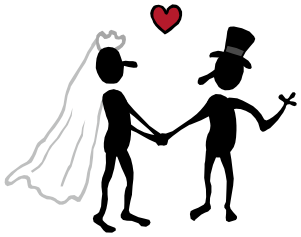
UNION-FIND

UNION-FIND-problemet



inddata:
en følge af heltalspar (p, q);
betydning: p er "forbundet med" q

uddata:
intet, hvis p og q er forbundet,
ellers (p, q)

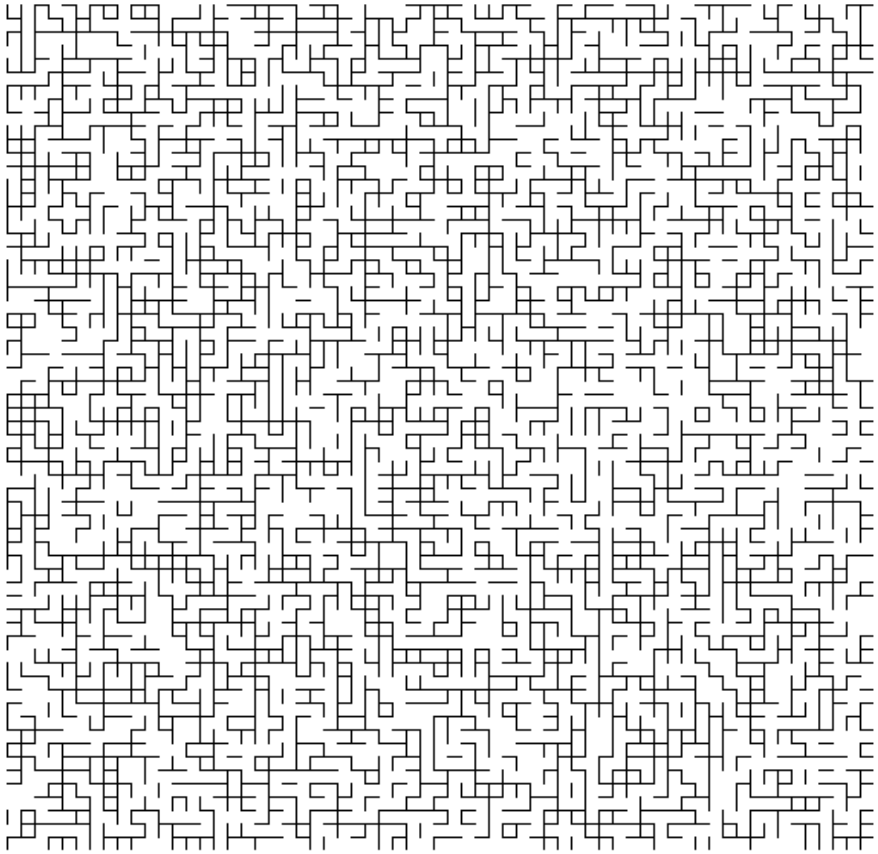
Eksempel på anvendelse:
Forbindelser i computernetværk

- Hvert heltal repræsenterer en computer
- Hvert talpar repræsenterer en netværksforbindelse imellem to computere

Eksempel på udførelse

ind	ud	begrundelse
3 4	3 4	
4 9	4 9	
8 0	8 0	
2 3	2 3	
5 6	5 6	
2 9		2 -- 3 -- 4 -- 9
5 9	5 9	
7 3	7 3	
4 8	4 8	
5 6		5 -- 6
0 2		2 -- 3 -- 4 -- 8 -- 0

Forbundethed kan være svær at afgøre
(især for en computer)



Egenskaber ved relationen "forbundet med", *F*

(1) Hvis p er forbundet med q, så er q forbundet med p

Symmetri: $F(p, q) \Rightarrow F(q, p)$

(2) Hvis p er forbundet q, og q er forbundet med r, så er p forbundet med r

Transitivitet: $F(p, q) \wedge F(q, r) \Rightarrow F(p, r)$

(3) p er forbundet med sig selv

Refleksivitet: $F(p, p)$

F er en **ækvivalensrelation**

Knuder, der er forbundet med hinanden, tilhører samme **ækvivalensklasse** (komponent)

Version 1
(Quick-FIND)

