Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

Holger Dell
Folien adaptiert von Philip Bille

Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

Union Find

- Union find. Abstrakte Datenstruktur. Verwalte dynamische Familie von Mengen und unterstütze die folgenden Operationen:
  - INIT(n): Konstruiere die Mengen {0}, {1}, …, {n-1}
  - UNION(i,j): Vereinige diejenigen zwei Mengen, die Element i und j enthalten. Wenn i und j bereits in derselben Menge sind, geschieht nichts.
  - FIND(i): Liefere einen Repräsentanten für diejenige Menge, in der i enthalten ist.

INIT(9)

{0} {1} {2} {3} {4} {5} {6} {7} {8}

{1, 0, 6} {8, 3, 2, 7} {4, 5} $\xrightarrow{\text{UNION(5,0)}}$ {1, 0, 6, 4, 5} {8, 3, 2, 7}

Union Find

- Anwendungen.
  - Dynamischer Zusammenhang.
  - Minimaler Spannbaum.
  - Identifizierung in Logik und Compilern.
  - Nächster gemeinsamer Vorfahre in Bäumen.
  - Hoshen-Kopelman Algorithmus in der Physik.
  - Spiele (Hex und Go)
  - Beispiel cleverer Techniken im Entwurf von Datenstrukturen.

Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

Quick Find

- Quick find. Verwalte Feld id[0..n-1], sodass id[i] = Repräsentant für i.
  - Init(n): setze id, sodass alle Elemente sich selbst repräsentieren (id[i]=i).
  - Union(i,j): wenn Find(i) ≠ Find(j), aktualisiere Repräsentant für alle Elemente in einer der Mengen.
  - Find(i): liefere Repräsentant.

Init(9)

{0} {1} {2} {3} {4} {5} {6} {7} {8}

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

{1, 0, 6} {8, 3, 2, 7} {4, 5}

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

Union(5,0)

{1, 0, 6, 4, 5} {8, 3, 2, 7}

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

Quick Find

```
INIT(n):
  for k = 0 to n-1
    id[k] = k
```

```
FIND(i):
  return id[i]
```

```
UNION(i,j):
  iID = FIND(i)
  jID = FIND(j)
  if (iID ≠ jID)
    for k = 0 to n-1
      if (id[k] == iID)
        id[k] = jID
```

{1, 0, 6} {8, 3, 2, 7} {4, 5}

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

UNION(5,0) →

{1, 0, 6, 4, 5} {8, 3, 2, 7}

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

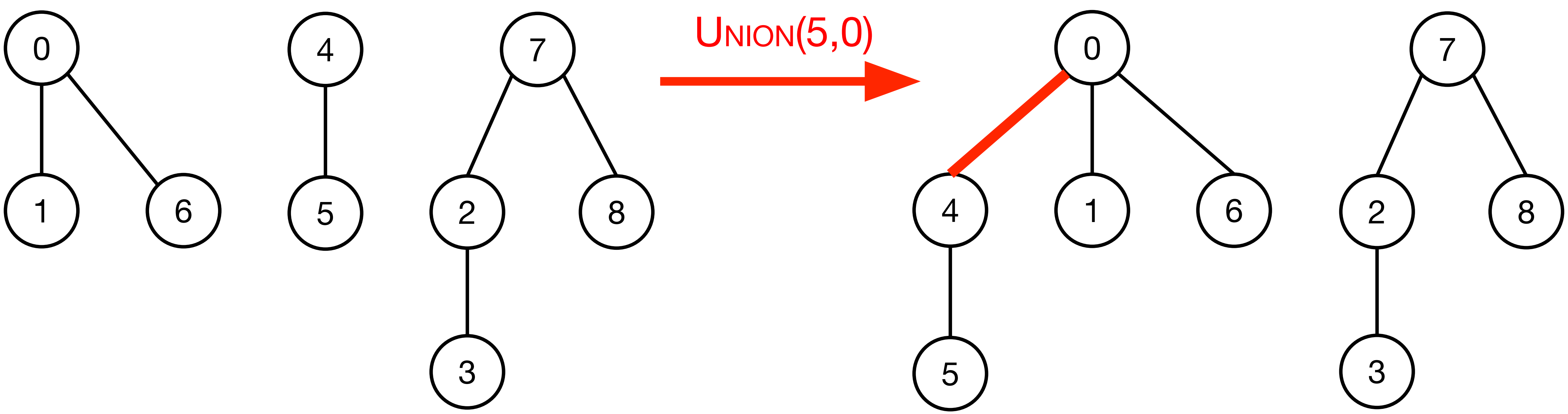
- Zeit.
  - O(n) Zeit für INIT, O(n) Zeit für UNION, und O(1) Zeit für FIND.

Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

Quick Union

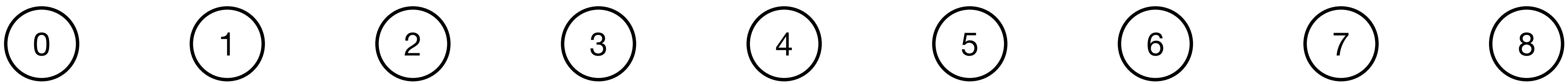
- Quick union. Verwalte jede Menge als gewurzelter Baum.
- Speichere Bäume **als Feld** p[0..n-1], sodass p[i] der Vorfahre von i ist und p[root] = root. Der Repräsentant ist die Wurzel des Baums.
  - Init(n): erzeuge n Bäume mit je einem Element.
  - Union(i,j): wenn Find(i) ≠ Find(j), mache die Wurzel des einen Baums das Kind der Wurzel des anderen Baums.
  - Find(i): folge dem Pfad zur Wurzel und liefere die Wurzel.



