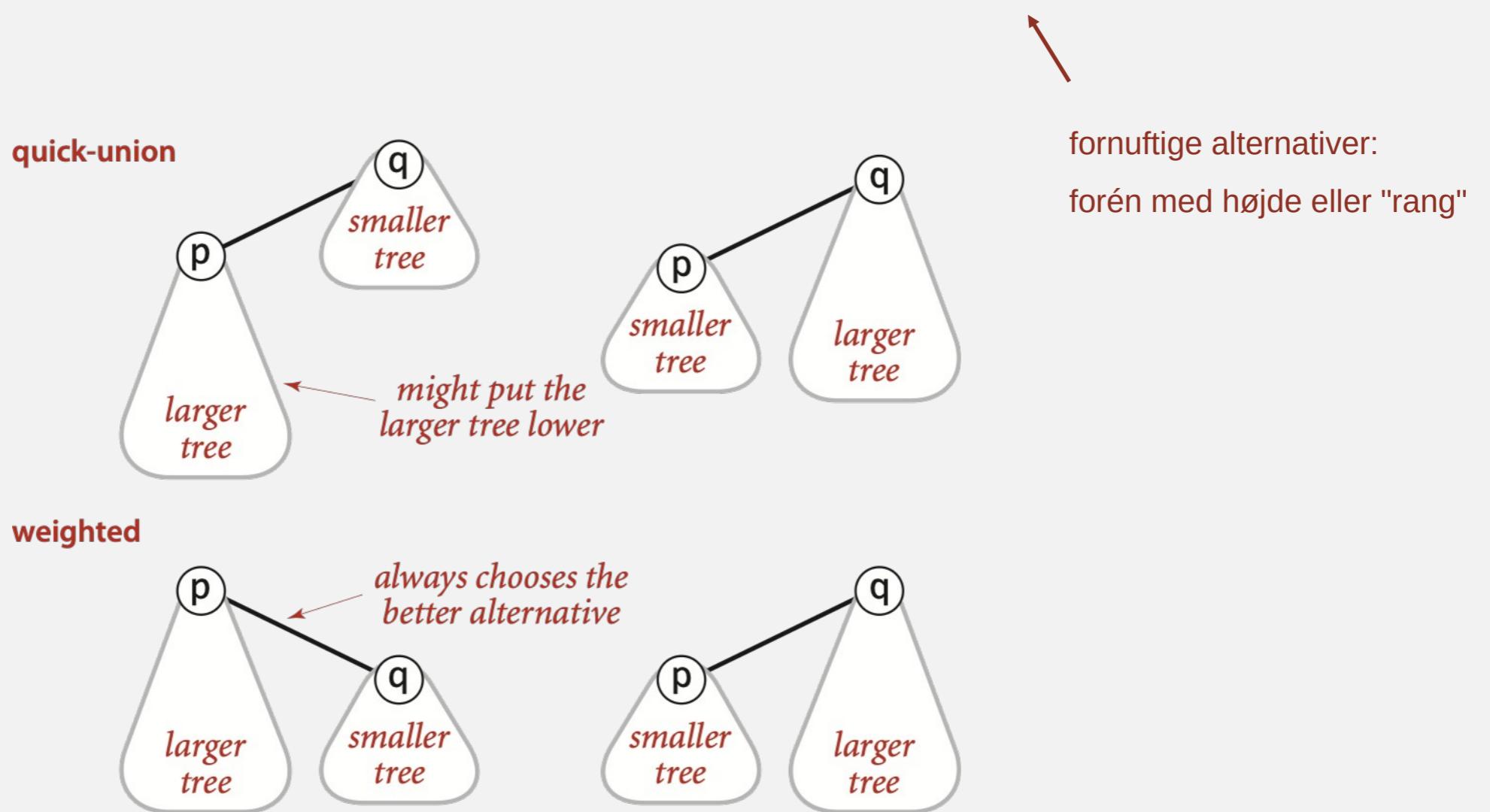
---

- ▶ *dynamisk sammenhæng*
- ▶ *hurtig find*
- ▶ *hurtig forening*
- ▶ ***forbedringer***
- ▶ *anvendelser*

# Forbedring 1: Vægtning

## Vægtet forén-og-find.

- Opdatér hurtig-forening så den undgår høje træer.
- Hold styr på størrelsen af hvert træ (antal elementer).
- Balancér ved at forbinde roden af det mindre træ til roden af det større træ.



# Weighted quick-union demo

---



	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Weighted quick-union demo

---

**union(4, 3)**

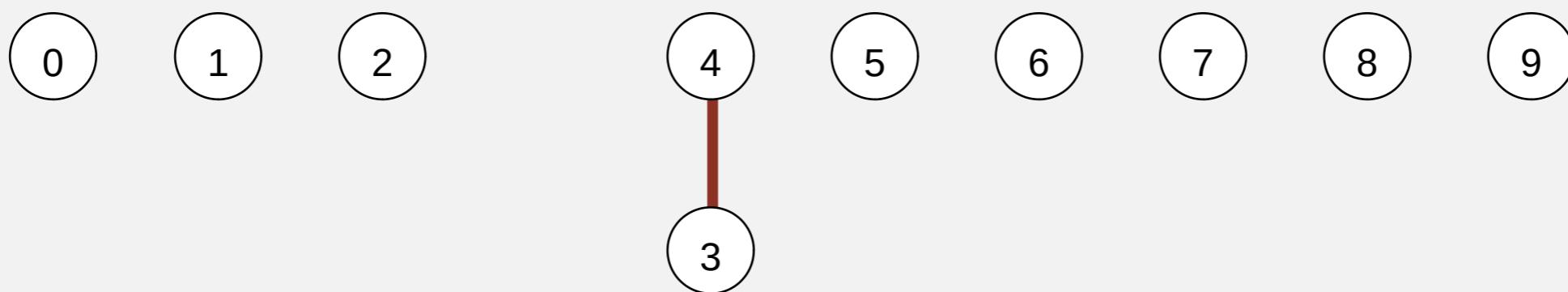


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	3	4	5	6	7	8	9

# Weighted quick-union demo

---

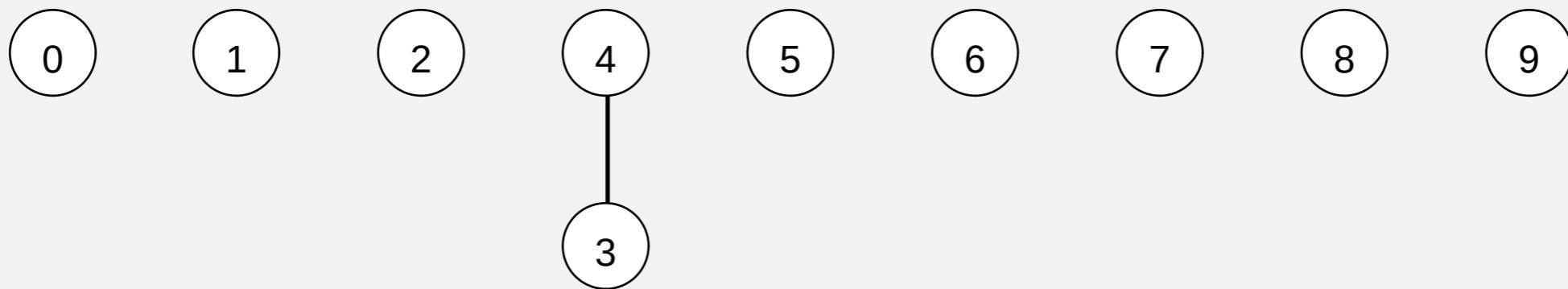
**union(4, 3)**



id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union demo

---

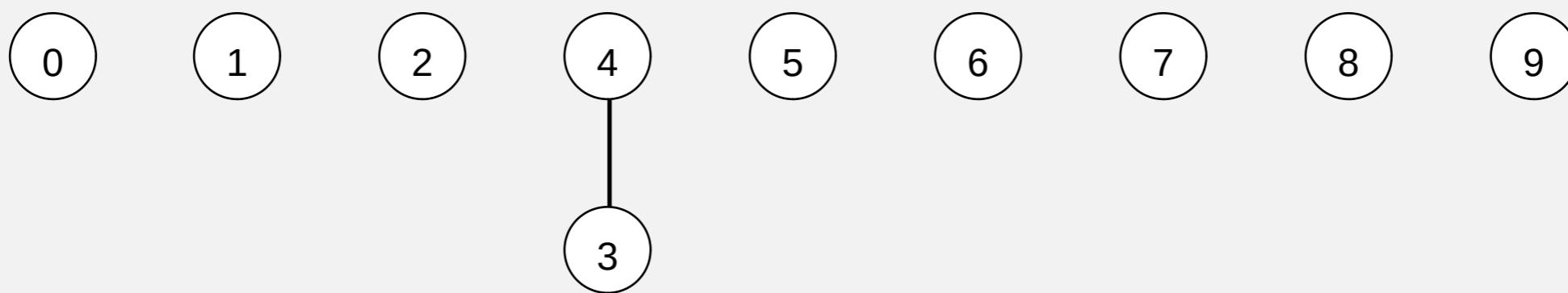


	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union demo

---

**union(3, 8)**



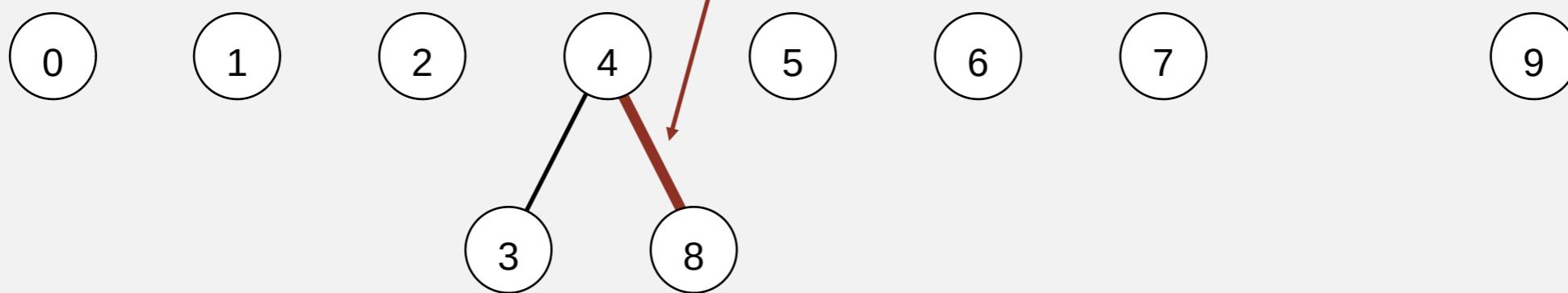
	0	1	2	3	4	5	6	7	8	9
<b>id[]</b>	0	1	2	4	4	5	6	7	8	9

# Weighted quick-union demo

---

**union(3, 8)**

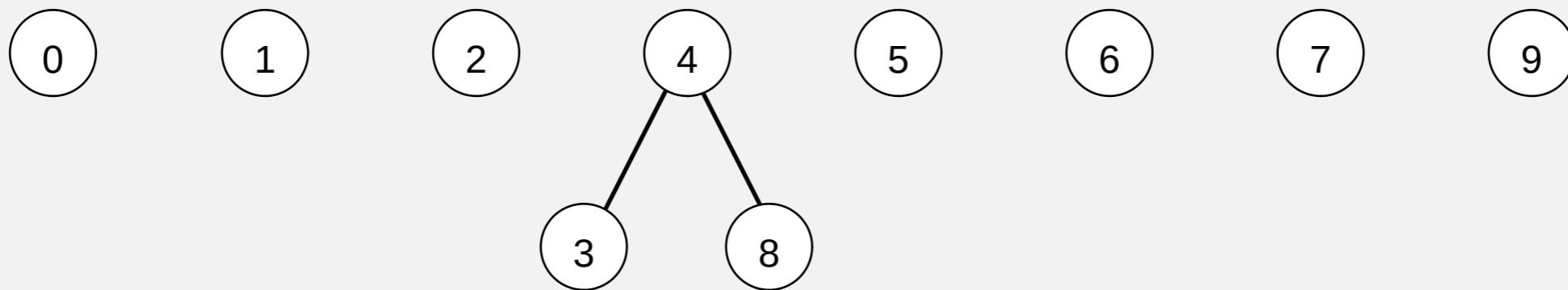
weighting: make 8 point to 4 (instead of 4 to 8)