Vedligehold et array, `id`, over navne på komponenter:
Hvis `p` og `q` er forbundet, så er `id[p]` = `id[q]`
(ellers er `id[p]` ≠ `id[q]`)

Dette kan gøres ved for hvert par (`p`, `q`)

- at gøre ingenting, hvis `id[p]` = `id[q]`
- ellers at ændre alle indgange med `p`'s `id` til `q`'s `id`

```
if (id[p] == id[q]) continue;
for (int i = 0; i < N; i++)
    if (id[i] == id[p])
        id[i] = id[q];
```

Initialisering:

Sæt `id[i]` = `i` for ethvert `i`

Quick-FIND

Navnet skyldes et **konstant** tidsforbrug til at afgøre,om der findes en forbindelse mellem `p` og `q` (om `p` og `q` tilhører samme komponent). Denne operation kaldes for FIND.

Med hvad med tidsforbruget til at forene to komponenter? Denne operation kaldes for UNION.

Et fingerpeg herom fås ved at følge ændringerne af arrayet `id`.

Inddata	`id`
	0 1 2 3 4 5 6 7 8 9
3 4	0 1 2 4 4 5 6 7 8 9
4 9	0 1 2 9 9 5 6 7 8 9
8 0	0 1 2 9 9 5 6 7 0 9
2 3	0 1 9 9 9 5 6 7 0 9
5 6	0 1 9 9 9 6 6 7 0 9
5 9	0 1 9 9 9 9 9 7 0 9
7 3	0 1 9 9 9 9 9 9 0 9
4 8	0 1 0 0 0 0 0 0 0 0
6 1	1 1 1 1 1 1 1 1 1 1

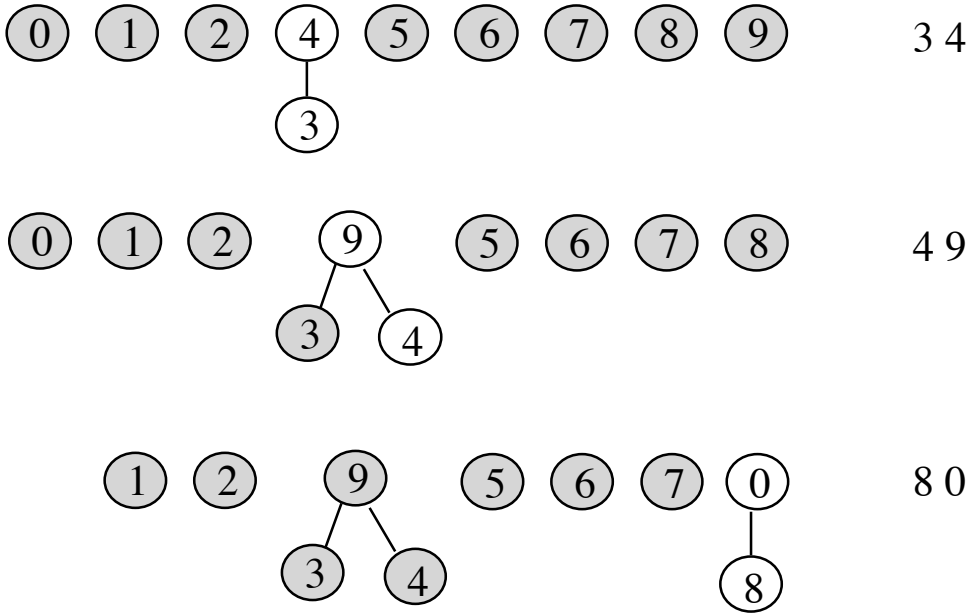
Problem ved Quick-FIND:
- undersøger alle indgange i `id` ved UNION og er derfor for langsom for store problemer

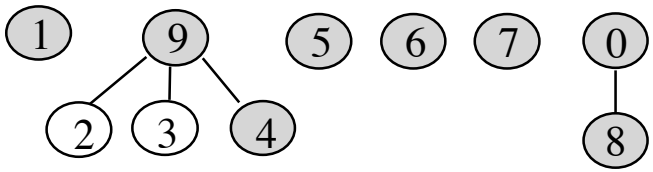
Brug grafisk repræsentation

Numre i knuder

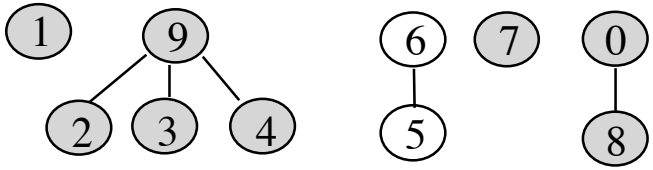
Topknuder repræsenterer komponenter (`id`)