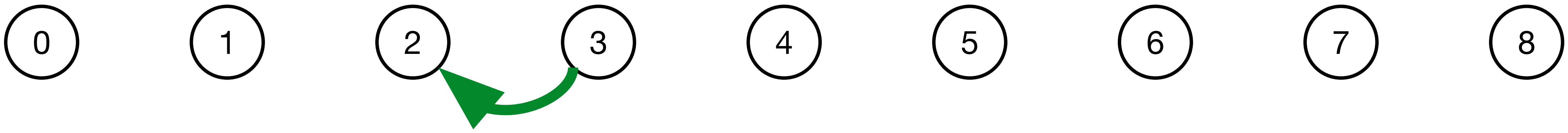
	0	1	2	3	4	5	6	7	8
p[]	0	0	7	2	4	4	0	7	7

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

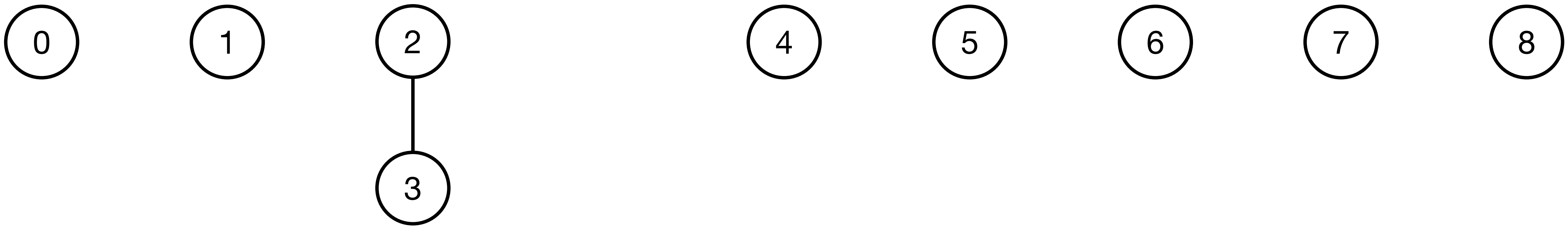
INIT(9)



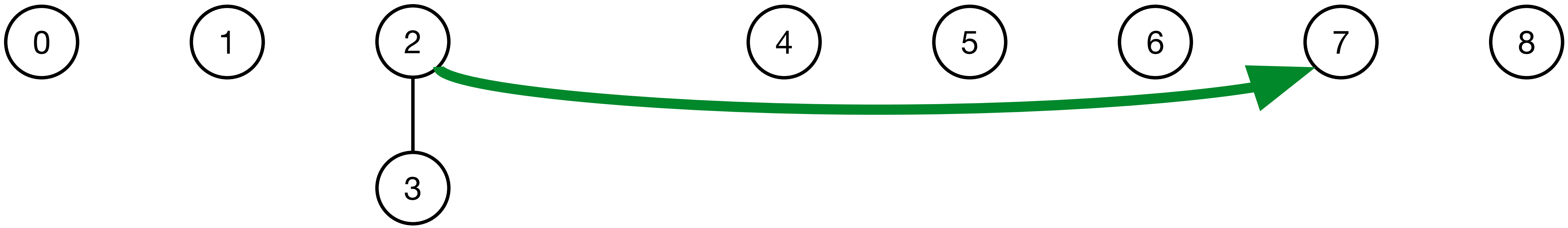
UNION(3,2)



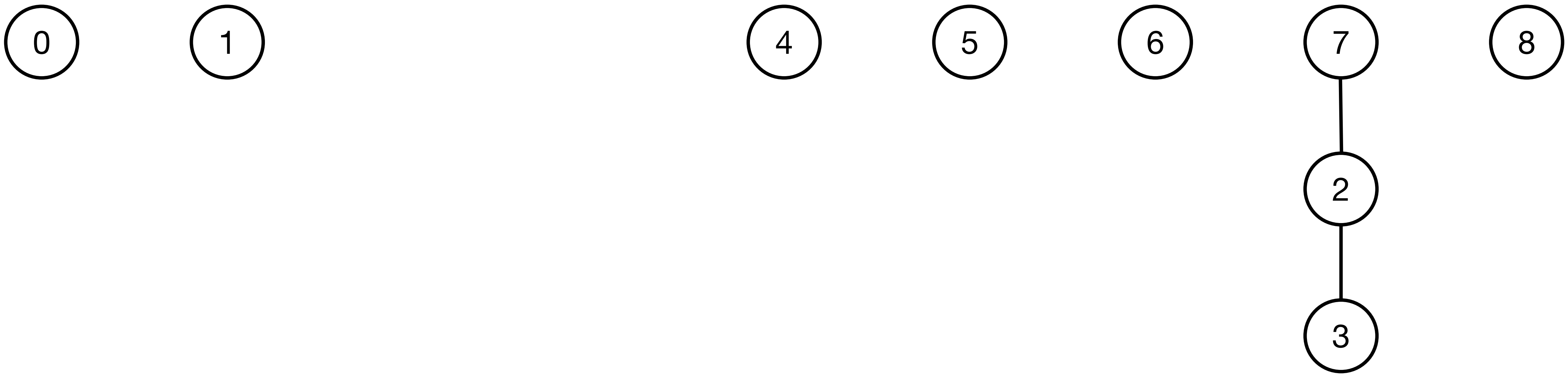
UNION(3,2)



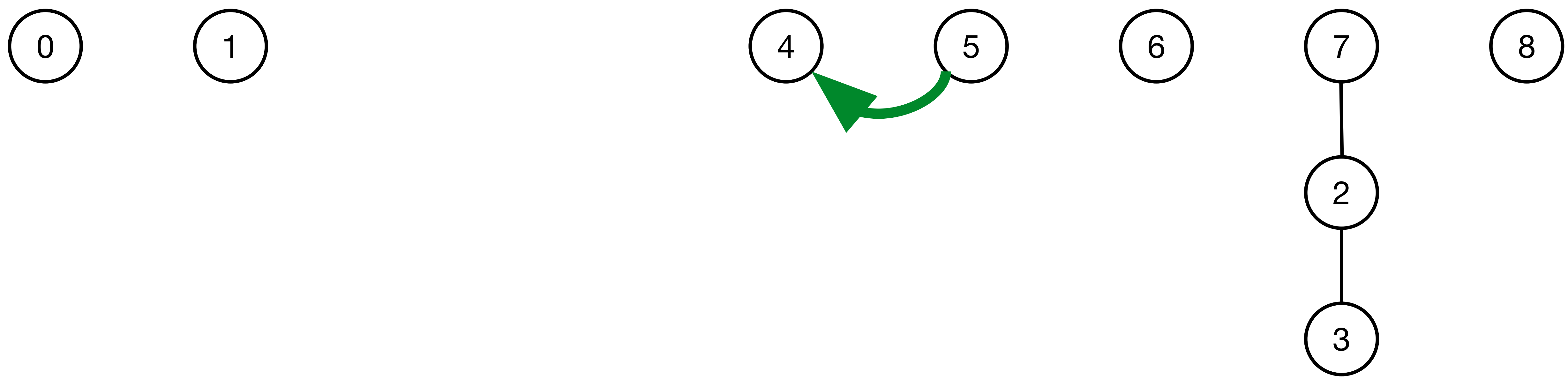
UNION(2,7)