id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union demo

---

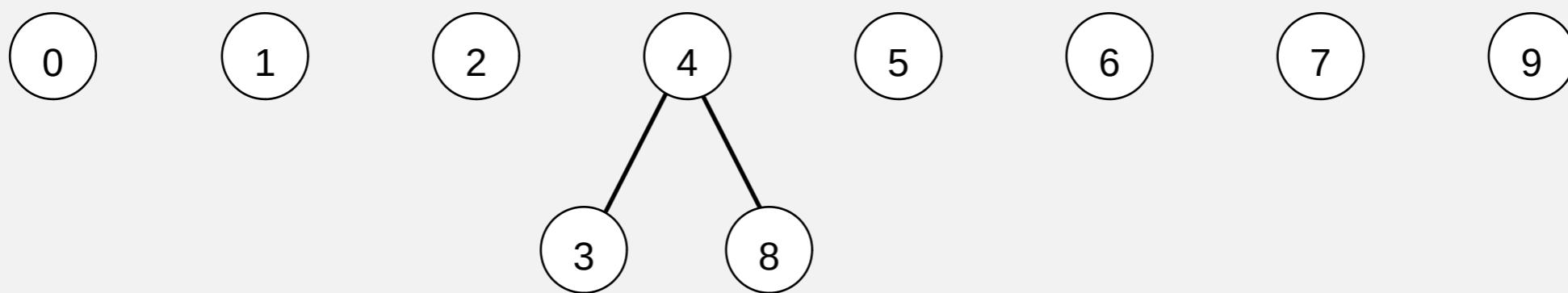


id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union demo

---

**union(6, 5)**

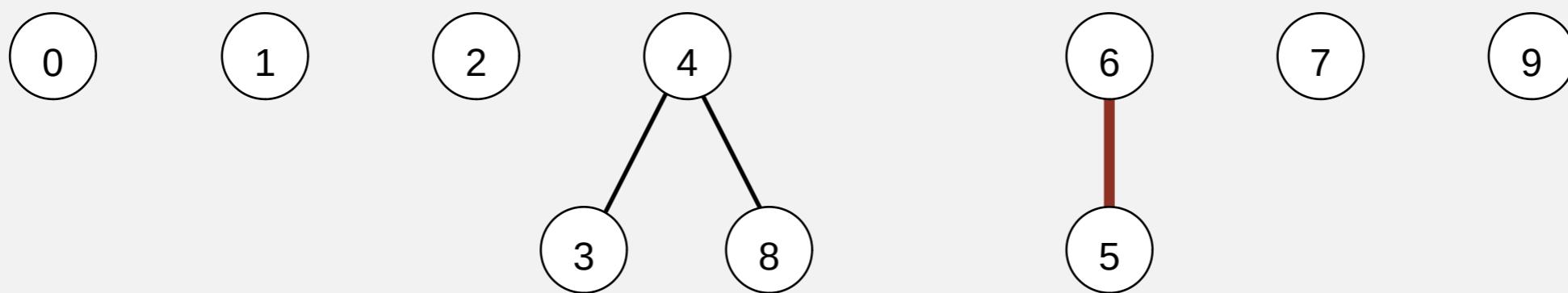


id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	5	6	7	4	9

# Weighted quick-union demo

---

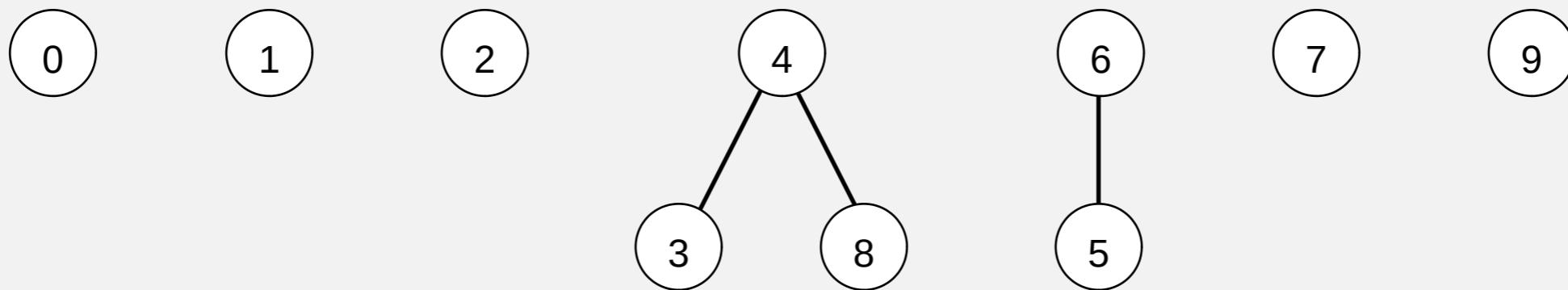
**union(6, 5)**



id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union demo

---

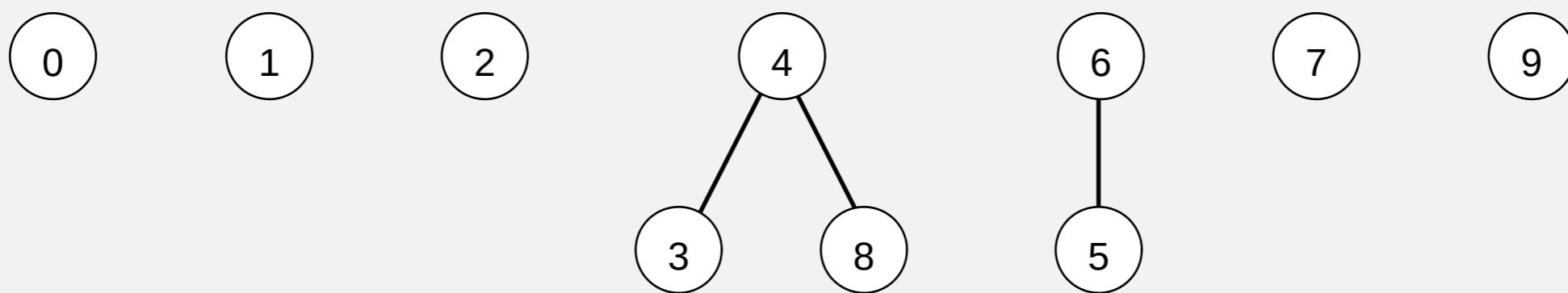


id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union demo

---

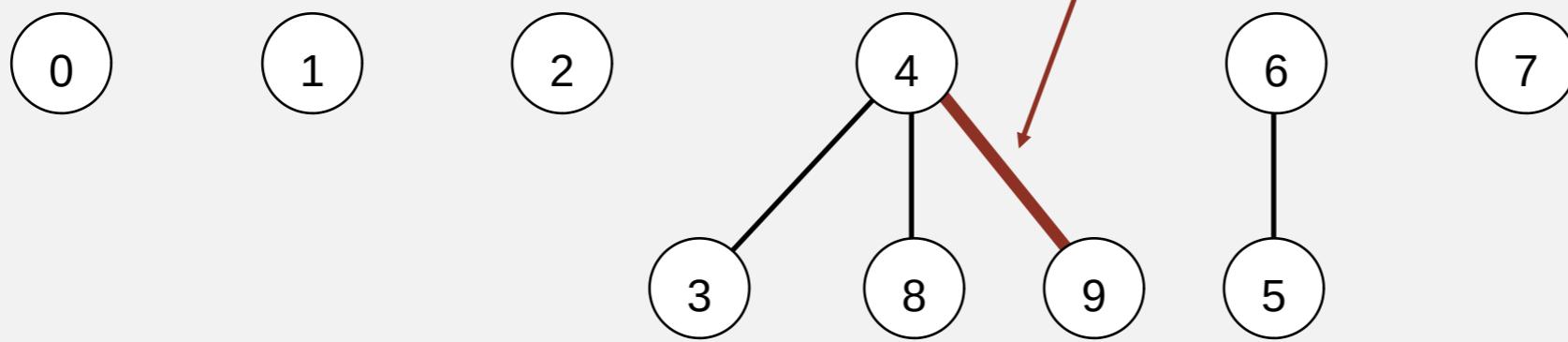
**union(9, 4)**



id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	6	6	7	4	9

# Weighted quick-union demo

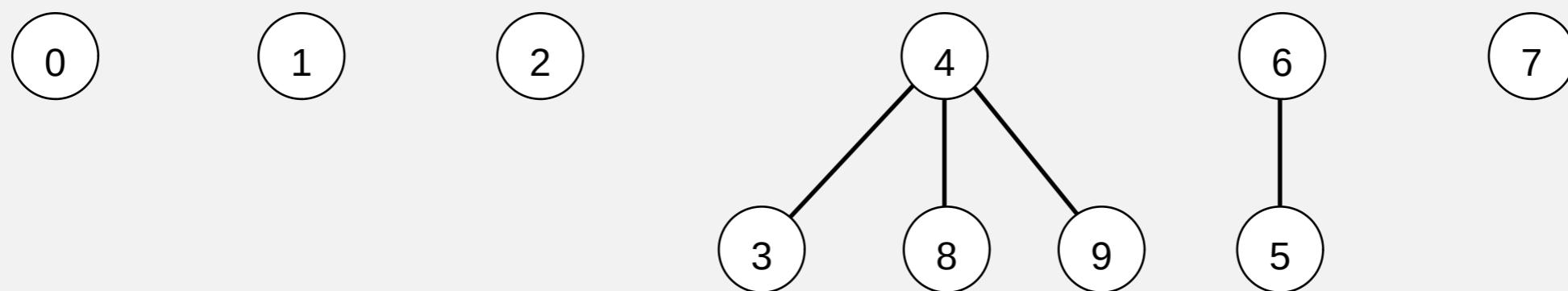
union(9, 4)  
weighting: make 9 point to 4



0	1	2	3	4	4	6	6	7	4	4
id[]	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

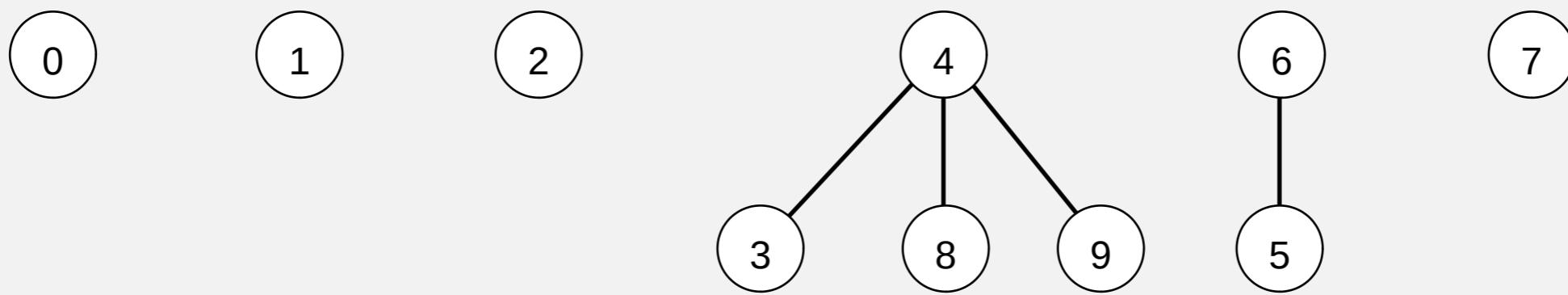


id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

**union(2, 1)**

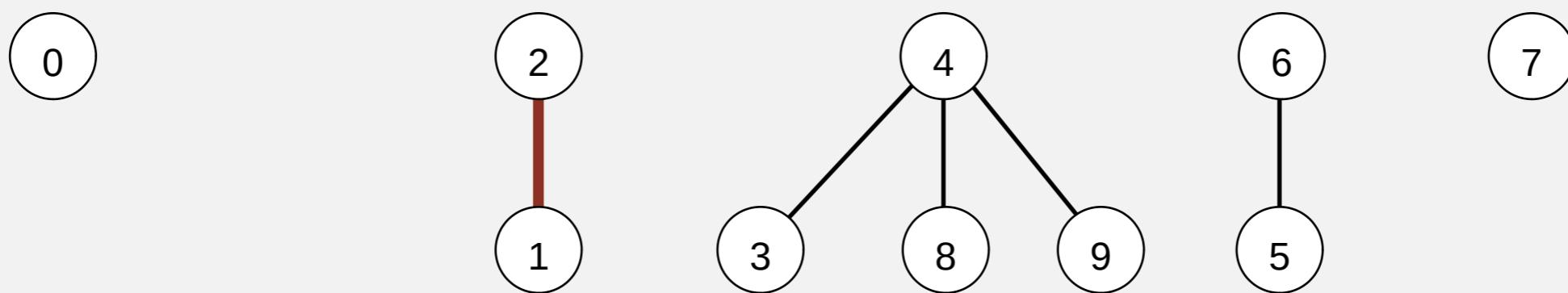


id[]	0	1	2	3	4	5	6	7	8	9
	0	1	2	4	4	6	6	7	4	4

# Weighted quick-union demo

---

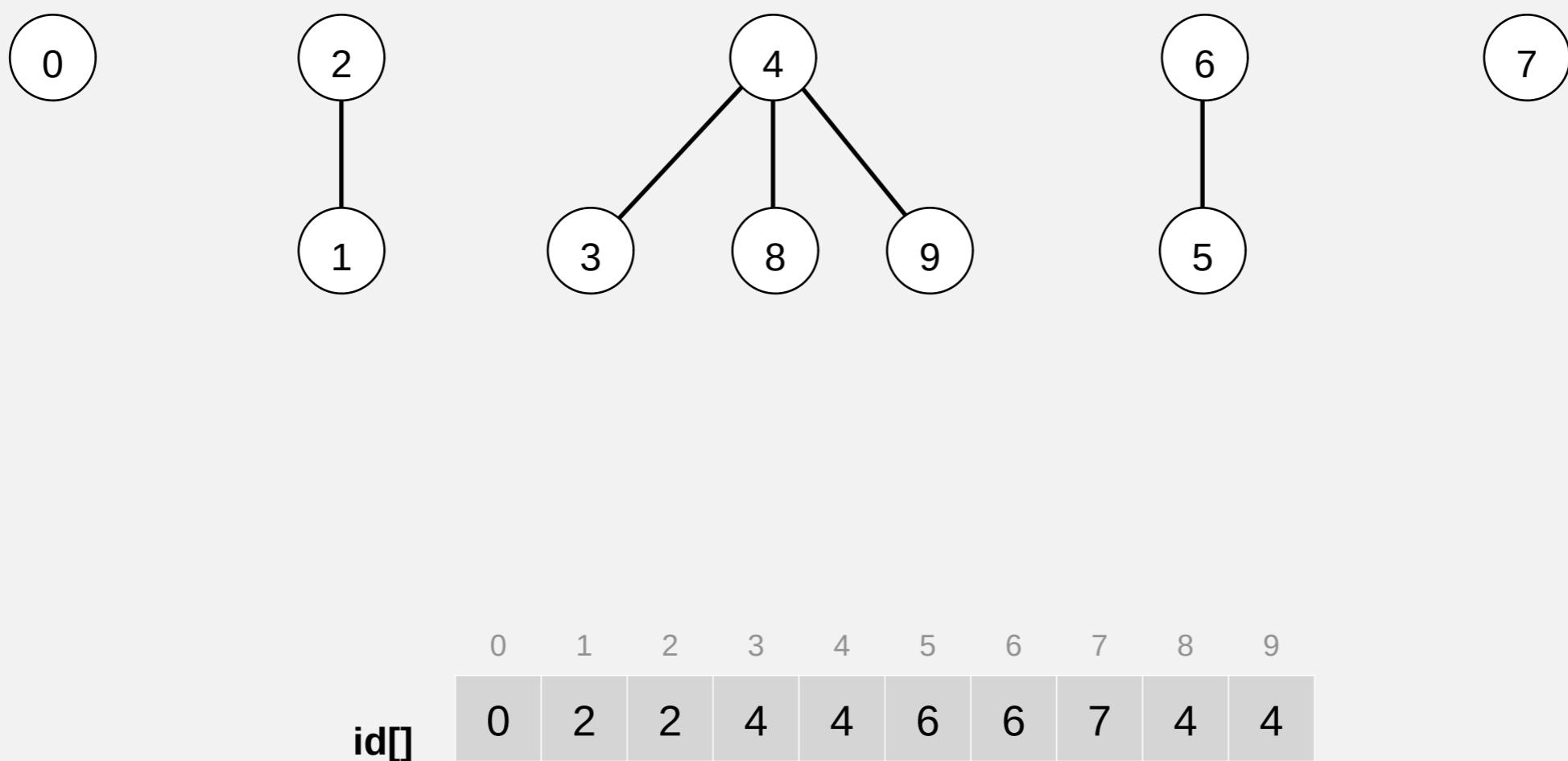
**union(2, 1)**



id[]	0	1	2	3	4	5	6	7	8	9
	0	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

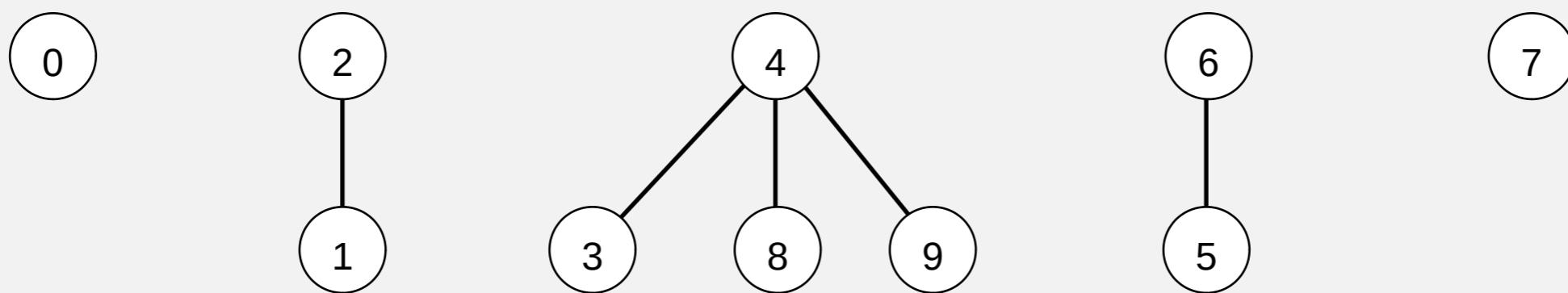
---



# Weighted quick-union demo

---

**union(5, 0)**



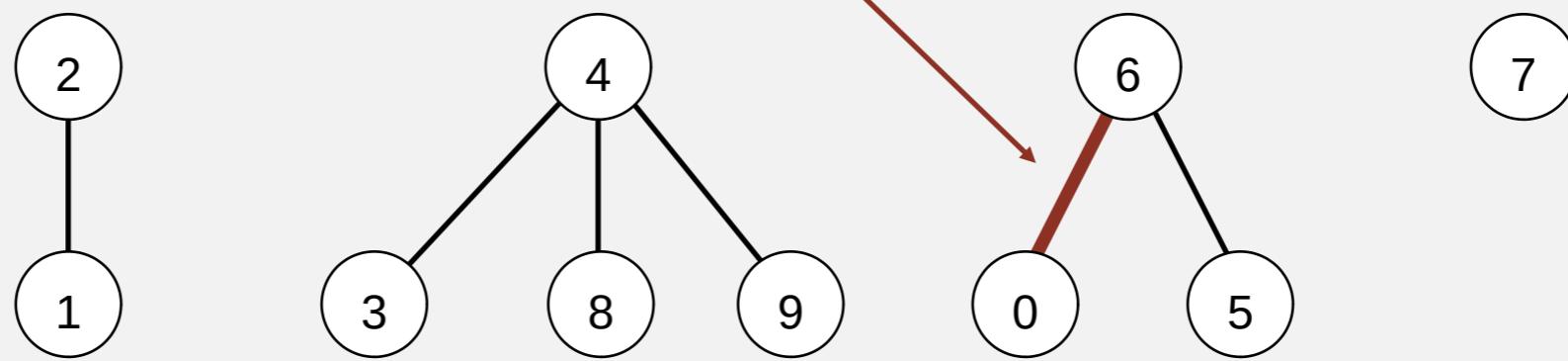
0	1	2	3	4	5	6	7	8	9
0	2	2	4	4	6	6	7	4	4

**id[]**

# Weighted quick-union demo

**union(5, 0)**

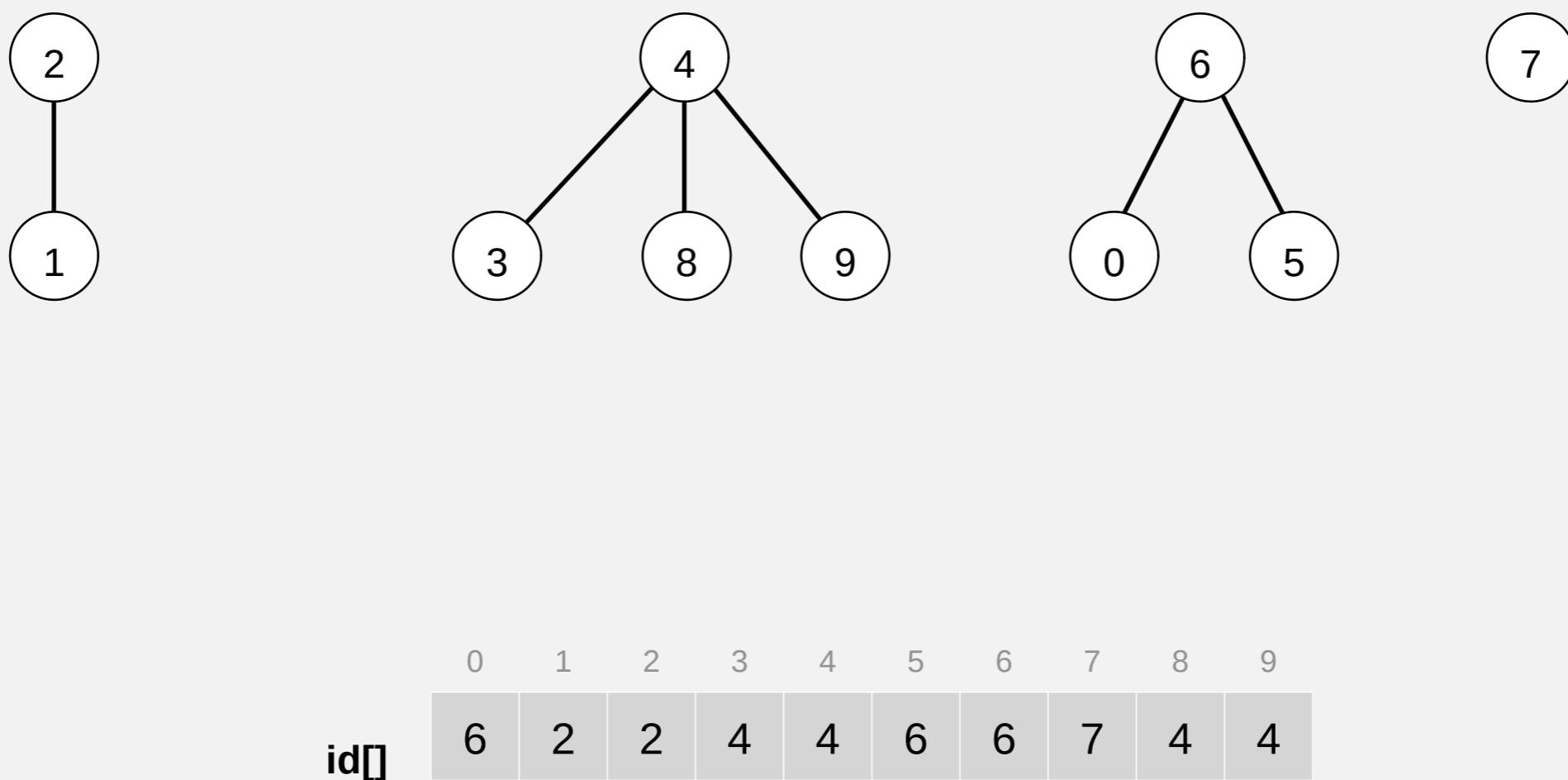
weighting: make 0 point to 6 (instead of 6 to 0)