Knude `i` tilhører komponenten `id[i]`

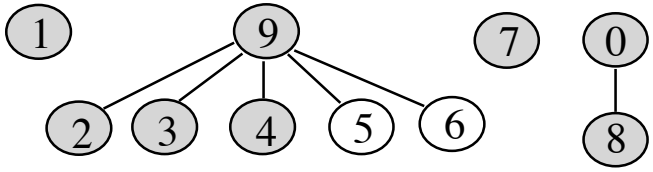




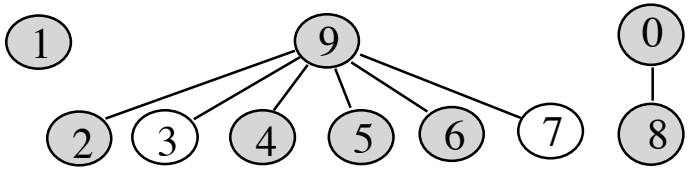
2 3



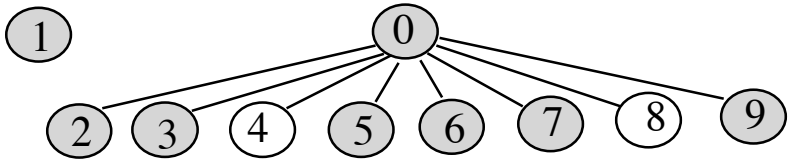
5 6



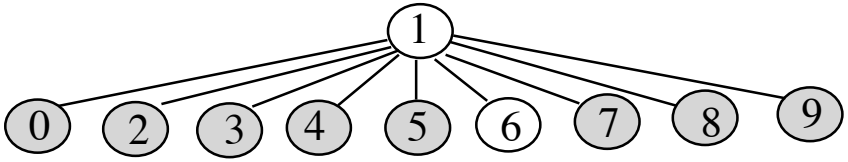
5 9



7 3



4 8



6 1

Problemstørrelse og tidsforbrug

Givet et **stort** problem med 10^9 knuder og 10^{10} forbindelser (f.eks. telefonnetværk, computerchip)

For hver ny forbindelse gennemløber Quick-FIND hele arrayet `id`

Hvis der for hver tabeltilgang i gennemsnit benyttes 10 maskinoperationer, så udføres i værste tilfælde cirka $10^9 * 10^{10} * 10 = 10^{20}$ maskinoperationer

Antag at tidsforbruget for en maskinoperation i gennemsnit er 10^{-9} sekunder. Så vil det samlede tidsforbrug være cirka $10^{20} * 10^{-9}$ sekunder = 10^{11} sekunder $\approx$ **3000 år**!

Version 2
(Quick-UNION)

Vedligehold et array over navne på komponenter:

hvis p og q er forbundet, så er (`id[p]`)* = (`id[q]`)*,

hvor (`id[p]`)* = `id[id[id[...id[p]]]]`

(fortsæt indtil værdien ikke ændres)

ellers er (`id[p]`)* ≠ (`id[q]`)*

Dette kan gøres ved for hvert par (`p`, `q`)

- at gøre ingenting, hvis (`id[p]`)* = (`id[q]`)*
- ellers sættes `id[i]` lig med `j`,
  hvor `i` = (`id[p]`)* og `j` = (`id[q]`)*

```
int i = p, j = q;
while (i != id[i]) i = id[i];
while (j != id[j]) j = id[j];
if (i == j) continue;
id[i] = j;
```

Quick-UNION

Navnet skyldes et **konstant** tidsforbrug til at forene to komponenter.

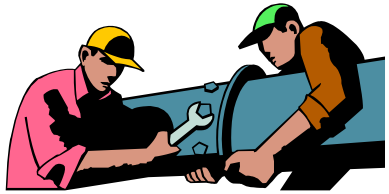
(operationen UNION)

Med hvad med tidsforbruget til at at afgøre, om der findes en forbindelse mellem `p` og `q`?