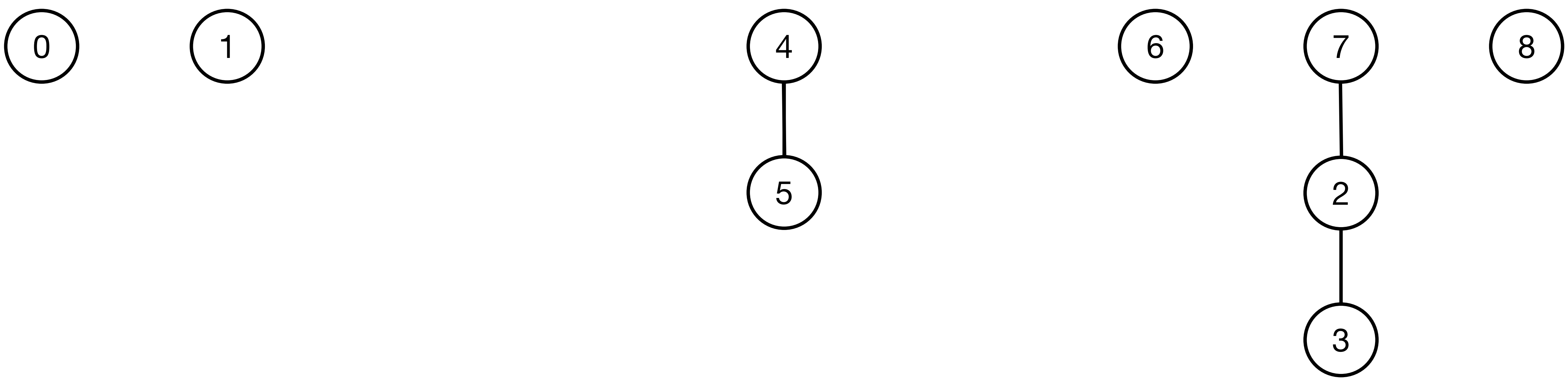
UNION(2,7)



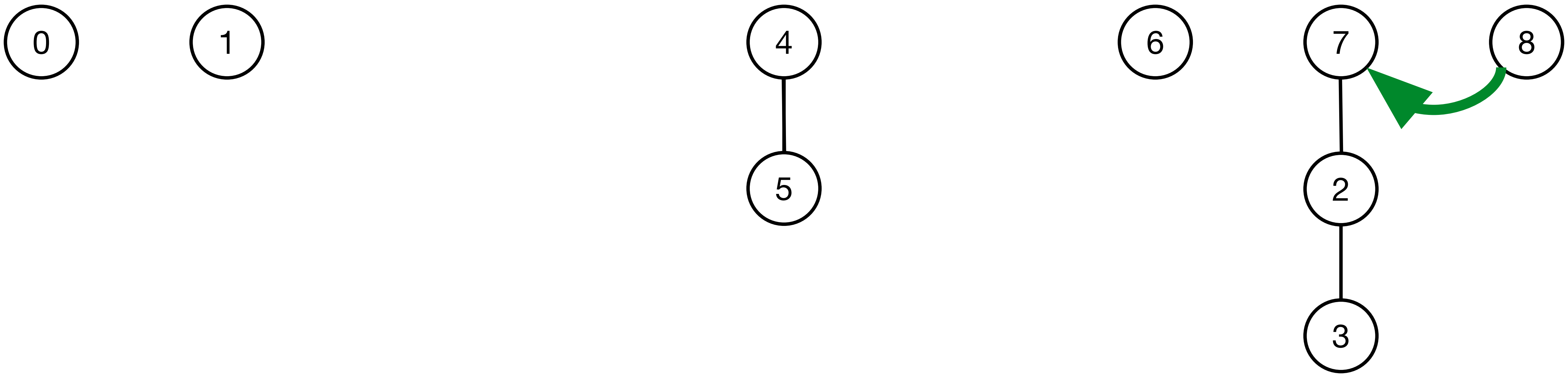
UNION(5,4)



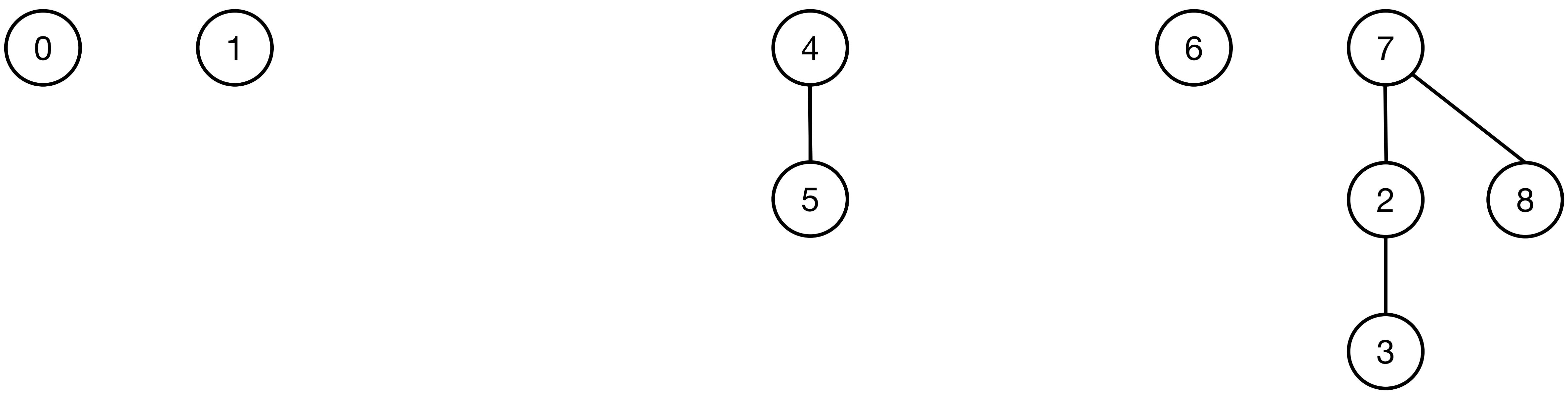
UNION(5,4)