0	1	2	3	4	5	6	7	8	9
<b>6</b>	2	2	4	4	6	6	7	4	4

# Weighted quick-union demo

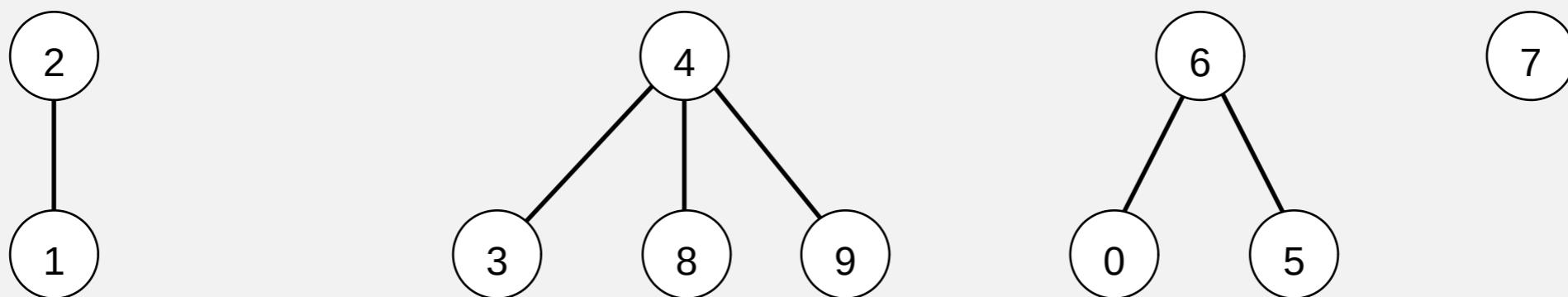
---



# Weighted quick-union demo

---

**union(7, 2)**

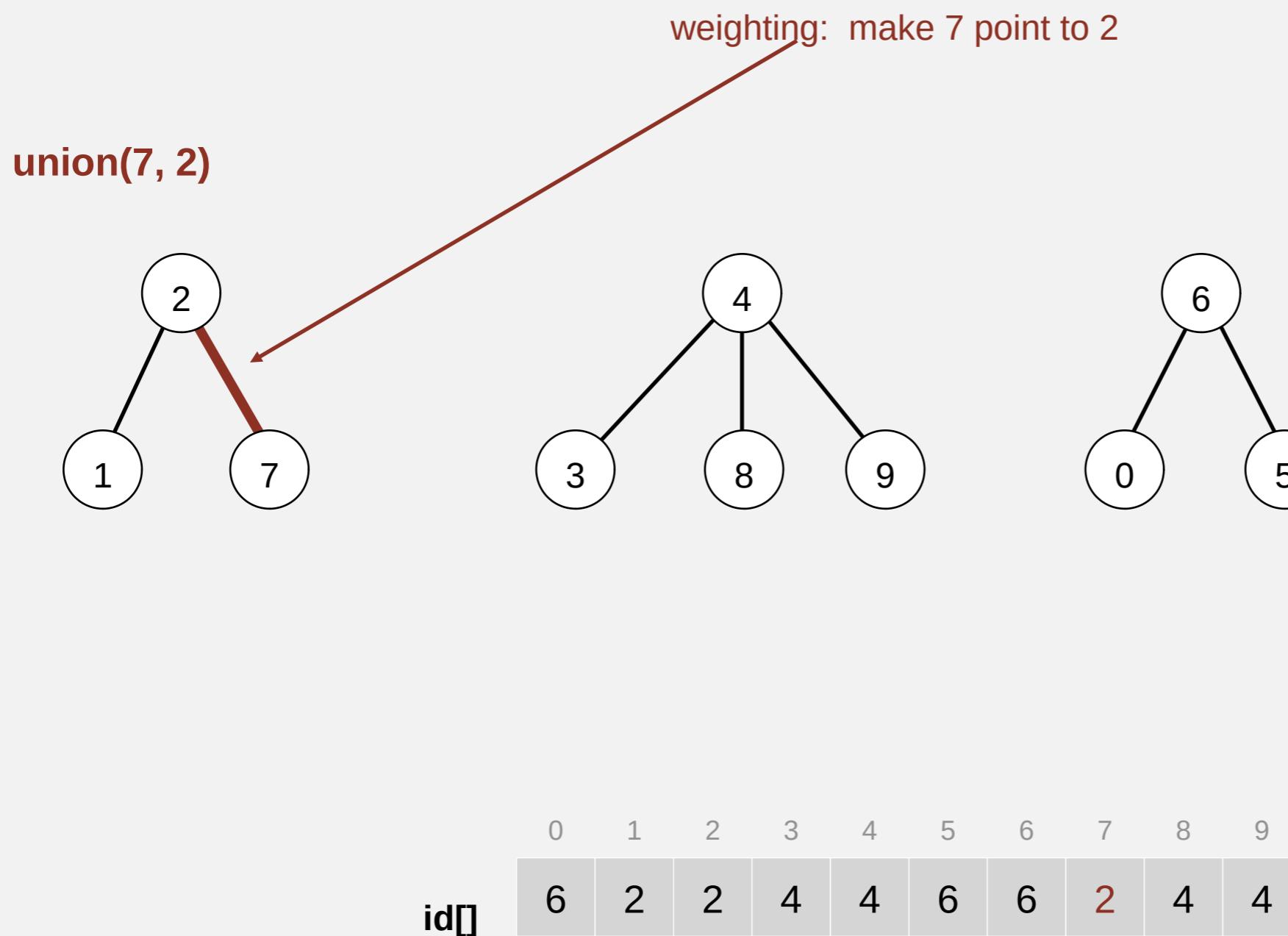


0	1	2	3	4	5	6	7	8	9
6	2	2	4	4	6	6	7	4	4

**id[]**

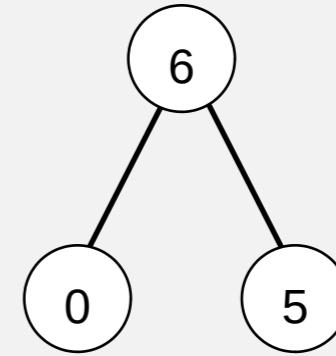
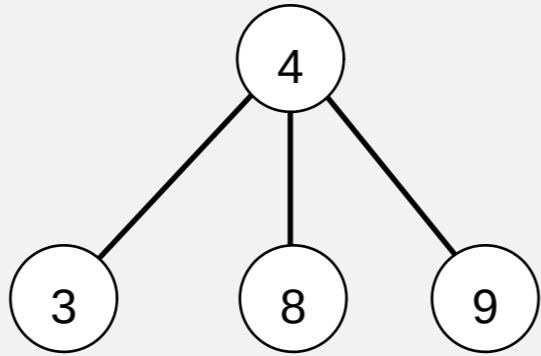
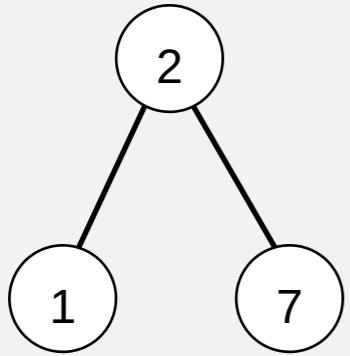
# Weighted quick-union demo

---



# Weighted quick-union demo

---

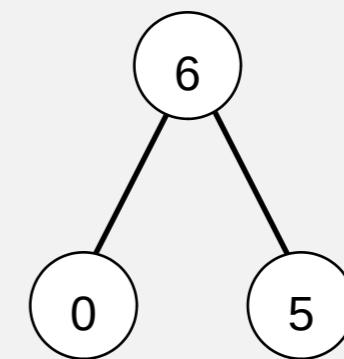
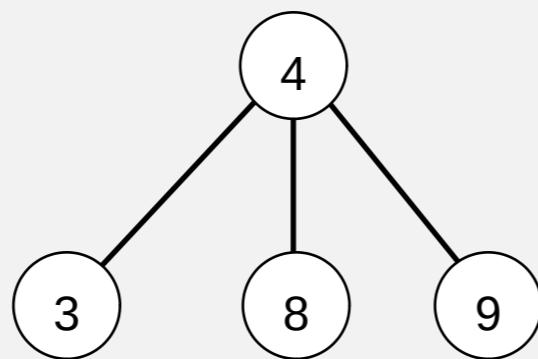
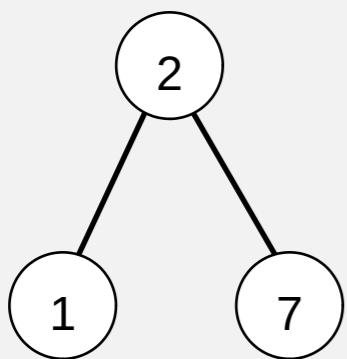


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union demo

---

**union(6, 1)**

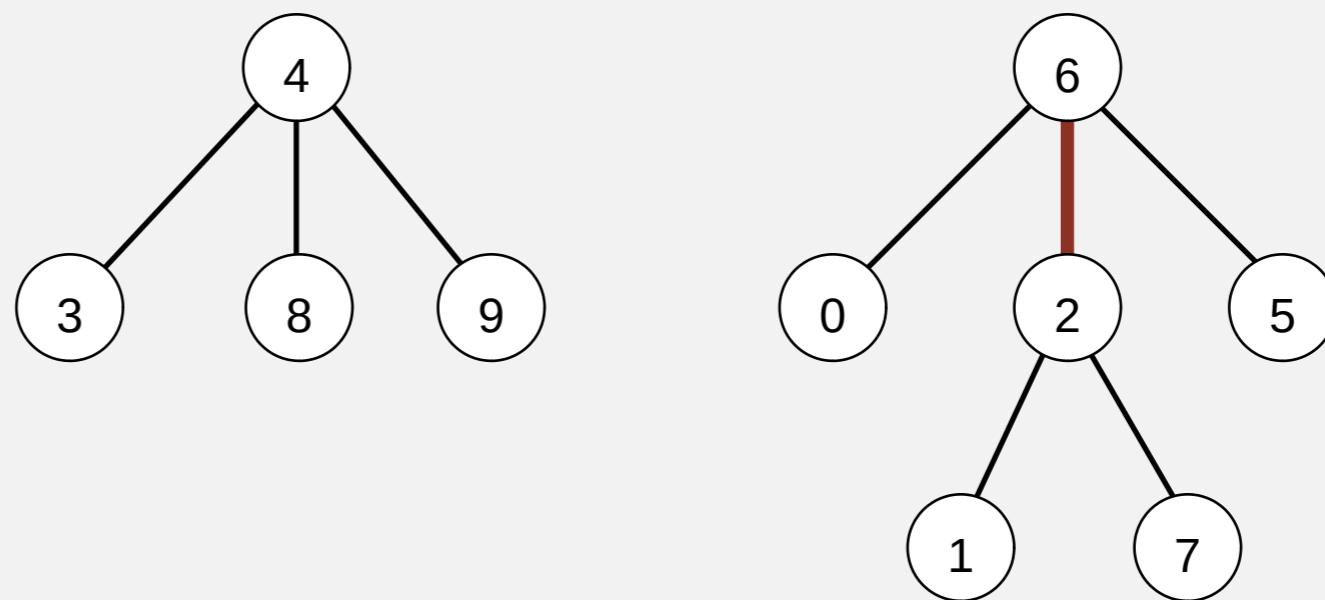


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	2	4	4	6	6	2	4	4

# Weighted quick-union demo

---

**union(6, 1)**

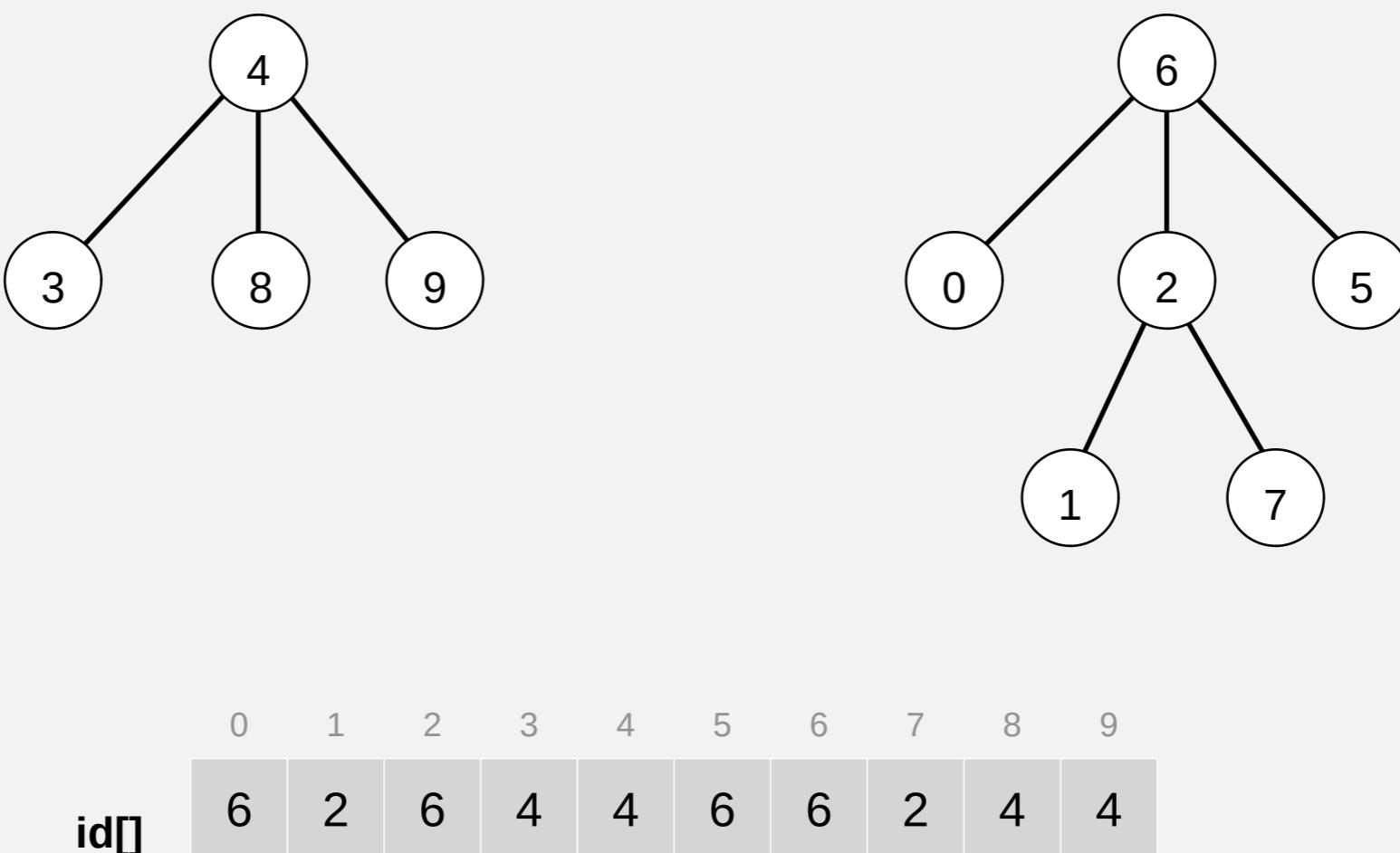


0	1	2	3	4	5	6	7	8	9
6	2	6	4	4	6	6	2	4	4

**id[]**

# Weighted quick-union demo

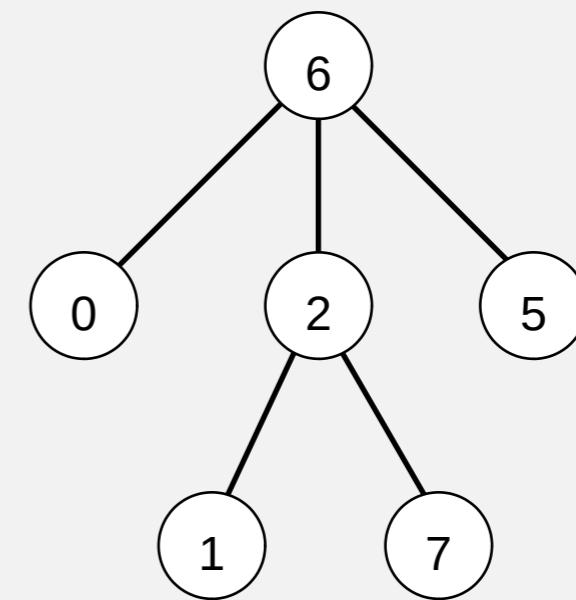
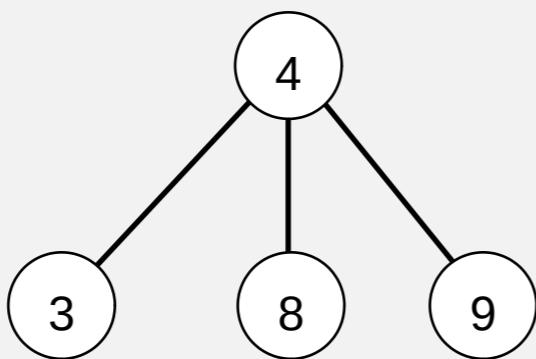
---



# Weighted quick-union demo

---

**union(7, 3)**



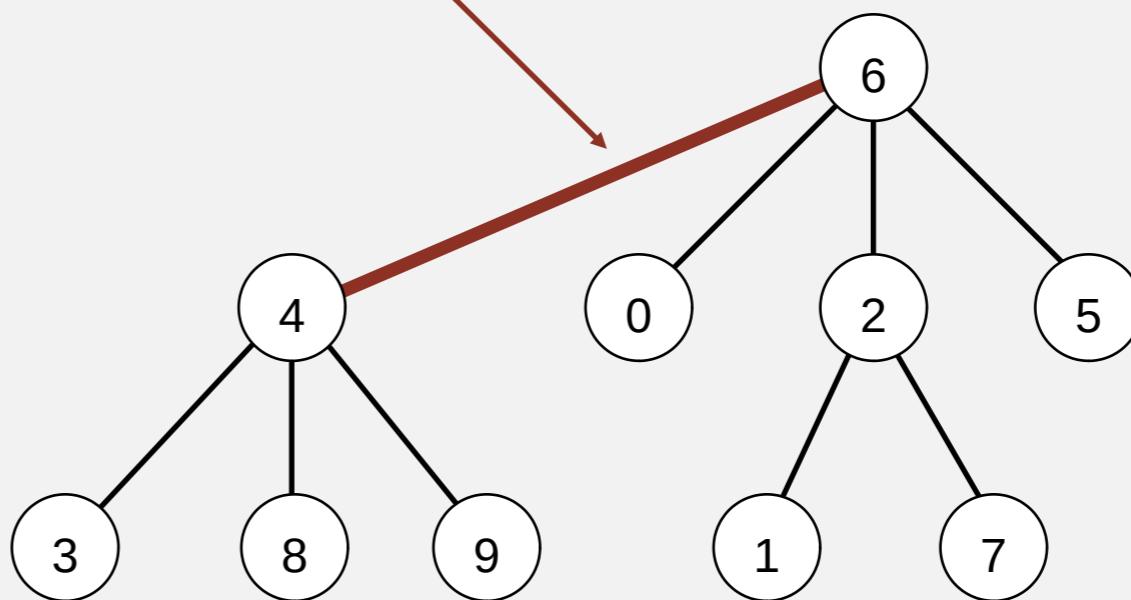
0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	4	6	6	2	4	4

# Weighted quick-union demo

---

union(7, 3)

weighting: make 4 point to 6 (instead of 6 to 4)