(operationen FIND)

Et fingerpeg herom fås ved at følge ændringerne af arrayet `id`.

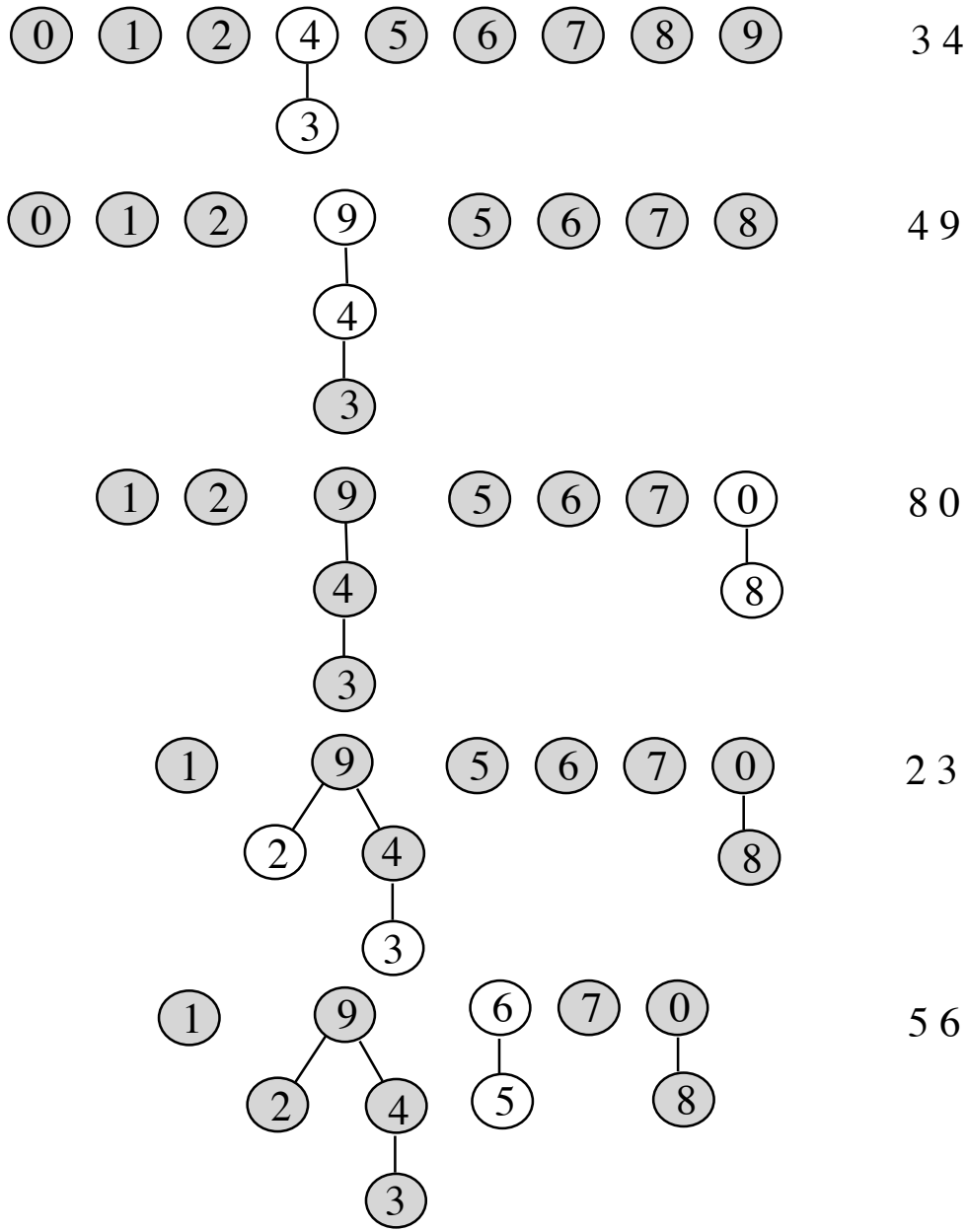
Inddata	id
	0 1 2 3 4 5 6 7 8 9
3 4	0 1 2 4 4 5 6 7 8 9
4 9	0 1 2 4 9 5 6 7 8 9
8 0	0 1 2 4 9 5 6 7 0 9
2 3	0 1 9 4 9 5 6 7 0 9
5 6	0 1 9 4 9 6 6 7 0 9
5 9	0 1 9 4 9 6 9 7 0 9
7 3	0 1 9 4 9 6 9 9 0 9
4 8	0 1 9 4 9 6 9 9 0 0
6 1	1 1 9 4 9 6 9 9 0 0