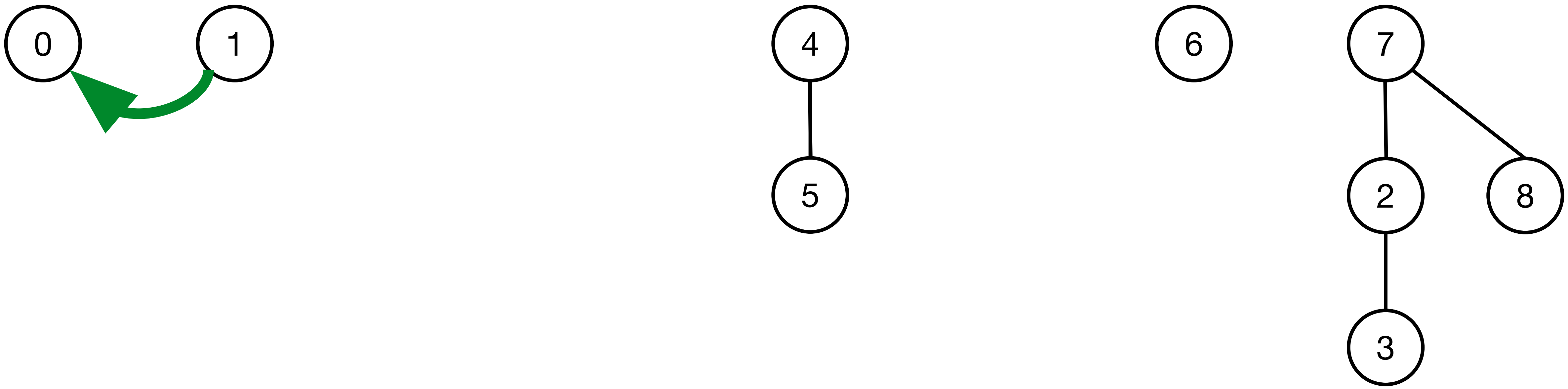
UNION(8,3)



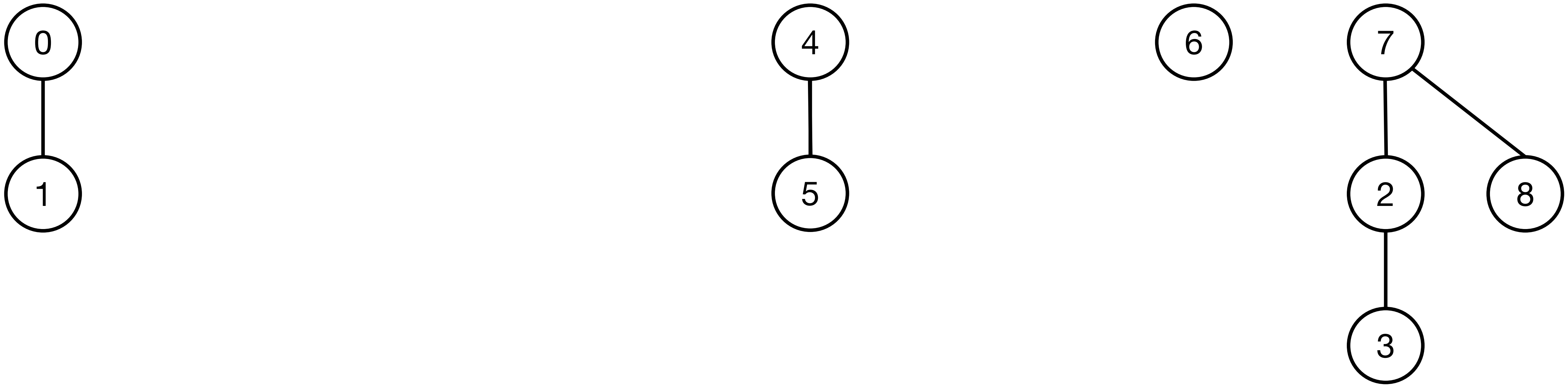
UNION(8,3)



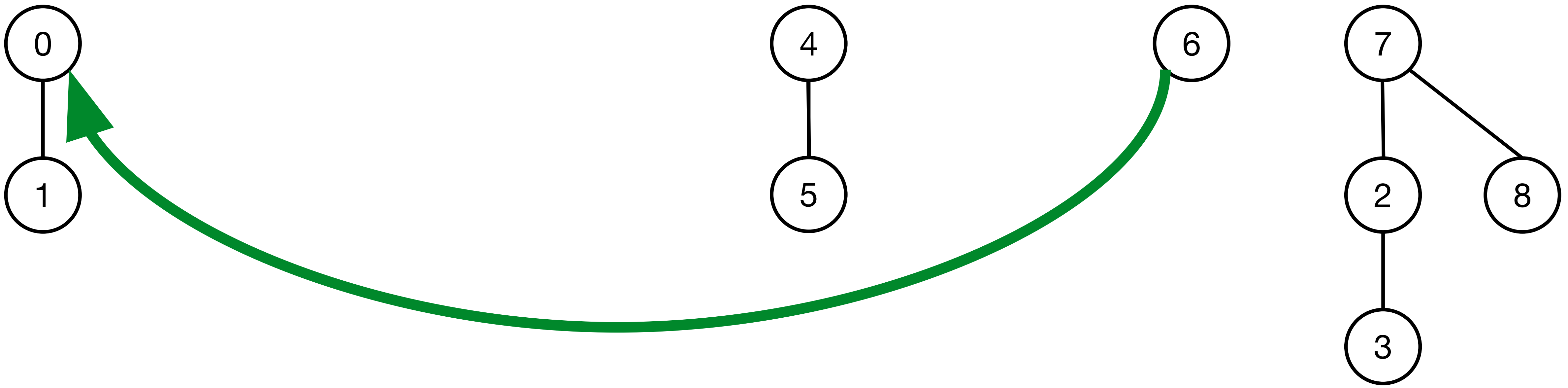
UNION(1,0)