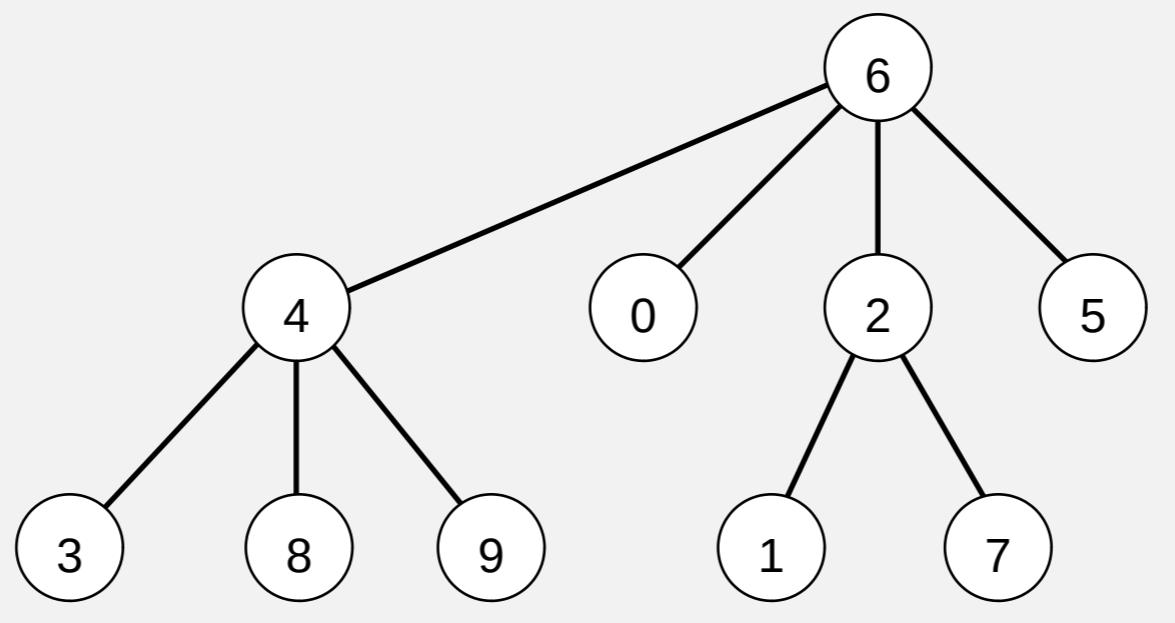


0	1	2	3	4	5	6	7	8	9
6	2	6	4	6	6	6	2	4	4

**id[]**

# Weighted quick-union demo

---

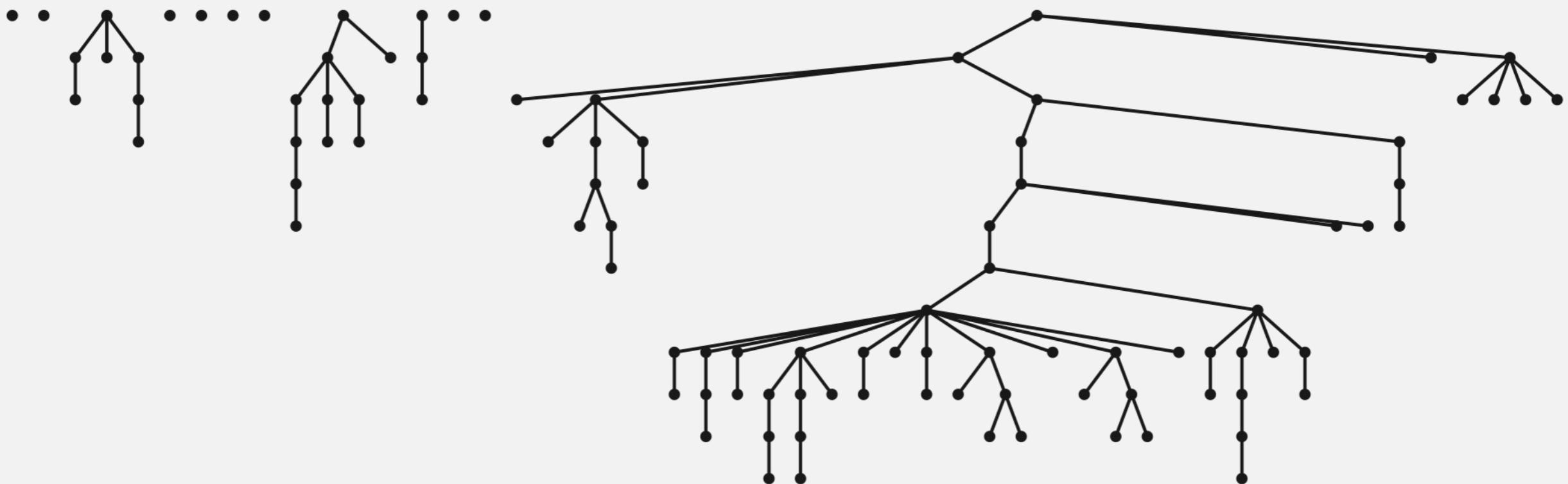


0	1	2	3	4	5	6	7	8	9	
<b>id[]</b>	6	2	6	4	6	6	6	2	4	4

# Hurtig-forening, med og uden vægtning

---

quick-union



*average distance to root: 5.11*

weighted



*average distance to root: 1.52*

Quick-union and weighted quick-union (100 sites, 88 union() operations)

# Vægtet hurtig-forening: Java implementation

---

**Datastruktur.** Samme som hurtig-forening, med et extra array  $sz[i]$  til at holde styr på antal elementer i træet der har element  $i$  som rod.

**Find/forbundet.** Samme som hurtig-forening.

**Forén.** Ændr hurtig-forening så:

- Det mindre træ blive forbundet til det større træ.
- At  $sz[]$  arrayet bliver opdateret for den nye rod.

```
int i = find(p);
int j = find(q);
if (i == j) return;
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else               { id[j] = i; sz[i] += sz[j]; }
```

Live kode

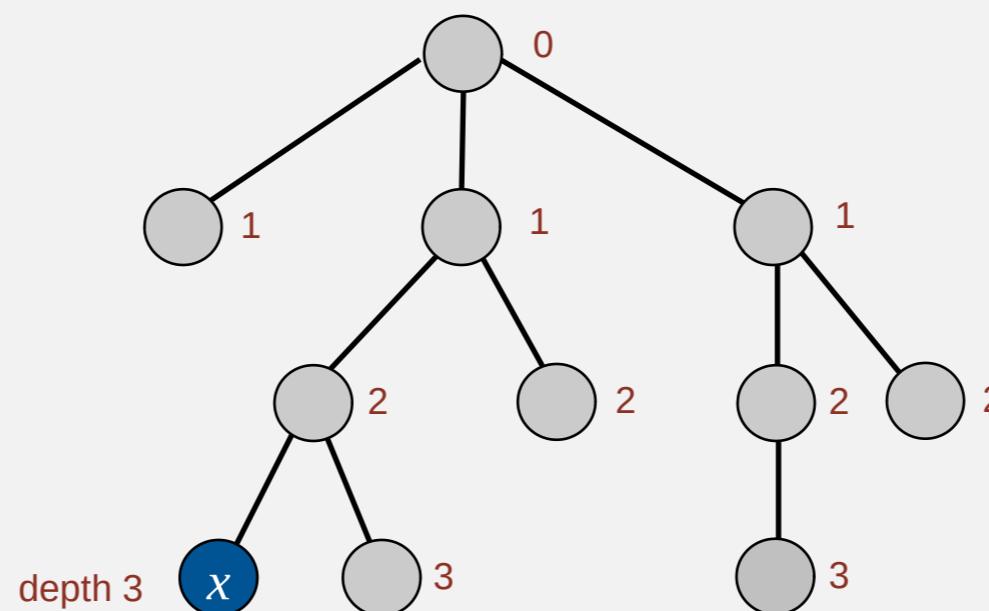
# Analyse af vægtet hurtig-forening

## Udførelstid.

- Find: tager tid proportionalt med dybden af  $p$ .
- Forén: tager konstant time, hvis vi har rødderne.

Sætning. Dybden af enhver knude  $x$  er højst  $\lg N$ .

$\lg = 2$ -tals logaritmen



$$N = 11$$

$$\text{depth}(x) = 3 \leq \lg N$$

# Analyse af vægtet hurtig-forening

## Udførselstid.

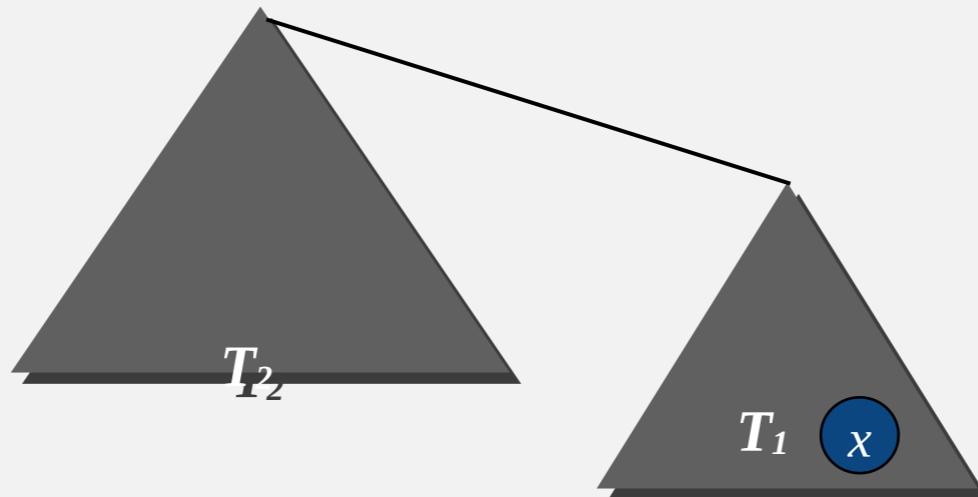
- Find: tager tid proportionalt med dybden af  $p$ .
- Forén: tager konstant time, hvis vi har rødderne.

Sætning. Dybden af enhver knude  $x$  er højst  $\lg N$ .

Bevis. Hvad får dybden for en knude  $x$  to stige?

Stiger med 1 når træ  $T_1$  (der indeholder  $x$ ) bliver forenet med et andet træ  $T_2$ .

- Foreningen, der indeholder  $x$ , bliver mindst fordoblet, da  $|T_2| \geq |T_1|$ .
- Størrelsen af træet, der indeholder  $x$ , kan fordoblet højst  $\lg N$  gange. Hvorfor?



1
2
4
8
16
:
N

$\lg N$

$\lg = 2$ -tals logaritmen

# Analyse af vægtet hurtig-forening

---

## Udførselstid.

- Find: tager tid proportionalt med dybden af  $p$ .
- Forén: tager konstant time, hvis vi har rødderne.

Sætning. Dybden af enhver knude  $x$  er højst  $\lg N$ .

algoritme	opret	forén	find	forbundne
<b>hurtig-find</b>	N	N	1	1
<b>hurtig-forening</b>	N	$N^{\dagger}$	N	N
<b>vægtet HF</b>	N	$\lg N^{\dagger}$	$\lg N$	$\lg N$

† inklusiv omkosningen af at finde rødder

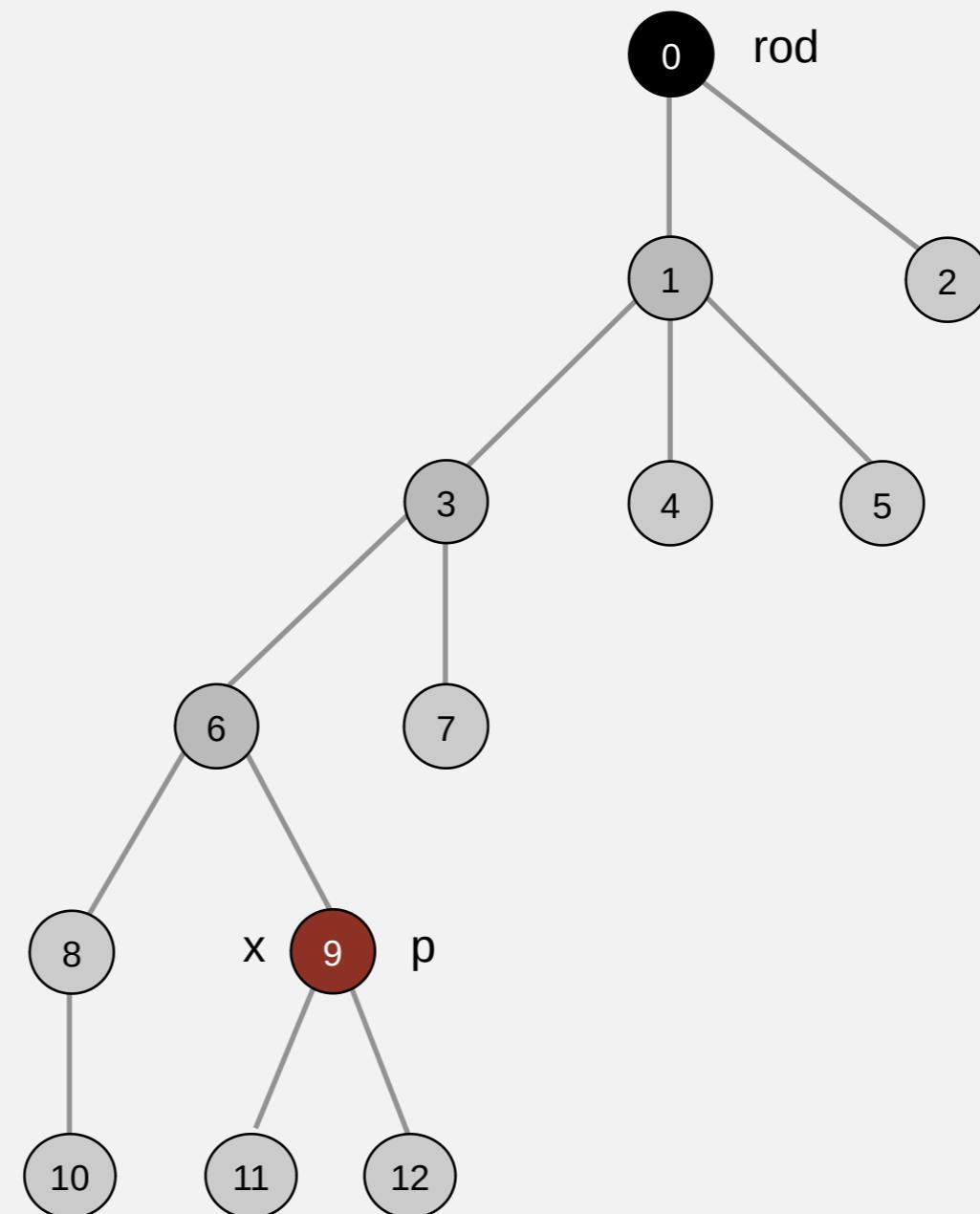
Q. Er vi så færdige, nu hvor vi har garanteret acceptabel udførselstid?

A. Nej, vi kan nemt forbedre endnu mere.

## Forbedring 2: Vejforkortning

---

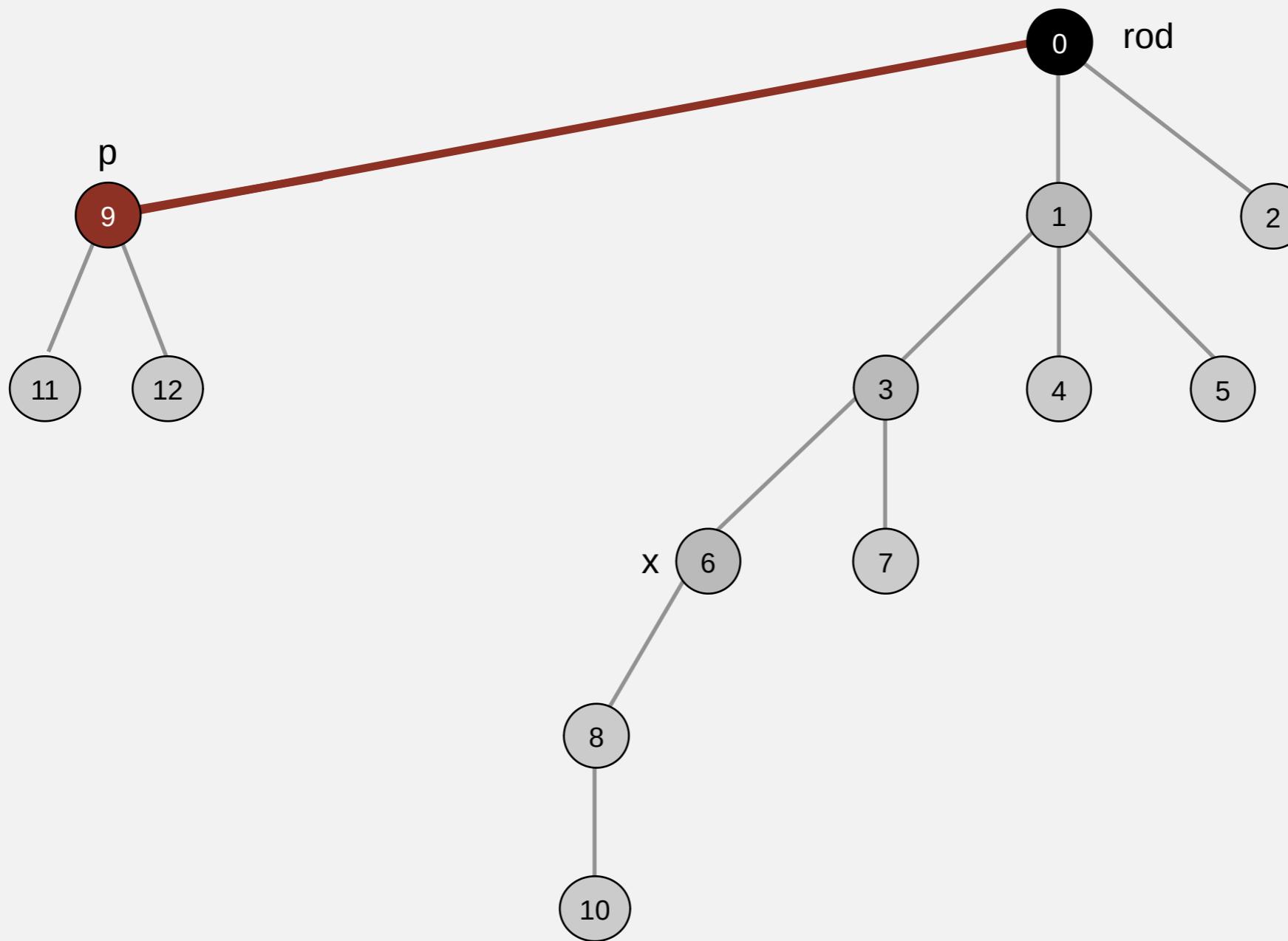
Hurtig-forening med vejforkortning. Efter vi har fundet roden af  $p$ , sæt  $\text{id}[]$  af hver besøgt knude til at pege på roden.