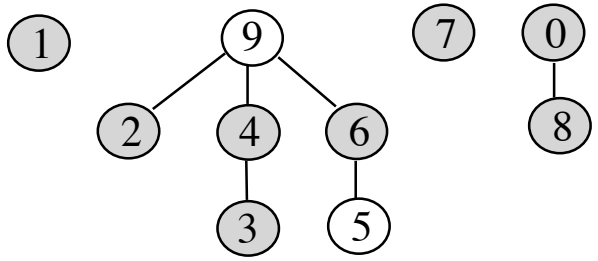


Problemer ved Quick-UNION:

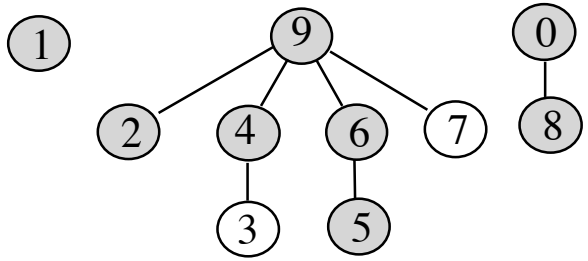
- beregningen af (`id[p]`)* og (`id[p]`)* kan være tidsmæssigt dyr
- ingen garanti for tilstrækkelig effektivitet for "meget store" problemer

Hurtigere end Quick-FIND for "tilfældige" inddata (men er inddata i praksis tilfældige?)

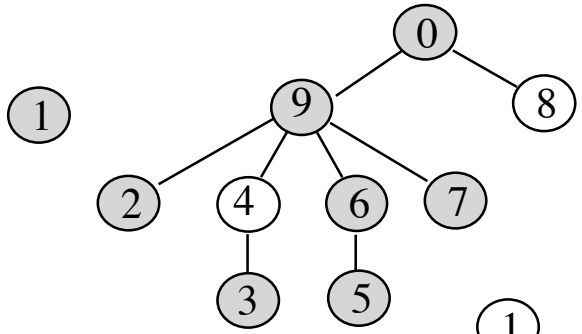

0 1 2 4 5 6 7 8 9
3
3 4
0 1 2 9 5 6 7 8
4
3
4 9
1 2 9 5 6 7 0
4 8
3
8 0
1 9 5 6 7 0
2 4 8
3
2 3
1 9 6 7 0
2 4 5 8
3
5 6
5 9



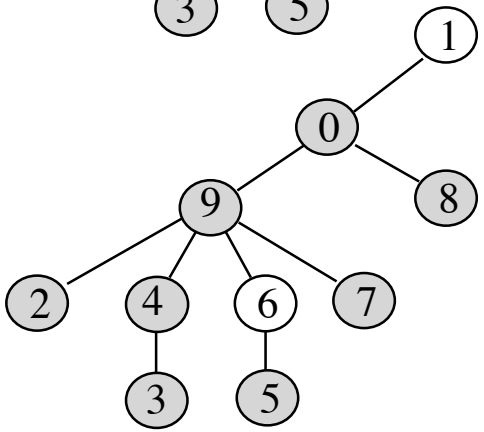
5 9



7 3



4 8

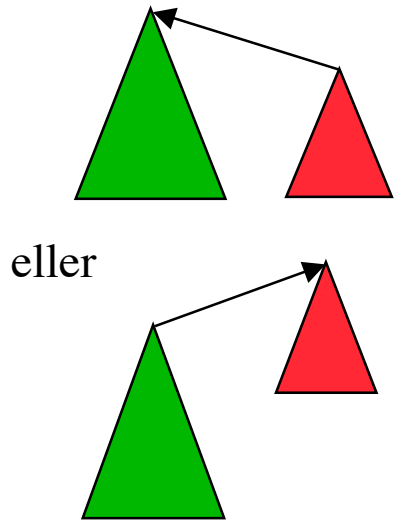


6 1

Den grafiske repræsentationen af Quick-UNION viser:

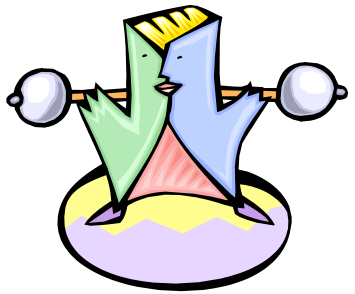
FIND gennemløber et eller to træer (tester om den samme rod nås)

UNION sammenkæder to træer:



Kan effektiviteten øges ved at vælge sammenkædningen hensigtsmæssigt?

Version 3
(vægtet quick-UNION)



Modificér Quick-UNION, så "ubalance" undgås

- vedligehold et array over komponenternes størrelse
- balancér ved at sammenkæde en mindre komponent under en større komponent

```
int i = p, j = q;
while (i != id[i]) i = id[i];
while (j != id[j]) j = id[j];
if (i == j) continue;
if (size[i] < size[j])
   { id[i] = j; size[j] += size[i]; }
else
   { id[j] = i; size[i] += size[j]; }
```

Er effektiviteten forbedret?

For at besvare dette spørgsmål:

- udfør empiriske undersøgelser, og
- analysér algoritmen matematisk

Det ses, at højden af det største træ kun øges, når det sammensættes med et lige så højt træ

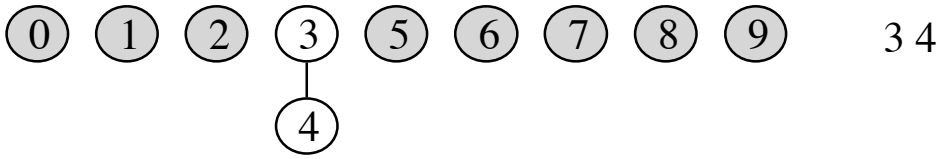
Det kan bevises, at FIND i **værste tilfælde** foretager $2 \log_2 N$ undersøgelser per forbindelse (i **gennemsnit** et konstant antal undersøgelser)

I tilfældet med 10^9 knuder og 10^{10} forbindelser udføres programmet på under **1 minut** (jvf. med de 3000 år for Quick-FIND)

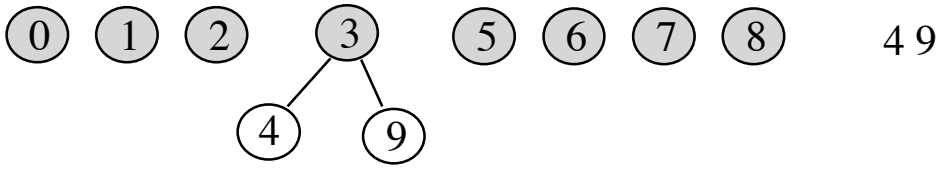
Version 4
(Vægtet Quick-UNION med vejkomprimering)

Metode: Sørg i UNION-delen for, at alle knuder, der er besøgt i FIND-delen, peger på den nye rod

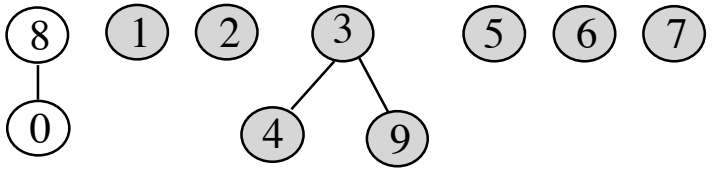
```
int i = p, j = q;
while (i != id[i]) i = id[i];
while (j != id[j]) j = id[j];
if (i == j) continue;
int root;
if (size[i] < size[j])
   { root = id[i] = j; size[j] += size[i]; }
else
   { root = id[j] = i; size[i] += size[j]; }
for (i = p; i != id[i]; i = j)
   { j = id[i]; id[i] = root; }
for (j = q; j != id[j]; j = i)
   { i = id[j]; id[j] = root; }
```