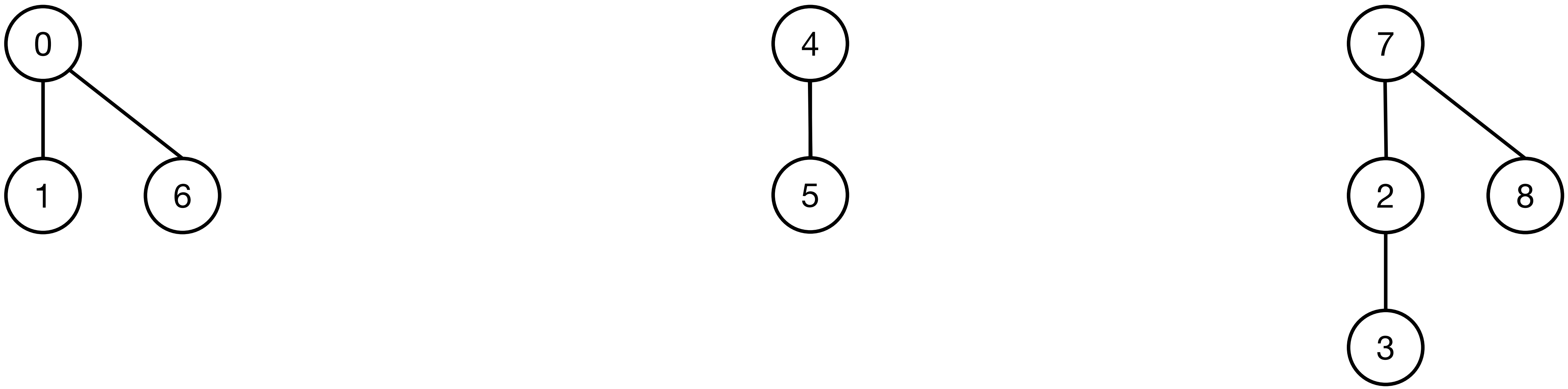
UNION(1,0)



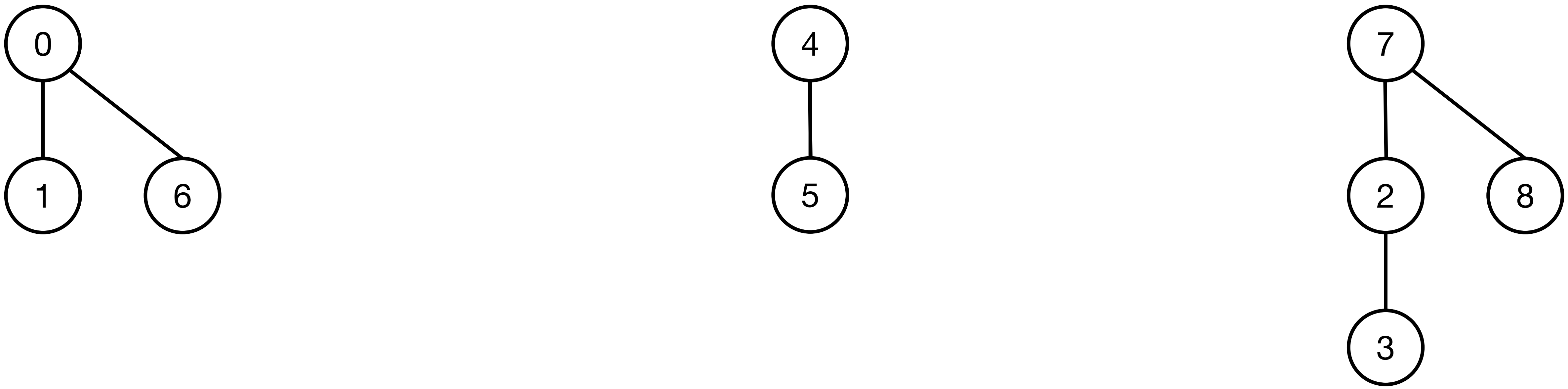
UNION(6,1)



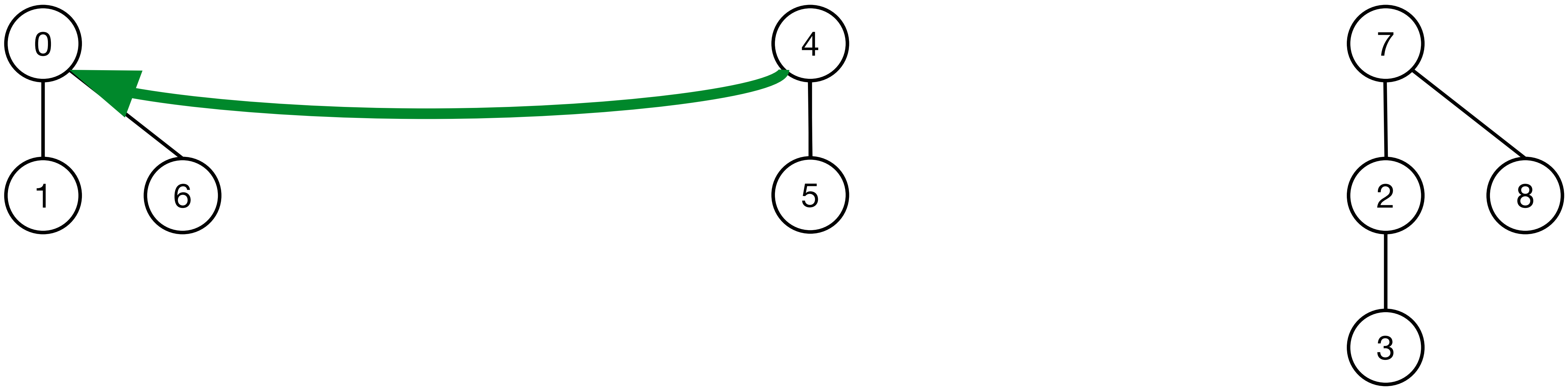
UNION(6,1)



UNION(7,3)

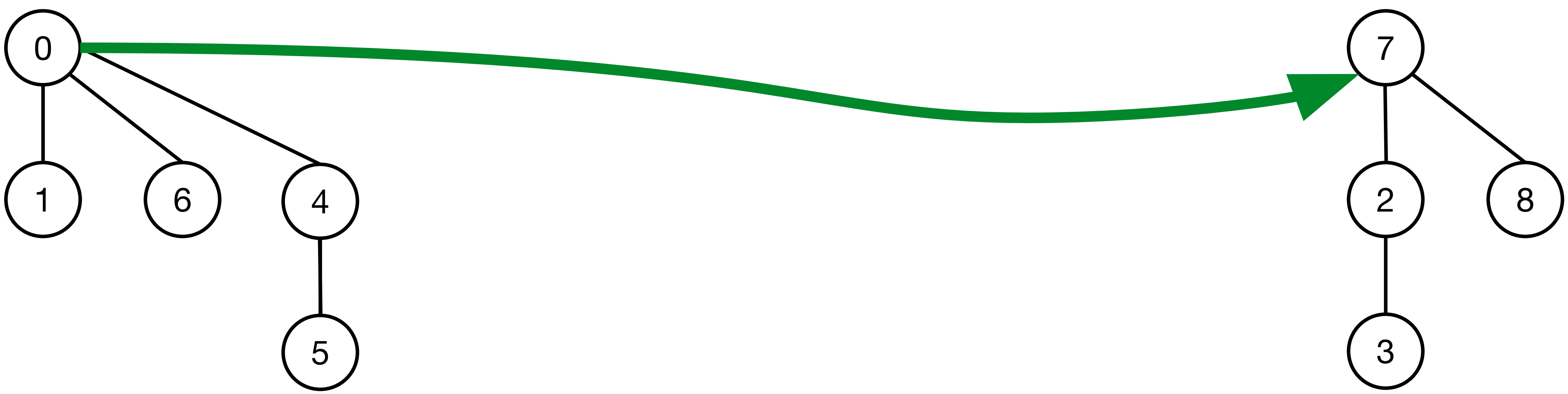


UNION(5,0)