## Forbedring 2: Vejforkortning

---

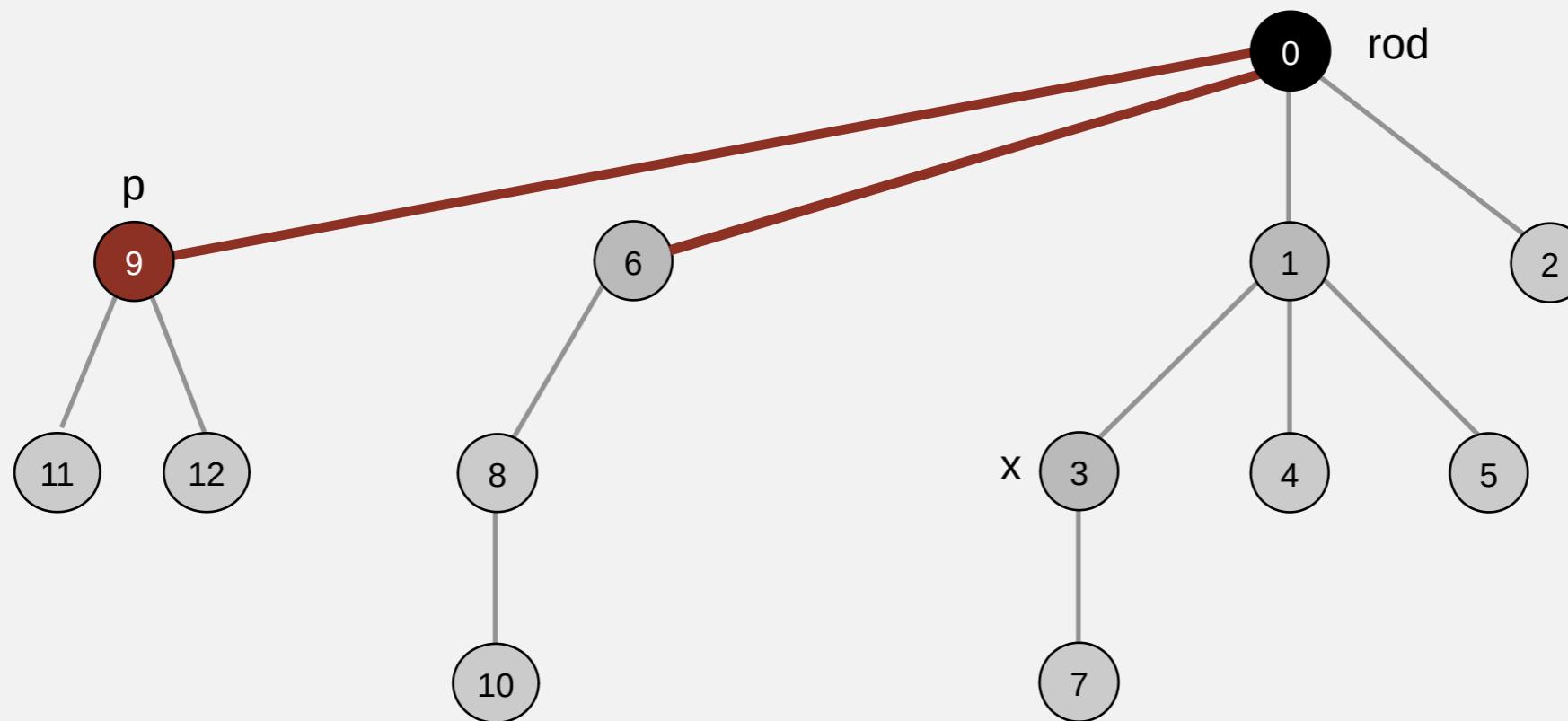
Hurtig-forening med vejforkortning. Efter vi har fundet roden af  $p$ , sæt  $\text{id}[]$  af hver besøgt knude til at pege på roden.



## Forbedring 2: Vejforkortning

---

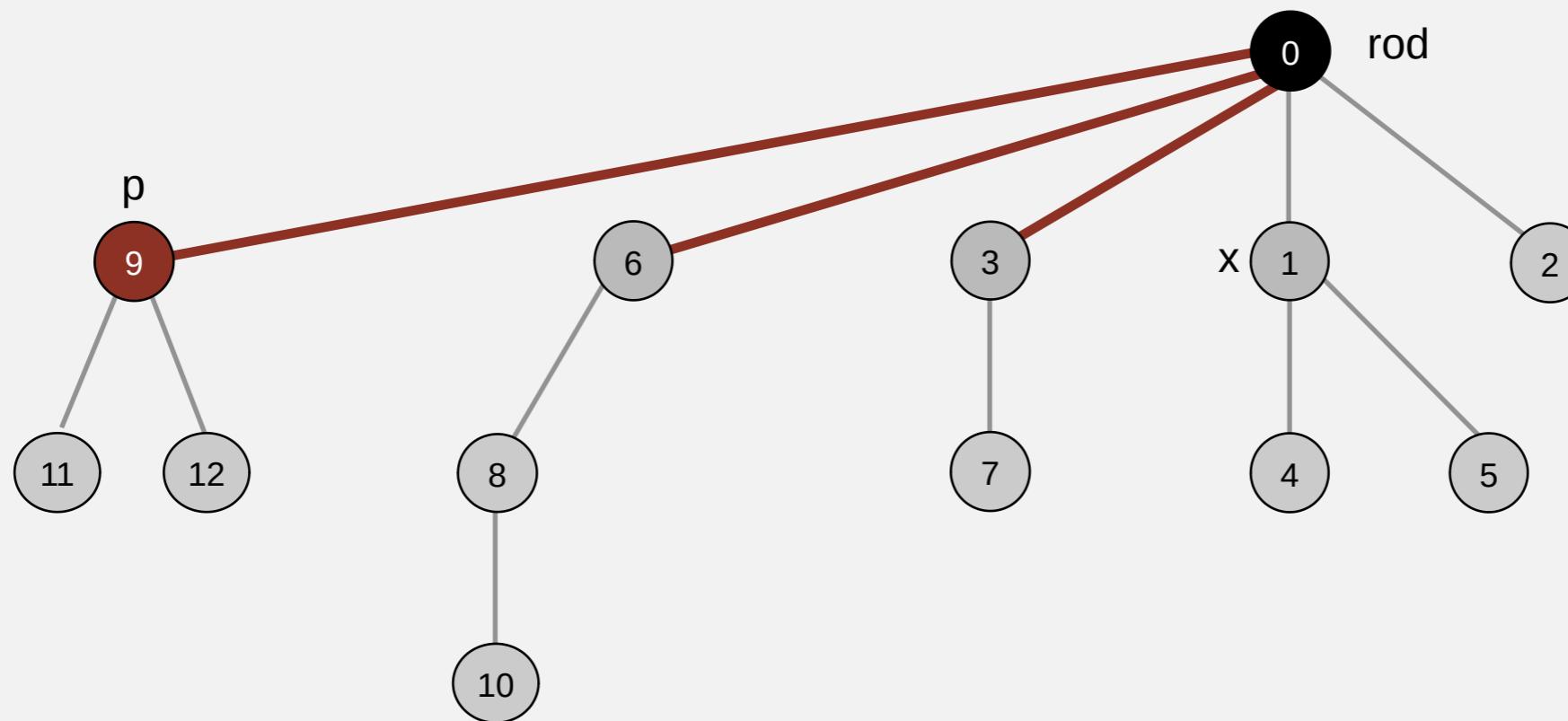
Hurtig-forening med vejforkortning. Efter vi har fundet roden af  $p$ , sæt  $\text{id}[]$  af hver besøgt knude til at pege på roden.



## Forbedring 2: Vejforkortning

---

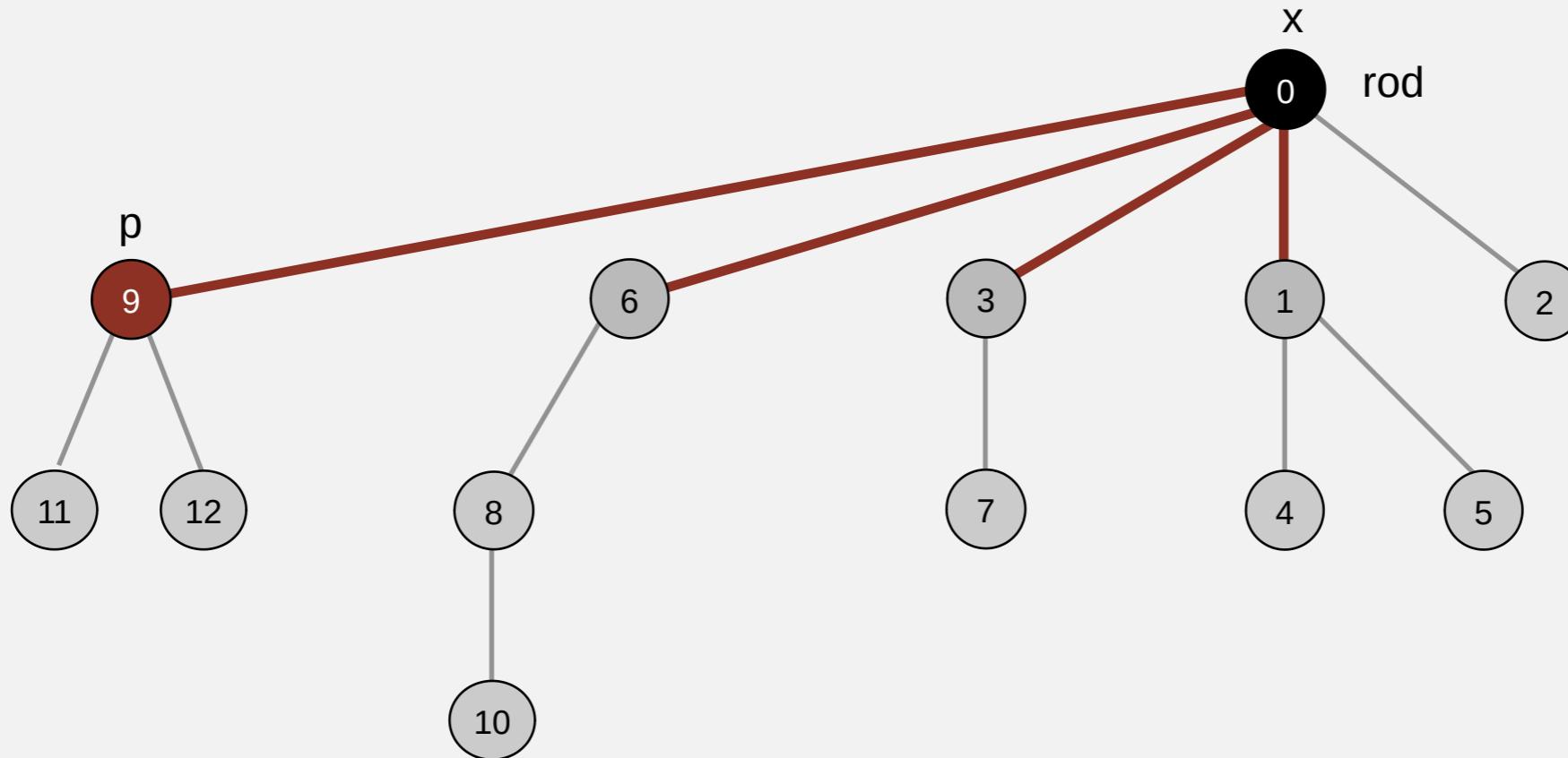
Hurtig-forening med vejforkortning. Efter vi har fundet roden af  $p$ , sæt  $\text{id}[]$  af hver besøgt knude til at pege på roden.



## Forbedring 2: Vejforkortning

---

Hurtig-forening med vejforkortning. Efter vi har fundet roden af  $p$ , sæt  $\text{id}[]$  af hver besøgt knude til at pege på roden.



Resultat. Nu har find() den bivirkning at flade træet ud.

## Vejforkortning: Java implementation

---

To-pas implementation: tilføj en løkke mere til find() som opdaterer id[]'erne for alle knuderne på vejen op til roden.

Enklere ét-pas variant (vejhalvering): Lad hver anden knude på vejen pege på dens bedsteforælder.

```
public int find(int i) {  
    while (i != id[i]) {  
        id[i] = id[id[i]]; ←———— kun én ekstra linje!  
        i = id[i];  
    }  
    return i;  
}
```

I praksis. Ingen grund til at lade være! Holder træet næsten helt fladt.

# Vægtet hurtig-forening med vejforkortning: amortiseret analyse

**Sætning.** [Hopcroft-Ulman, Tarjan] Hvis man starter med en tom datastruktur, så vil enhver sekvens af  $M$  forén-og-find ops på  $N$  elementer lave  $\leq c(N + M \lg^* N)$  arrayadgange.

- Analyse kan forbedres til  $N + M \alpha(M, N)$ .
- Simpel algoritme med fascinerende matematik.

$\lg^* N :=$  antallet af gange vi skal tage  $\lg$  for at nå ned til 1

N	$\lg^* N$
1	0
2	1
4	2
16	3
65536	4
$2^{65536}$	5

Lineær-tids algoritme for  $M$  forén-og-find ops på  $N$  elementer?

Ig\* funktion

- Omkostning indenfor en konstant faktor af bare at læse input.
- I teorien er VHFVF ikke helt lineær.
- I praksis er VHFVF så tæt på lineær at vi ikke kan se forskellen.

Overraskende resultat. [Fredman-Saks] Ingen lineærtids algoritme kan findes.

i "cell-probe" beregningsmodellen

# Opsummering

---

**Hovedpointe.** Vægtet hurtig-forening (og/eller vejforkortning) gør det muligt at løse problemer som vi ellers ikke kunne.

algoritme	værste-tilfælde tid
<b>hurtig-find</b>	$M N$
<b>hurtig-forening</b>	$M N$
<b>vægtet hurtig-forening</b>	$N + M \log N$
<b>hurtig-forening + vejforkortning</b>	$N + M \log N$
<b>hele molevitten</b>	$N + M \lg^* N$

størrelsesorden for  $M$  foren-og-find operationer på  $N$  elementer

**Eks.** [10<sup>9</sup> foren-og-find ops på 10<sup>9</sup> elementer]

- VHFVF reducerer tiden fra 30 år til 6 sekunder.
- Supercomputerne hjælper ikke meget; gode algoritmer muliggører løsningen.

Hele molevitten: Java implementation

---

Live kode

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 1.5 FORÉN-OG-FIND

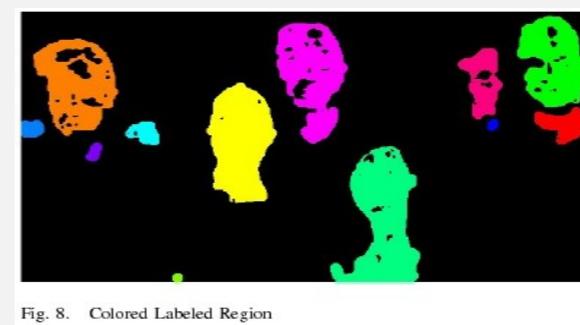
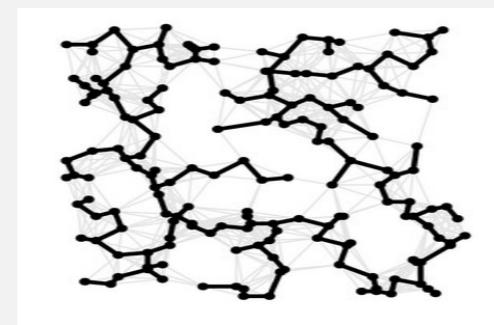
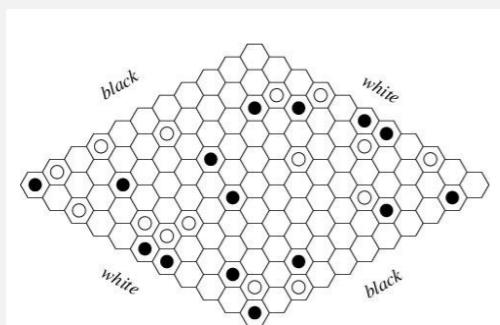
---

- ▶ *dynamisk sammenhæng*
- ▶ *hurtig find*
- ▶ *hurtig forening*
- ▶ *forbedringer*
- ▶ ***anvendelser***

# Forén-og-find anvendelser

---

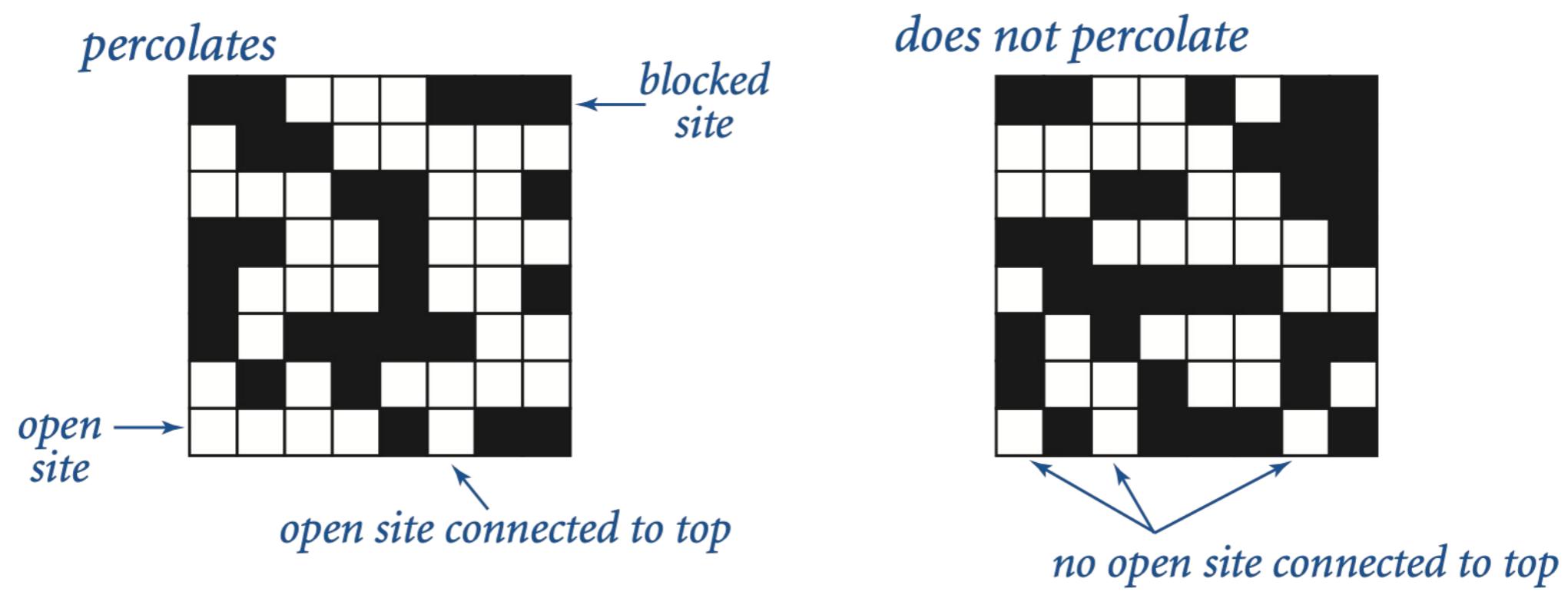
- Perkolation.
- Spil (Go, Hex).
- ✓ Dynamisk sammenhæng.
  - Least common ancestor.
  - Äkvivalens af endelige automater.
  - Hoshen-Kopelman algoritmen i fysik.
  - Hinley-Milner polymorphisk type inferens.
  - Kruskal's algortime til mindste udspændende træ.
  - Kompilering af ækvivalente udtryk i Fortran.
  - Matlab's bwlabel() funktion til billedanalyse.
  - Danmarks Statistikks definition af byområde.



# Perkolation

En abstrakt model for mange fysiske systemer:

- $N$ -gange- $N$  gitter af celler.
- Hver celle er åben med ss.  $p$  (og lukket med ss.  $1 - p$ ).
- Systemet **perkolerer** hviss top og bund er forbundne af åbne celler.