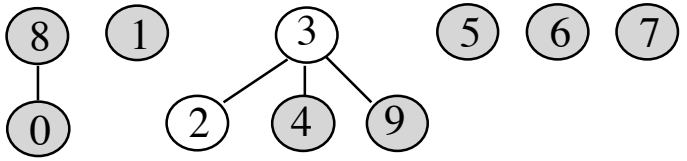
3 4



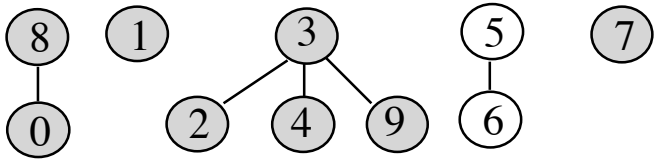
4 9



8 0

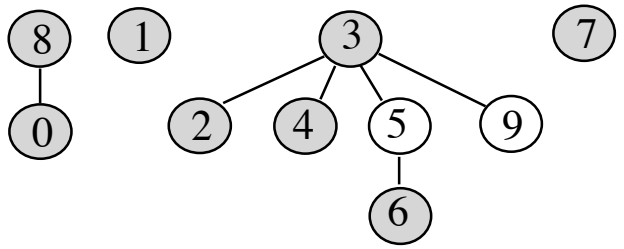


2 3

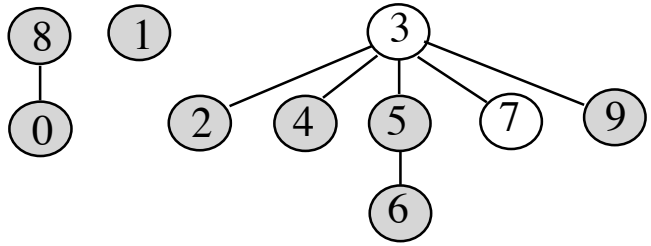


5 6

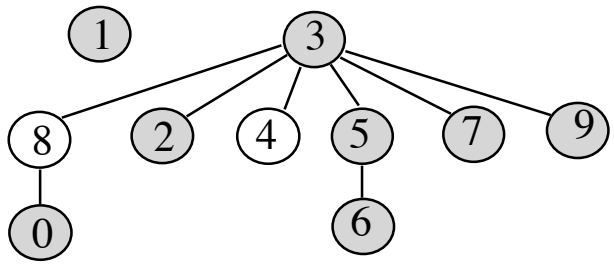
5 9



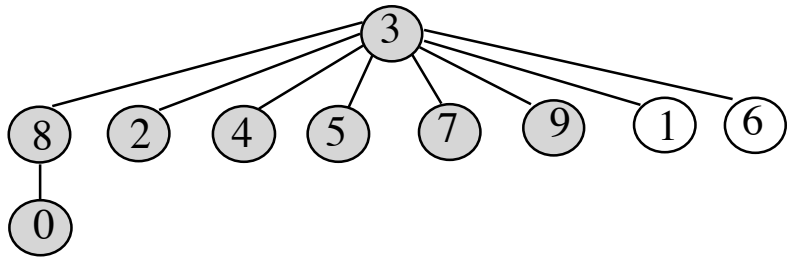
5 9



7 3



4 8



6 1

Effektiviteten af version 4

Sætning: Den maksimale højde er log*(N)

hvor funktionen log*(N) er det antal gange $\log_2$ kan tages af N, før værdien bliver ≤ 1

log* vokser meget langsomt, således er log*(N) ≤ 5 for alle praktiske værdier af N ($N \leq 2^{65536} \approx 10^{20000}$).
Den maksimale højde er i praksis konstant

```
log*(2) = 1
log*(4) = log*(2^2) = 2
log*(16) = log*(2^4) = 3
log*(65536) = log*(2^16) = 4
log*(2^65536) = 5
```

Beviset for sætningen er meget svært (på trods af, at algoritmen er simpel)

[$\log_2(10^9) \approx 30$, mens log*$(10^9) \approx 5$]

Sammenfatning

Gotha Andersen

Værste tidsforbrug per forbindelse er proportional med

Quick-FIND	N
Quick-UNION	N
Vægtet Quick-UNION	$\log_2(N)$
Vejkomprimering	$\log^*(N) \leq 5$

Og hvad kan vi så lære af det?

- start med en simpel algoritme
- benyt ikke en simpel algoritme på et "stort" problem
- abstraktioner (her træer) er værdifulde
- tilstræb garanti for køretidsforbrug i værste tilfælde