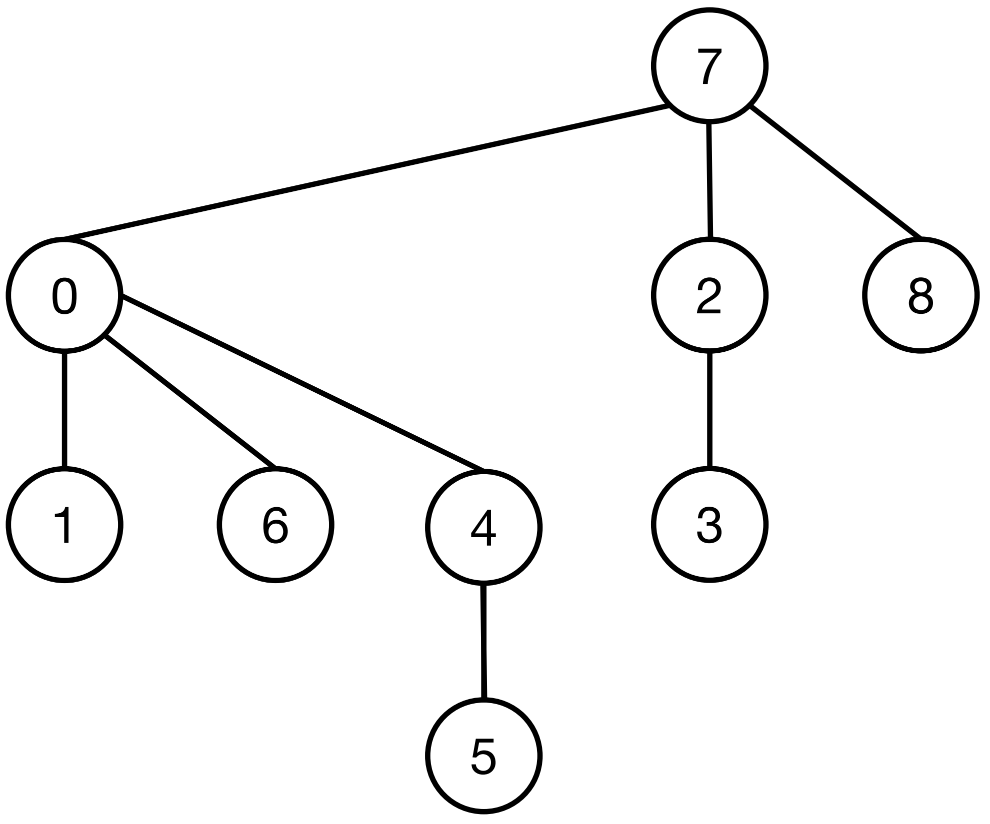


UNION(5,0)

UNION(6,2)



UNION(6,2)



Quick Union

- INIT(n): erzeuge n Bäume mit je einem Element.
- UNION(i,j): wenn FIND(i) ≠ FIND(j), mache die Wurzel des einen Baums das Kind der Wurzel des anderen Baums.
- FIND(i): folge dem Pfad zur Wurzel und liefere die Wurzel.

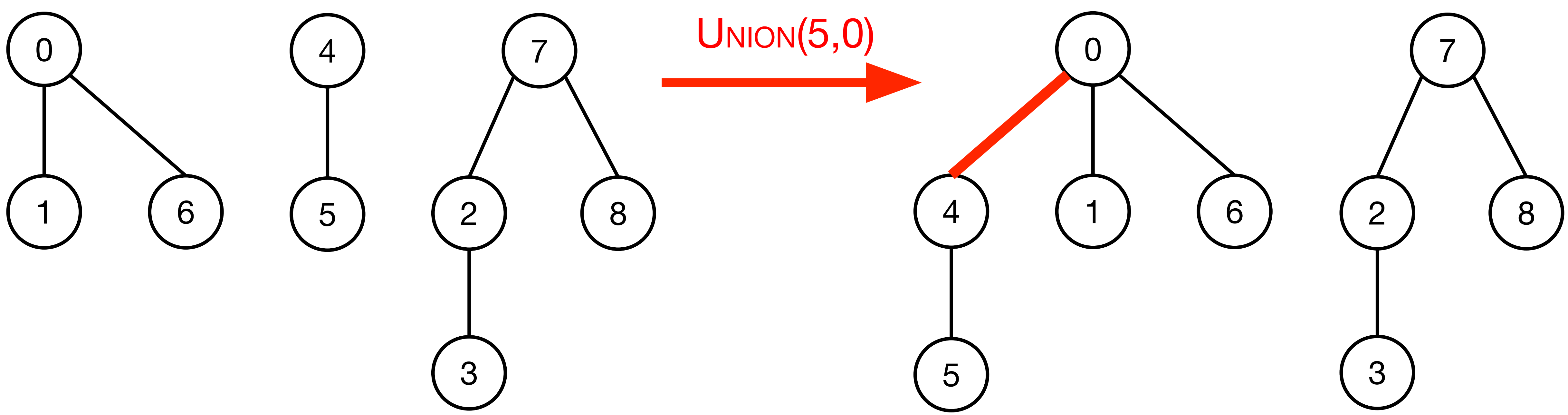
- Übung. Zeichne die Datenstruktur nach jeder Operation in folgendem Ablauf:
  - Init(7), Union(0,1), Union(2,3), Union(5,1), Union(5,0), Union(0,3), Union(5,2), Union(4,3), Union(4,6).