# Perkolation

---

En abstrakt model for mange fysiske systemer:

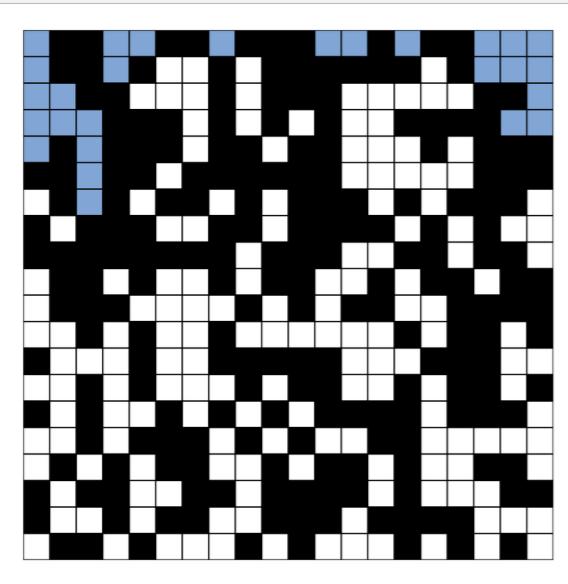
- $N$ -gange- $N$  gitter af celler.
- Hver celle er åben med ss.  $p$  (og lukket med ss.  $1 - p$ ).
- Systemet **perkolerer** hviss top og bund er forbundne af åbne celler.

model	system	åben celle	lukket celle	perkolerer
elektricitet	materiale	leder	isolator	leder
væskestørsm	materiale	tom	blokeret	porøs
social interaktion	befolkning	person	tom	kommunikerer

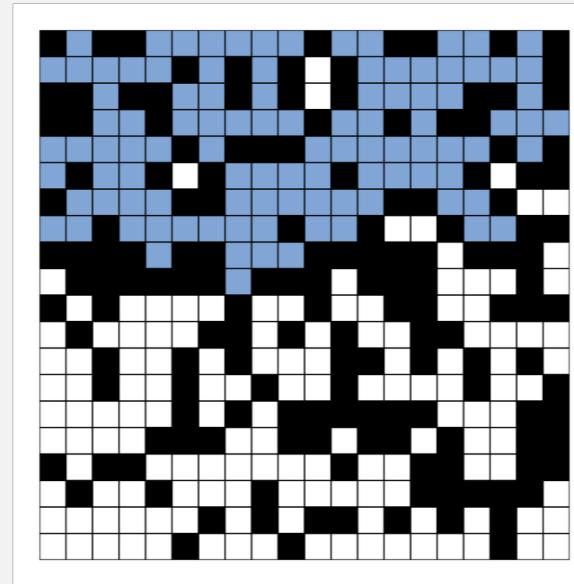
# Sandsynlighed for perkolation

---

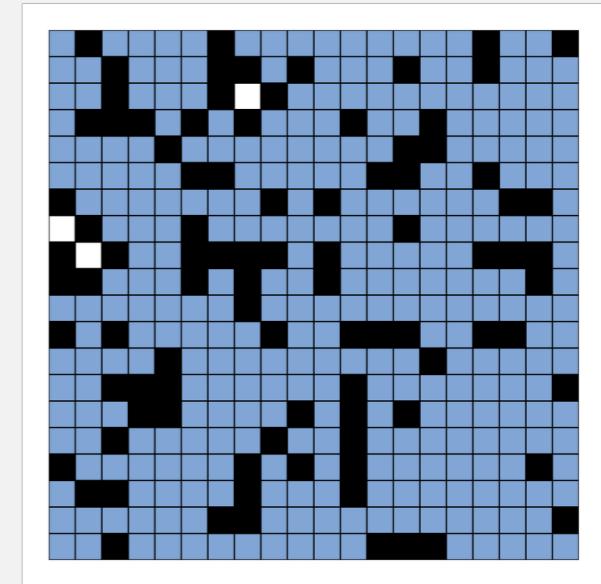
Afhænger af gitterstørrelse  $N$  og celle tomhedssandsynligheden  $p$ .



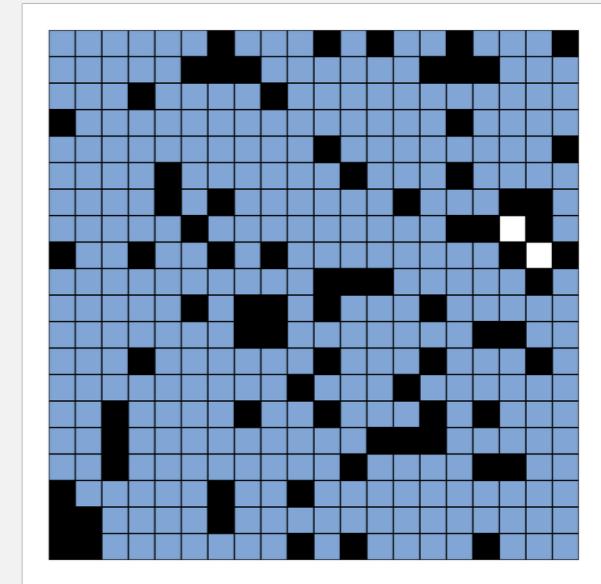
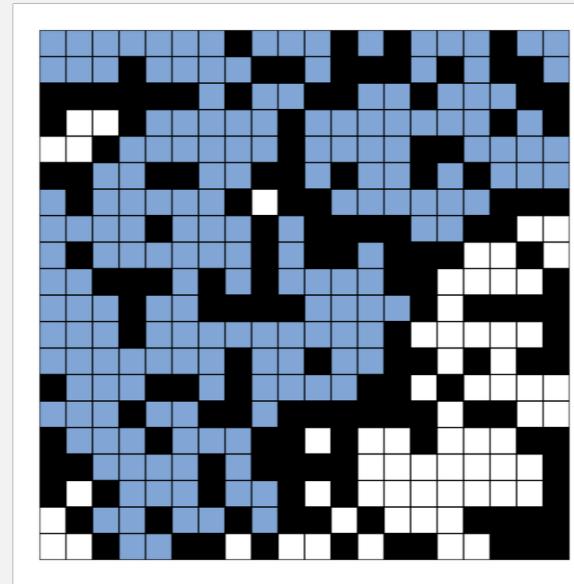
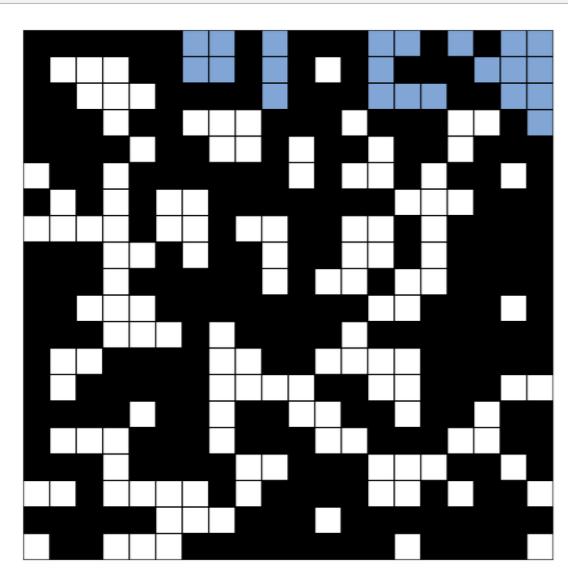
$p$  low (0.4)  
does not percolate



$p$  medium (0.6)  
percolates?



$p$  high (0.8)  
percolates

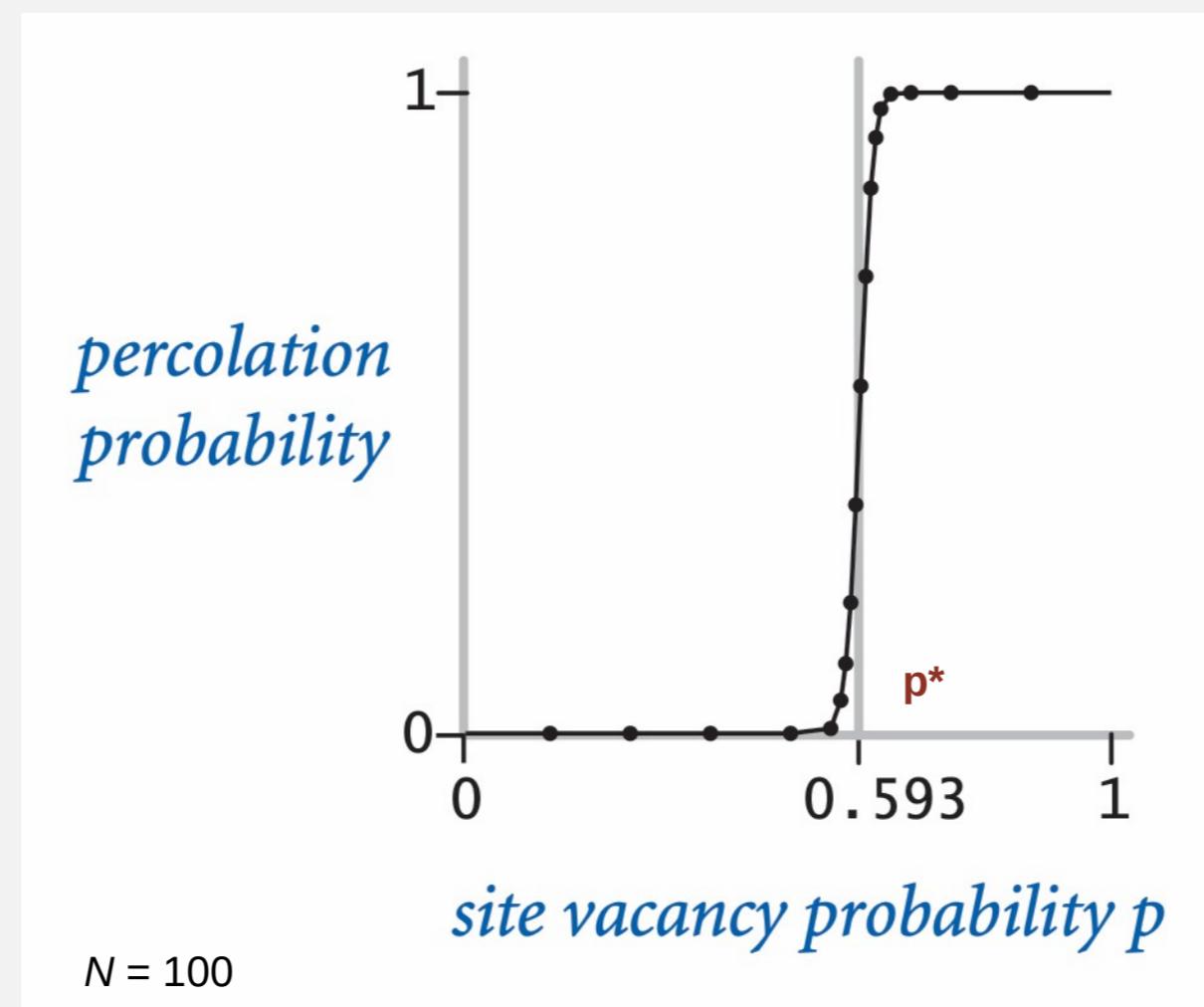


# Faseovergang for perkolation

Hvis  $N$  er stor, så garanterer teorien en skarp grænse  $p^*$ .

- $p > p^*$ : perkolerer næsten med sikkerhed.
- $p < p^*$ : perkolerer næsten med sikkerhed ikke.

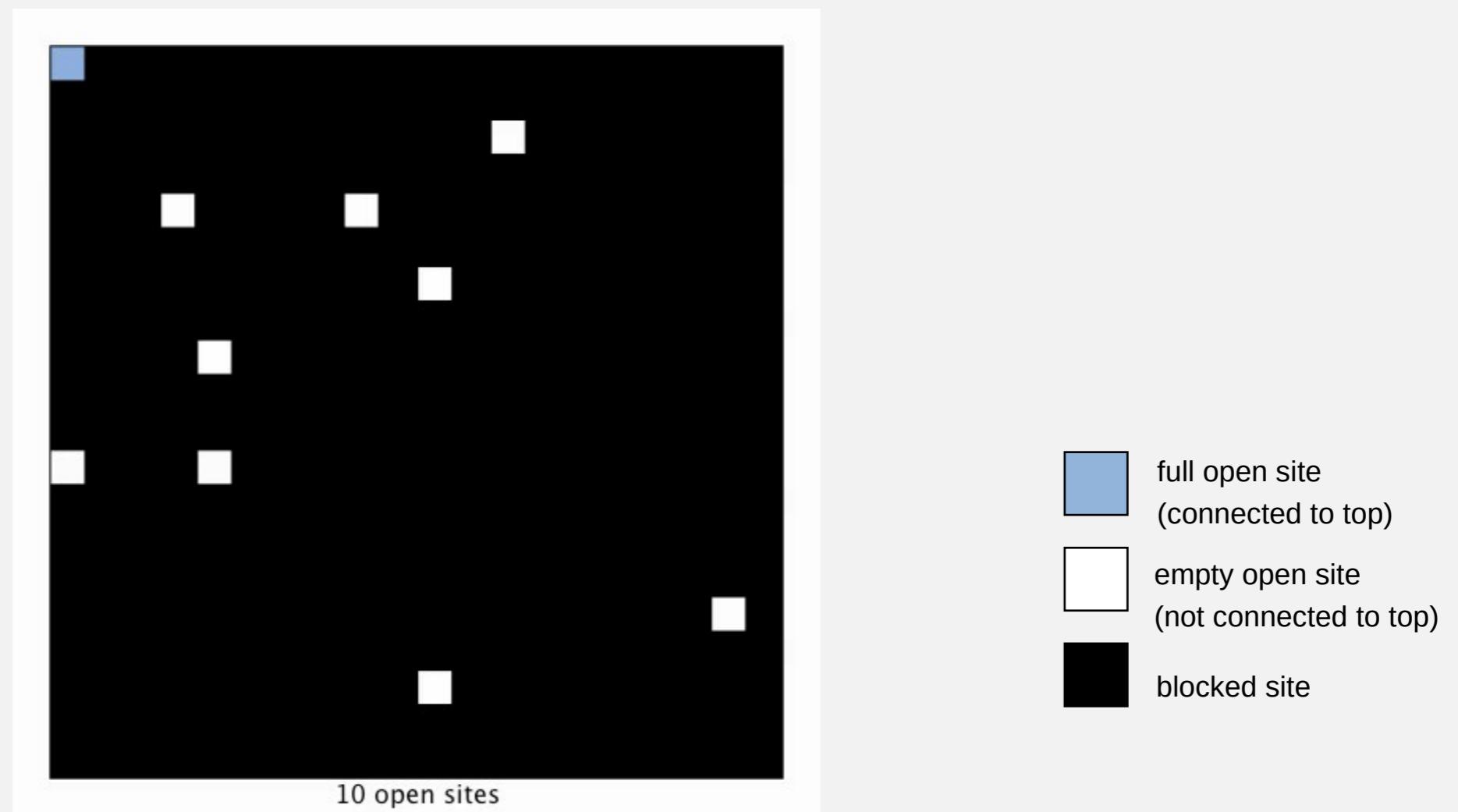
Q. Hvad er værdien af  $p^*$  ?



# Monte Carlo simulation

---

- Alle celler i et  $N$ -gange- $N$  gitter starter med at være blokkerede.
- Åben tilfældige celler indtil toppen er forbundet med bunden.
- Andel af åbne, når det sker, estimerer  $p^*$ .



$N = 20$

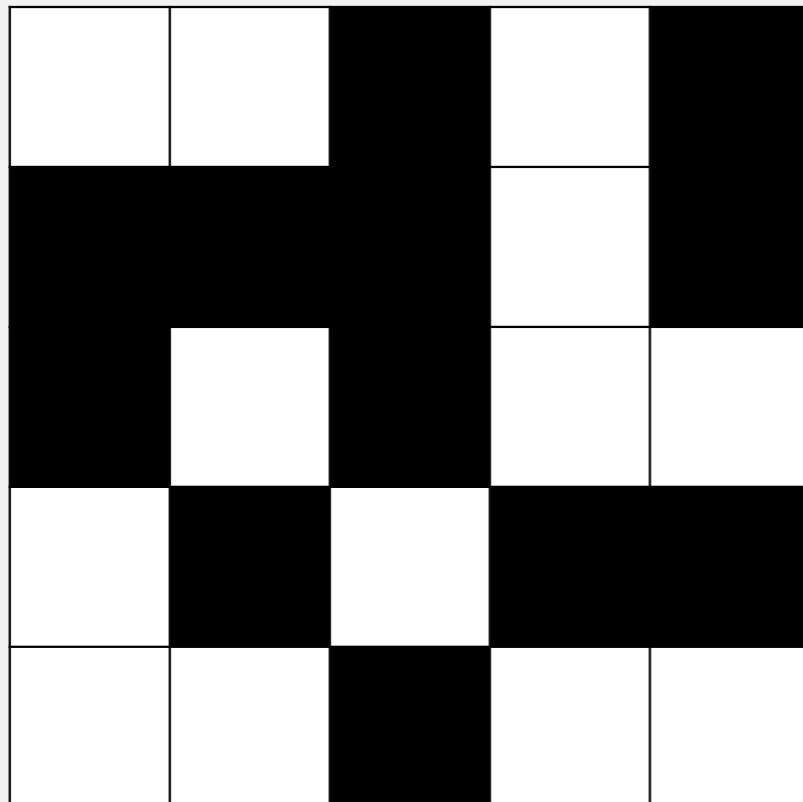
# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