Quick Union

```
Init(n):
  for i = 0 to n-1
    p[i] = i
```

```
Find(i):
  while (i != p[i])
    i = p[i]
  return i
```

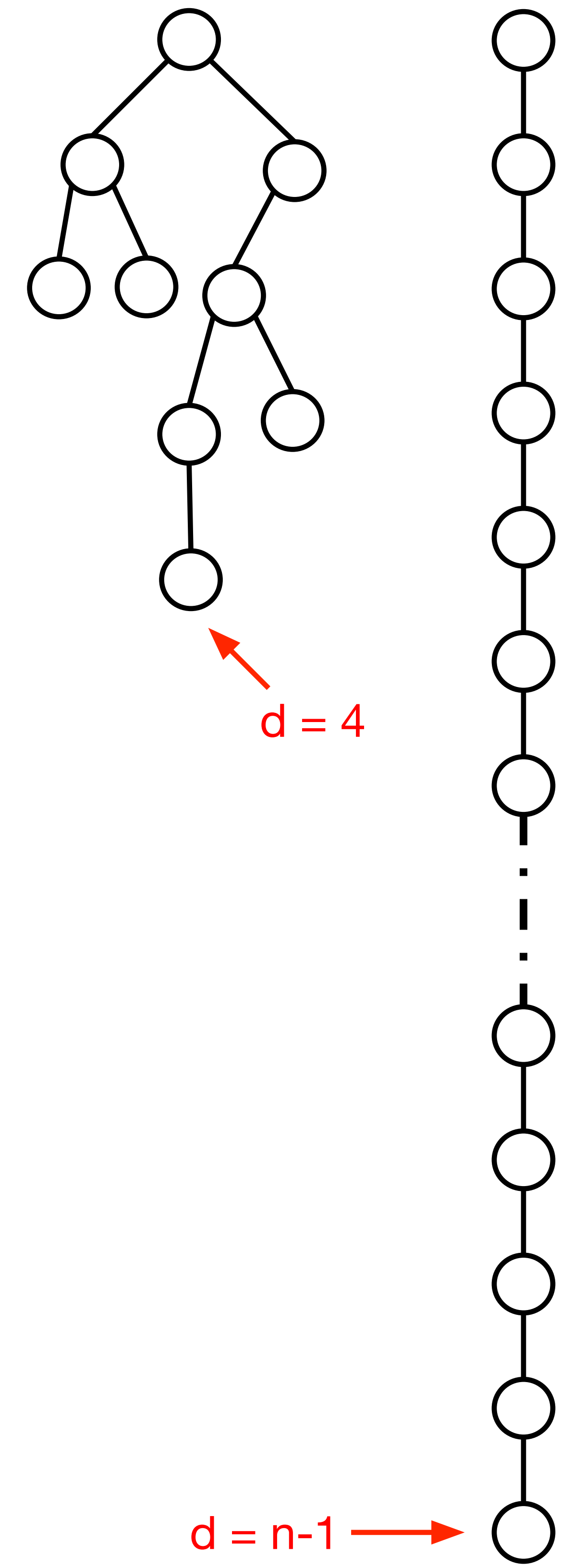
```
Union(i,j):
  r_i = Find(i)
  r_j = Find(j)
  if (r_i ≠ r_j)
    p[r_i] = r_j
```



- Zeit.
  - O(n) Zeit für Init, O(d) Zeit für Union und Find, wobei d die Tiefe des Baums ist.

Quick Union

- UNION und FIND hängen von der Tiefe des Baums ab.
- Schlechte Neuigkeiten. Tiefe könnte bis zu n-1 sein.
- Frage. Können wir Bäume so kombinieren, dass die Tiefe relativ klein bleibt?

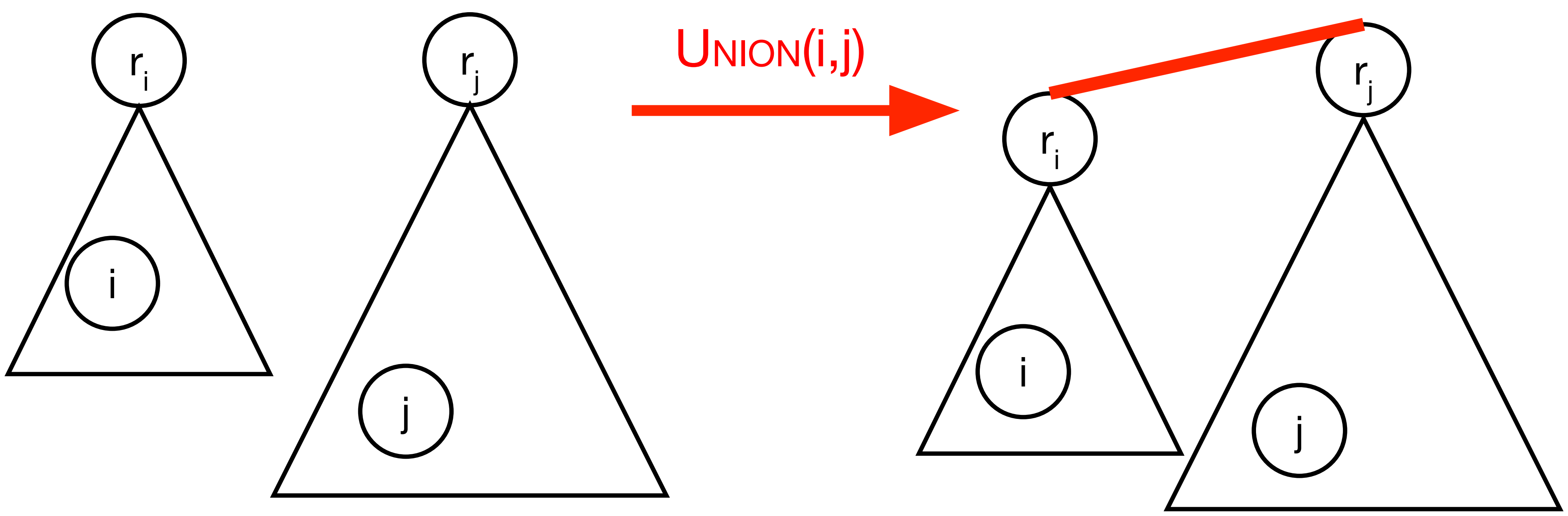


Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

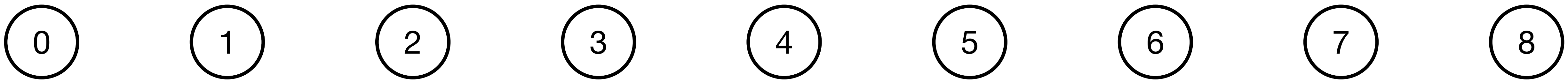
Weighted Quick Union

- Weighted quick union. Erweiterung von quick union.
- Verwalte Extra-Feld sz[0..n-1], sodass sz[i] = die Größe des Teilbaums unter i.
  - INIT: wie vorher & initialisiere sz[0..n-1].
  - FIND: wie vorher.
  - UNION(i,j): wenn FIND(i) ≠ FIND(j), mach die Wurzel des kleineren Baums zum Kind der Wurzel des größeren Baums.
- Intuition. UNION balanciert die Bäume.