---

Q. Hvordan undersøger man om et  $N$ -gange- $N$  system perkolerer?

A. Modellér som et **dynamisk sammenhængs** problem og brug **forén-og-find**.

$$N = 5$$



open site



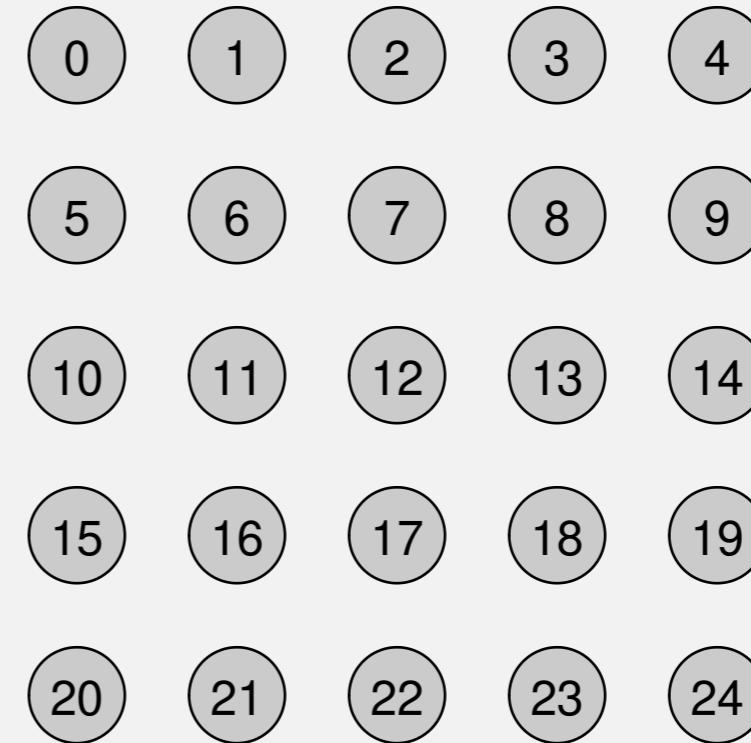
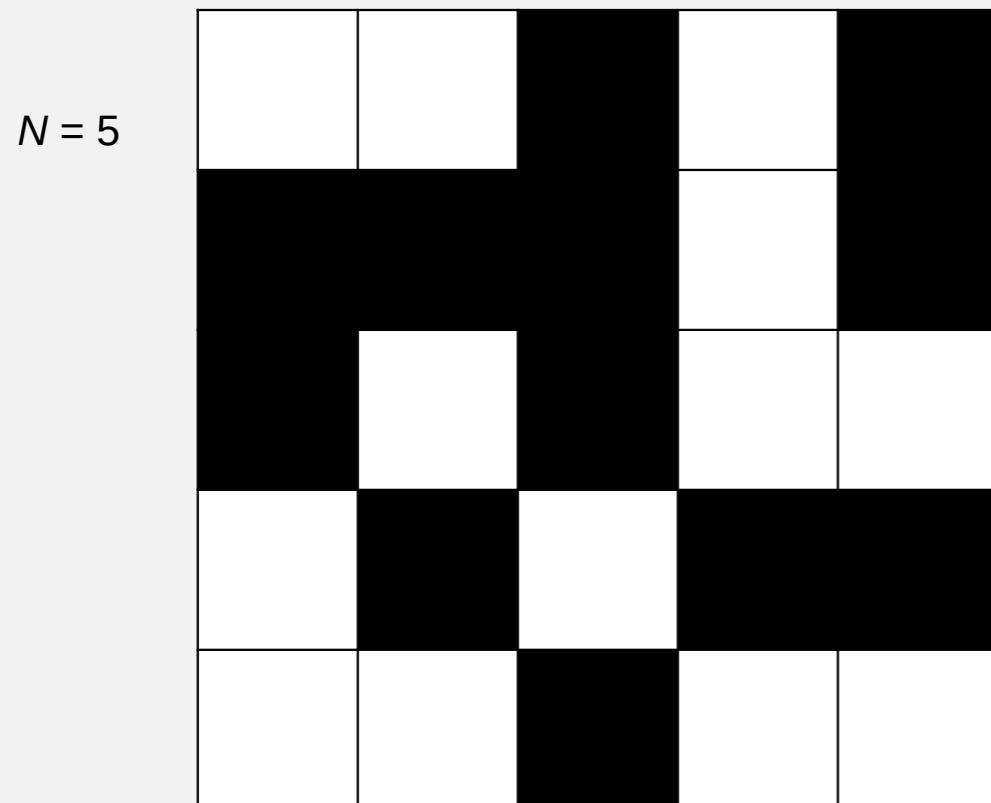
blocked site

# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

---

Q. Hvordan undersøger man om et  $N$ -gange- $N$  system perkolerer?

- Lav elementer for hver celle og navngiv dem 0 til  $N^2 - 1$ .



open site

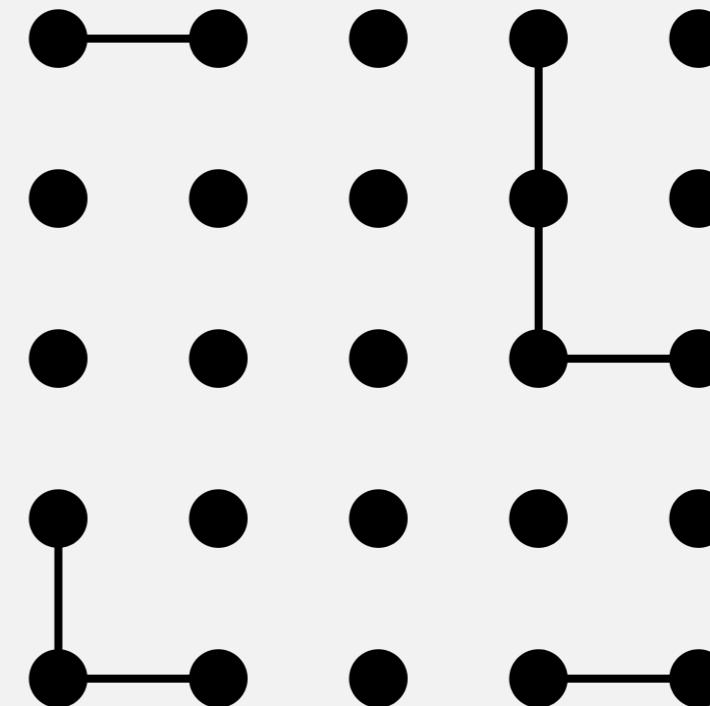
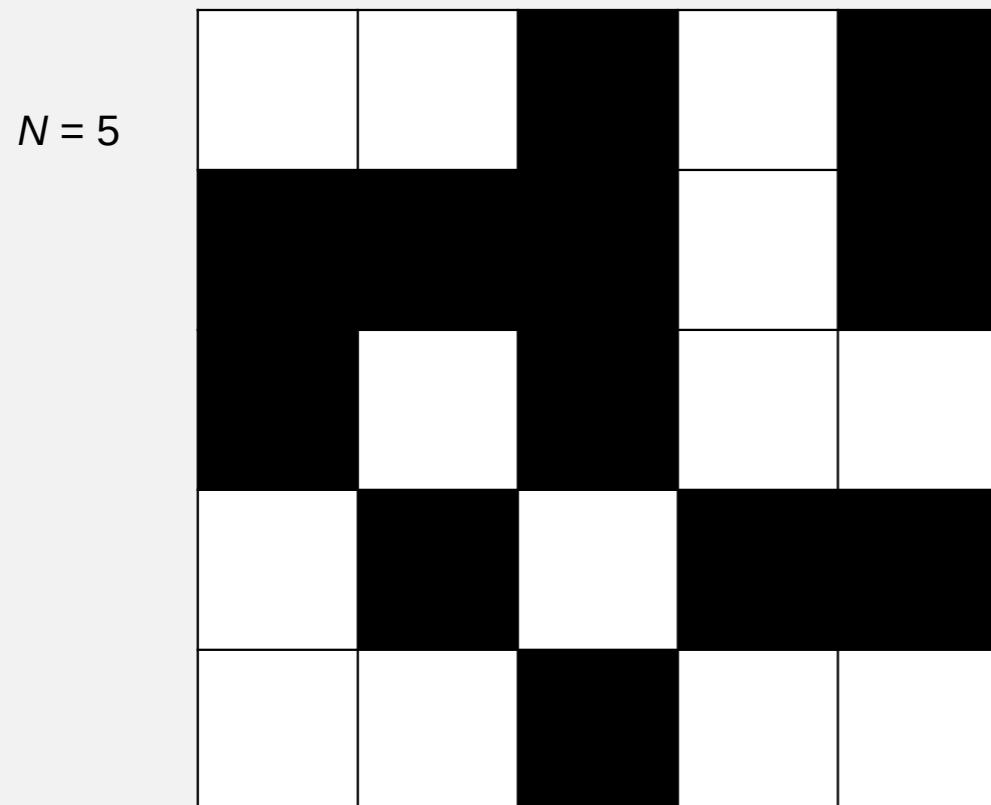
blocked site

# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

---

Q. Hvordan undersøger man om et  $N$ -gange- $N$  system perkolerer?

- Lav elementer for hver celle og navngiv dem 0 til  $N^2 - 1$ .
- Celle er i samme komponent hviss de er forbundne af åbne celler.



open site

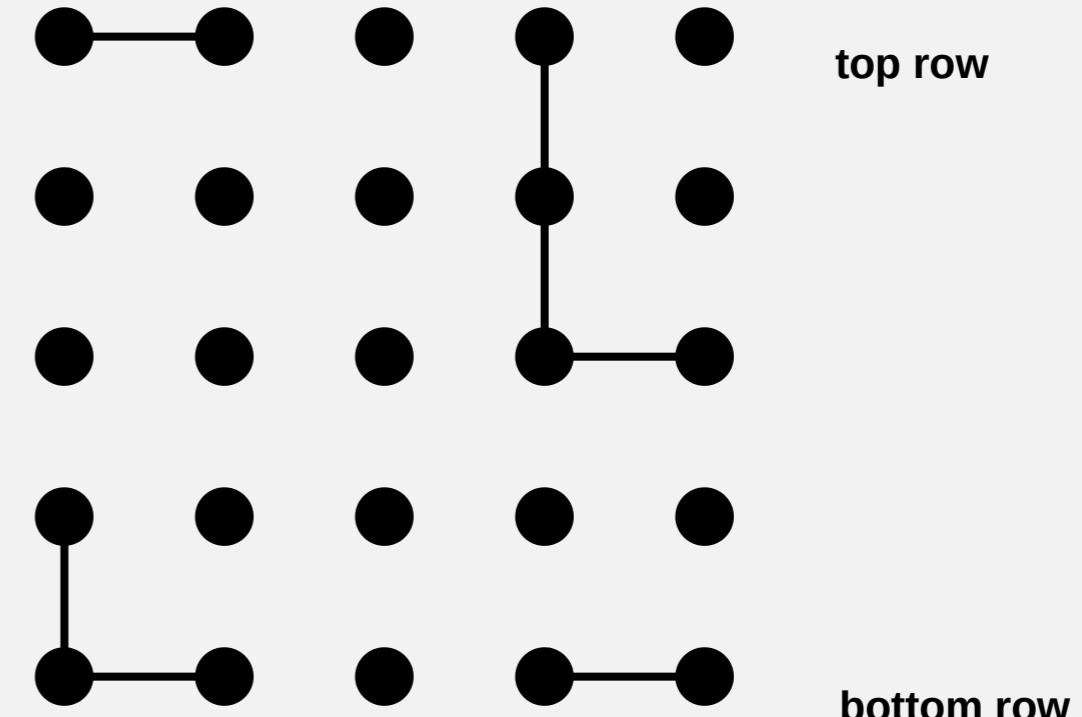
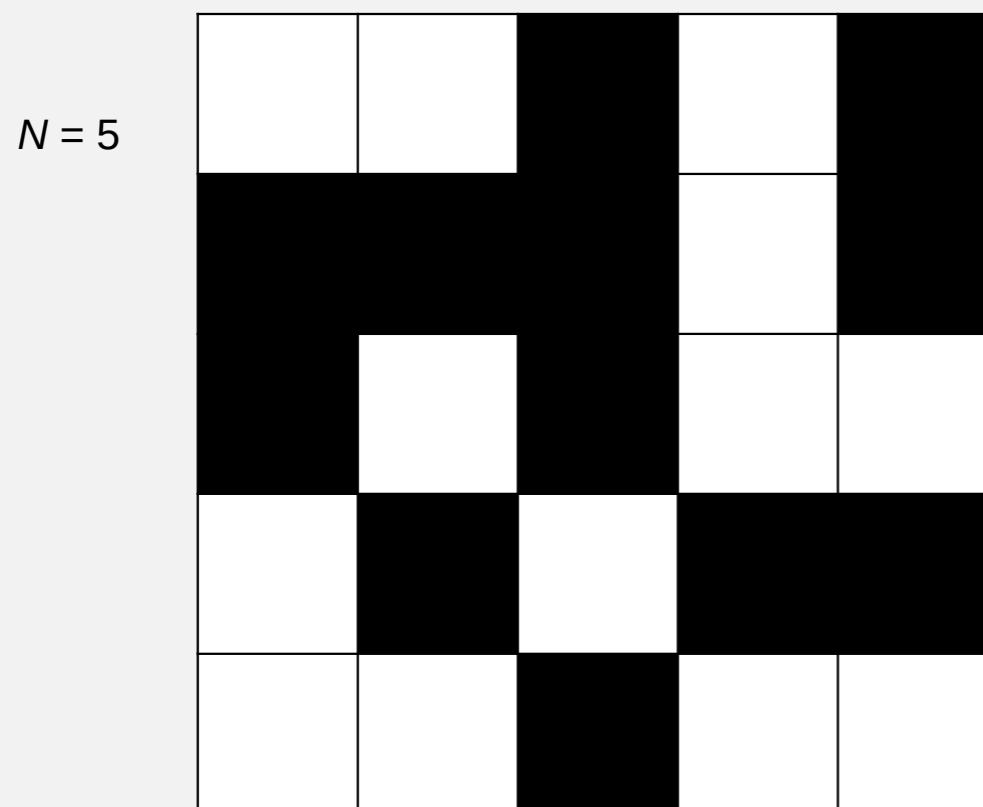
blocked site

# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

Q. Hvordan undersøger man om et  $N$ -gange- $N$  system perkolerer?

- Lav elementer for hver celle og navngiv dem 0 til  $N^2 - 1$ .
- Celle er i samme komponent hviss de er forbundne af åbne celler.
- Perkolerer hviss nogen celle i toppen er forbundet men nogen celle i bunden.

brute-force algoritme:  $N^2$  kald til connected()



open site

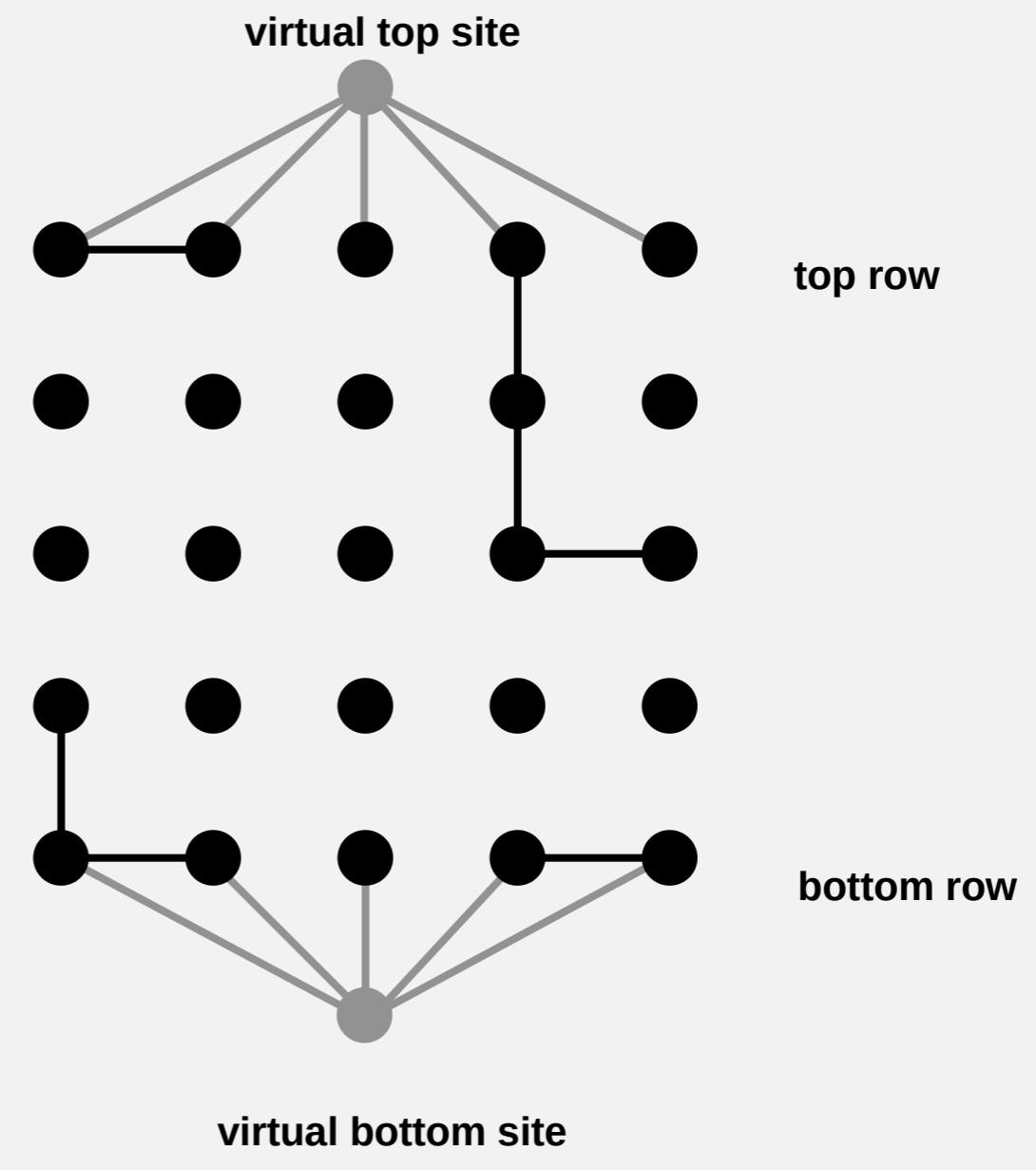
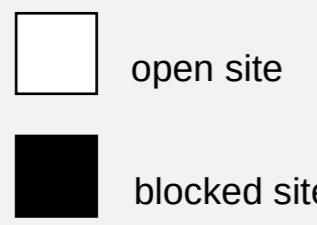
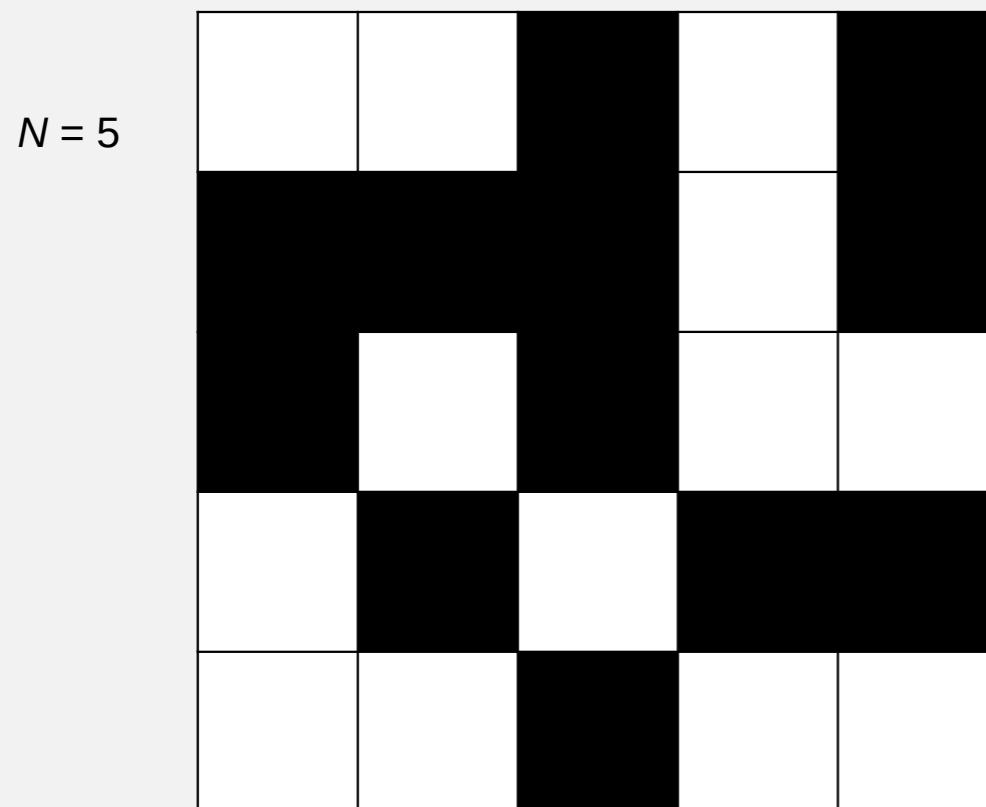
blocked site

# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

**Smart trick.** Indfør 2 virtuelle celler (med forbindelser til hhv top og bund).

- Perkolerer hviss virtuel top celle er forbundet med virtuel bund celle.

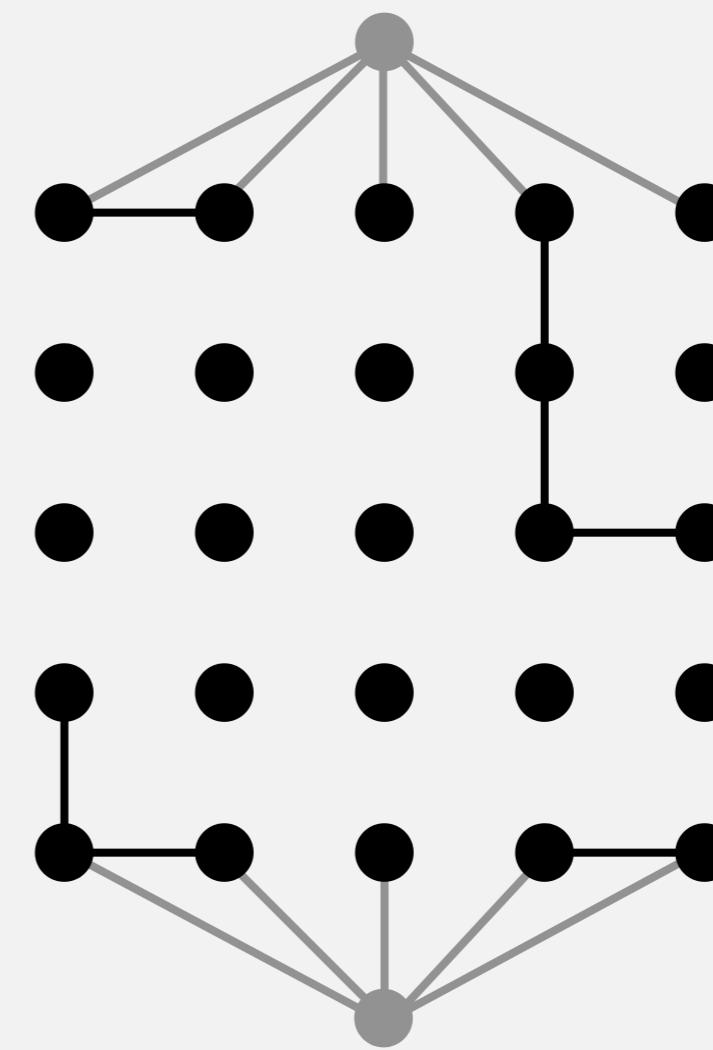
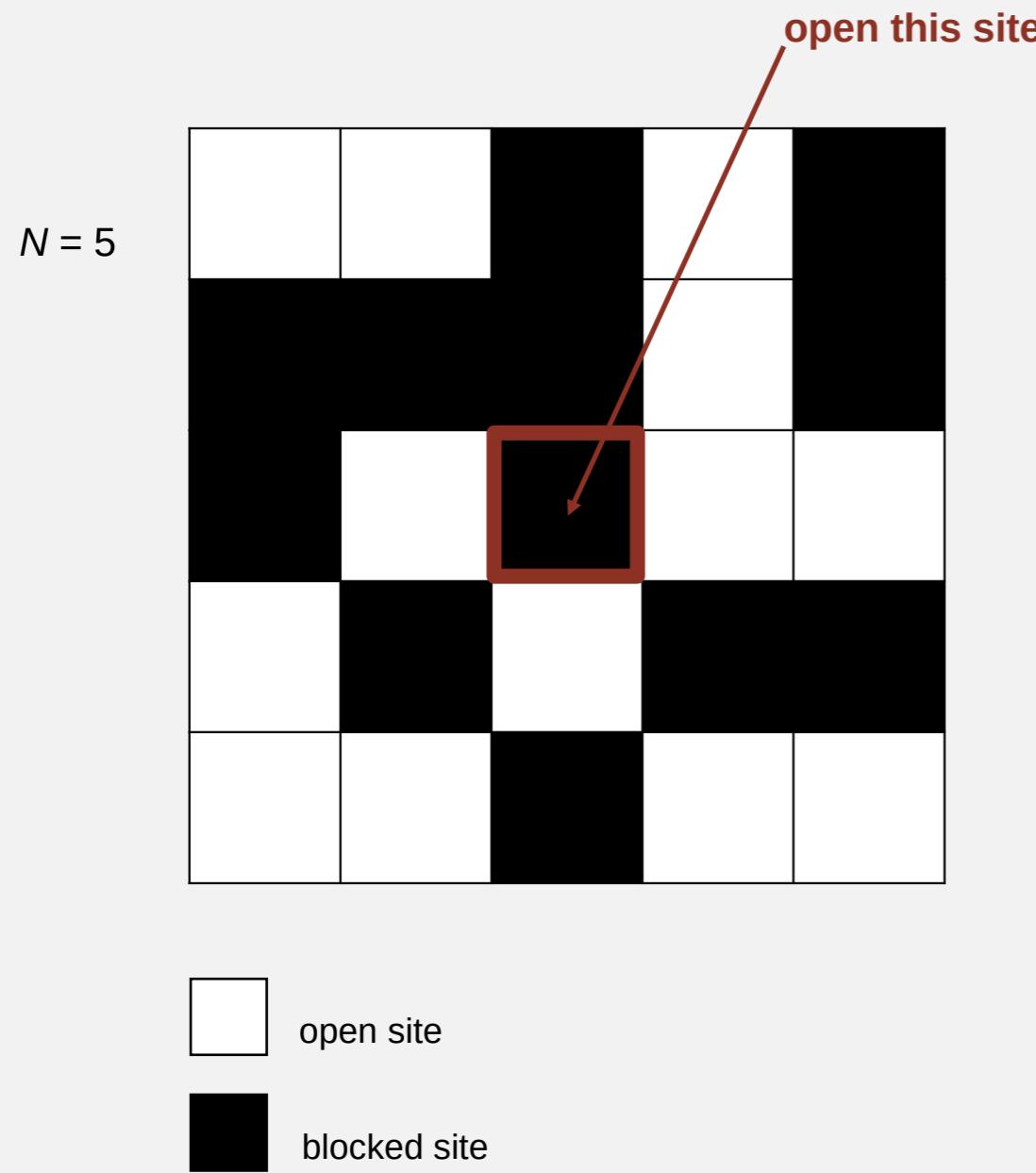
mere effektiv algoritme: kun 1 kald til connected()



# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

---

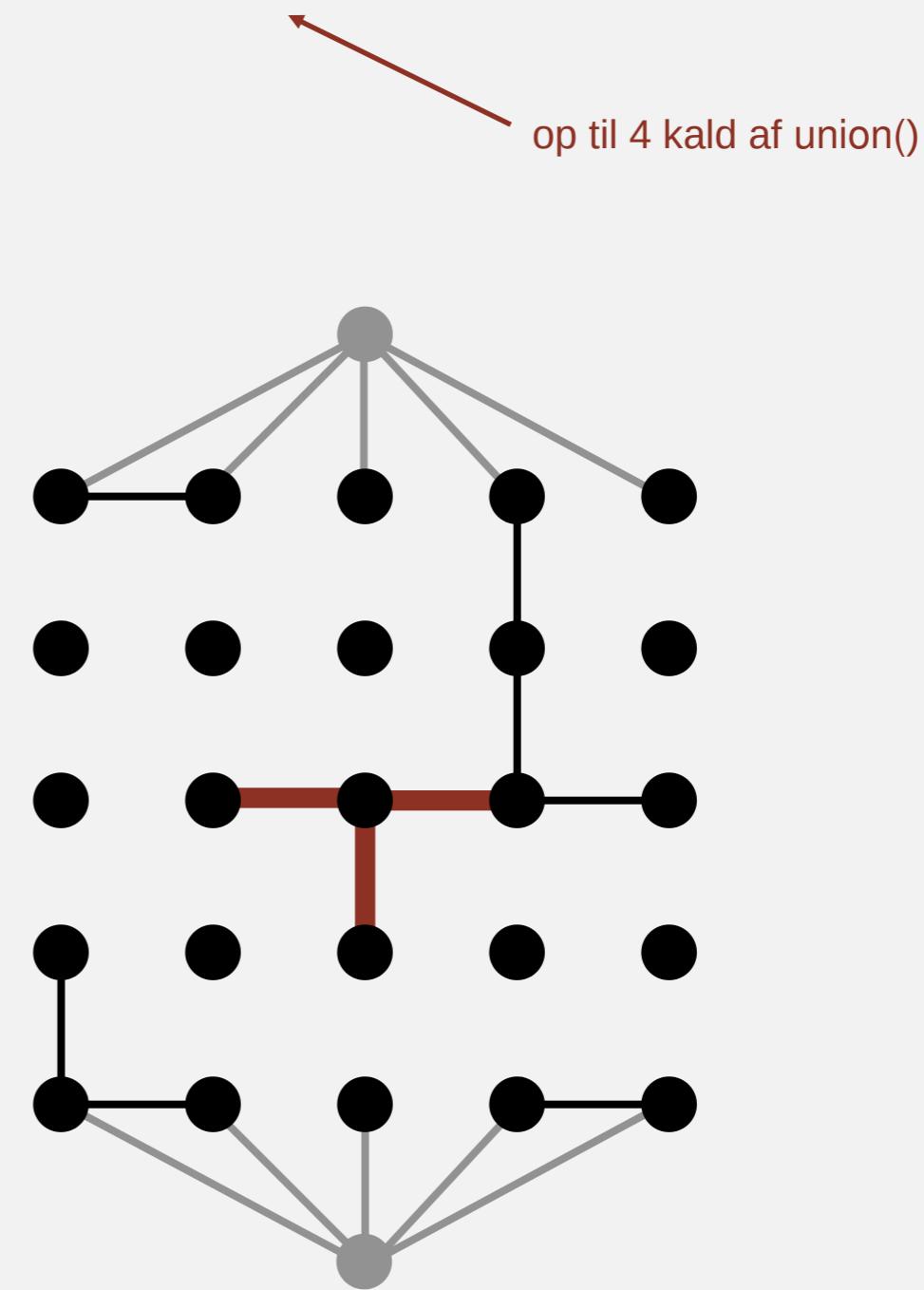
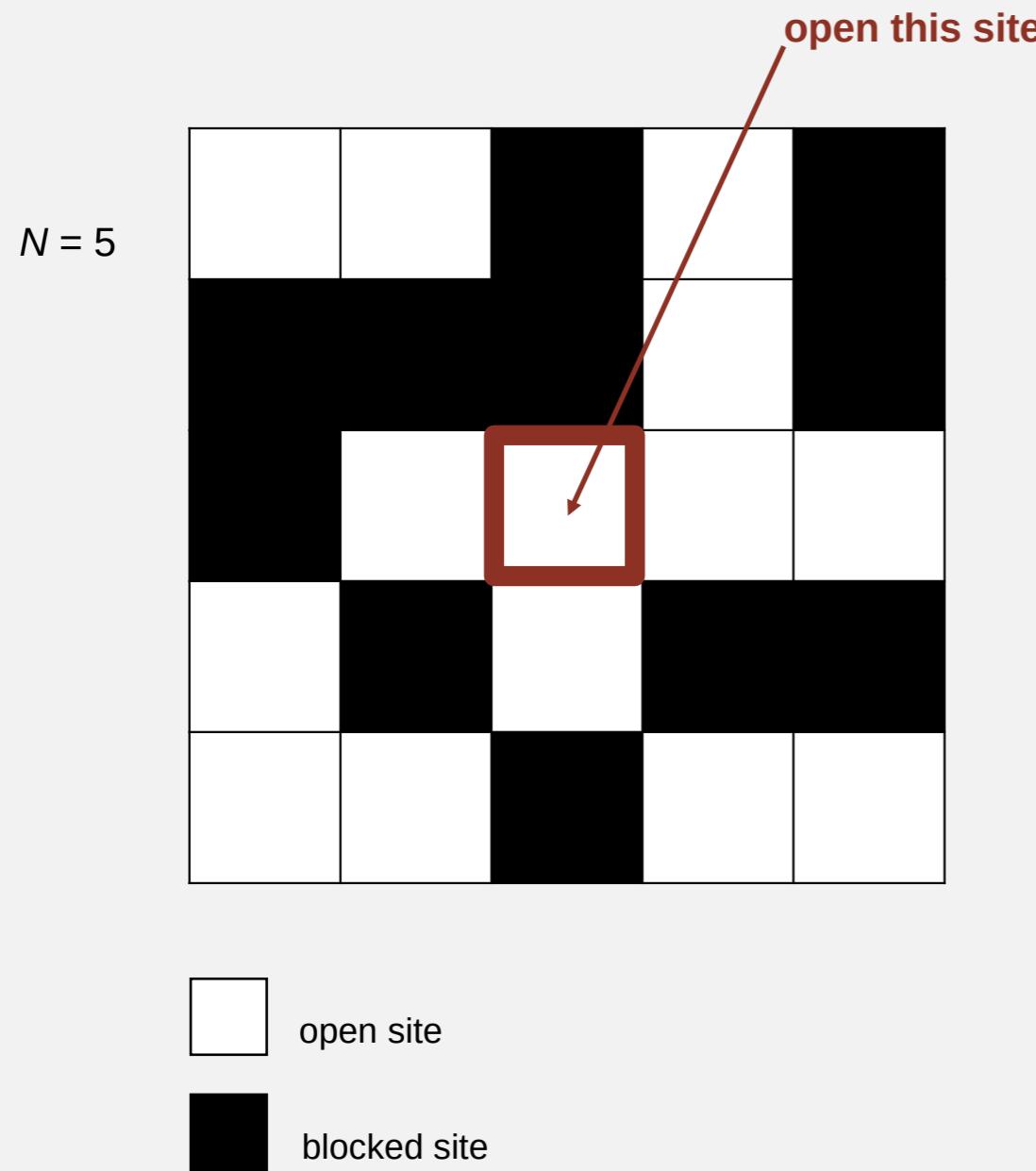
Q. Hvordan modellerer man at åbne en ny celle?



# Løsning med dynamisk sammenhæng til estimering af fasegrænsen

Q. Hvordan modellerer man at åbne en ny celle?

A. Markér ny celle som åben; forén den med alle åbne naboceller.

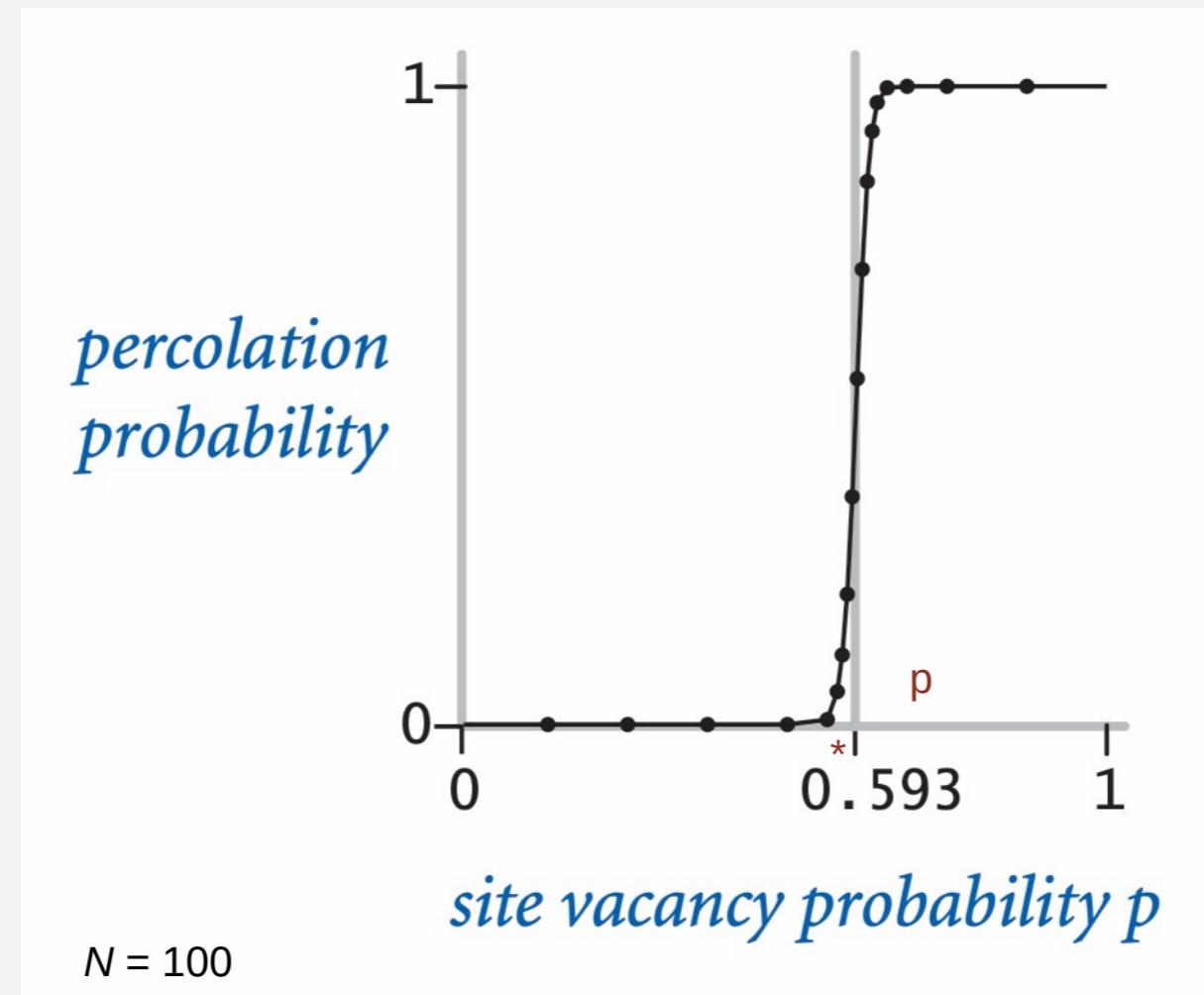


# Perkolationsgrænseværdien

Q. Hvad er perkolationsgrænseværdien  $p^*$  ?

A. Cirka 0.592746 for store kvadratiske gitre.

konstant kun kendt via simulation



Hurtige algoritmer muliggører præcise svar på videnskabelige spørgsmål.

## Næste punkt på programmet

---

- 12:00 - 12:45: Frokost
- 12:45 - 14:00: Værktøjer og Kattis
- 14:00 - 14:15: Kaffepause
- 14:15 - 16:00: Øvelser