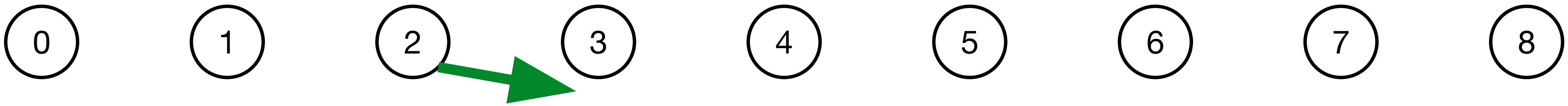


$$sz[r_j] = sz[r_i] + sz[r_j]$$

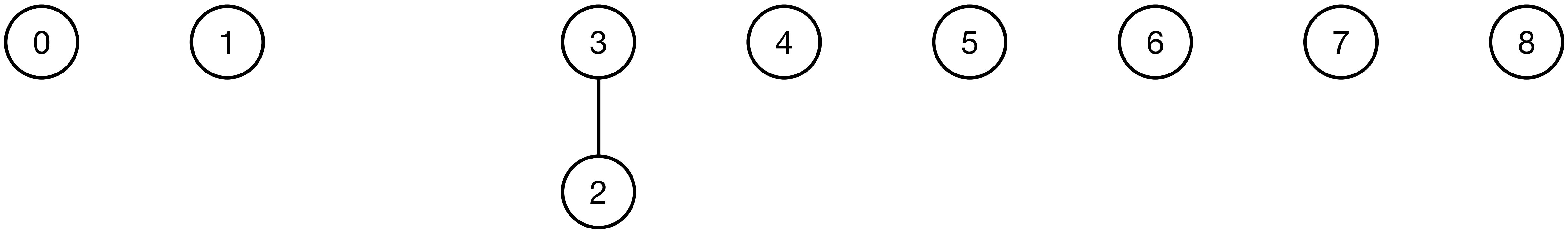
INIT(9)



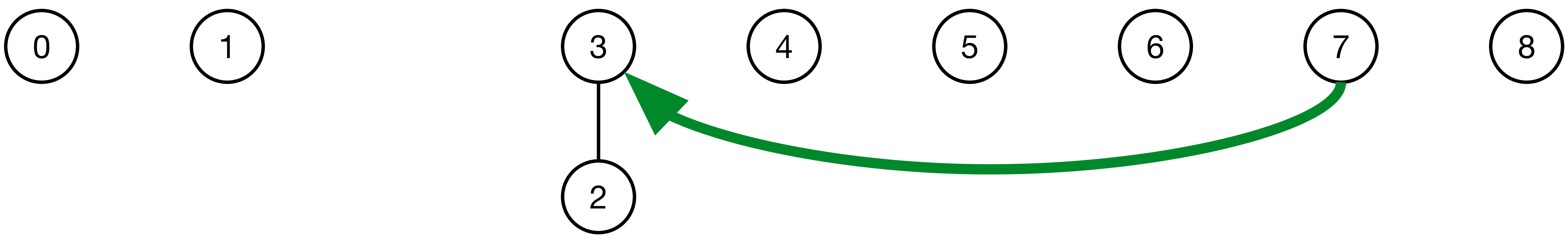
UNION(3,2)



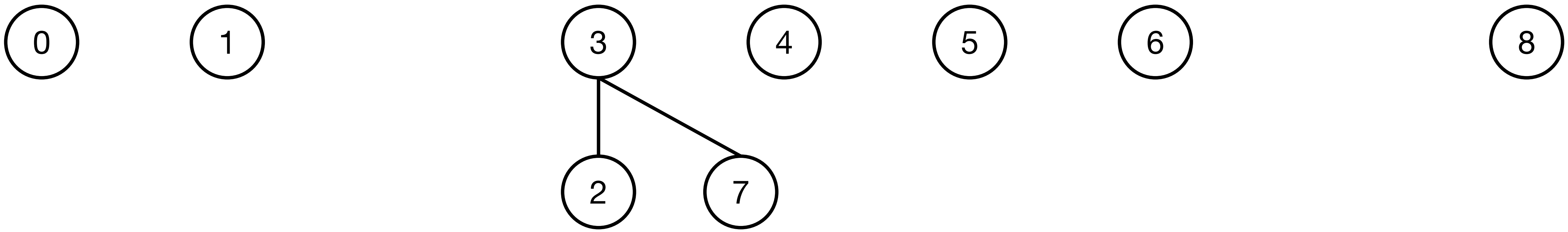
UNION(3,2)



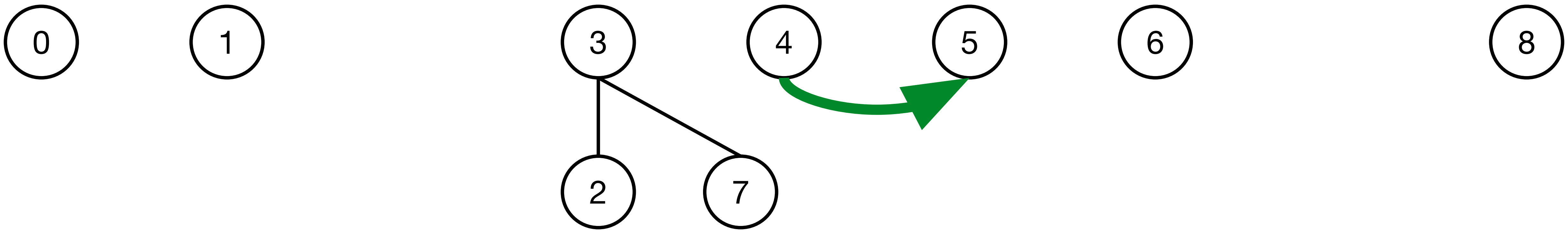
UNION(2,7)



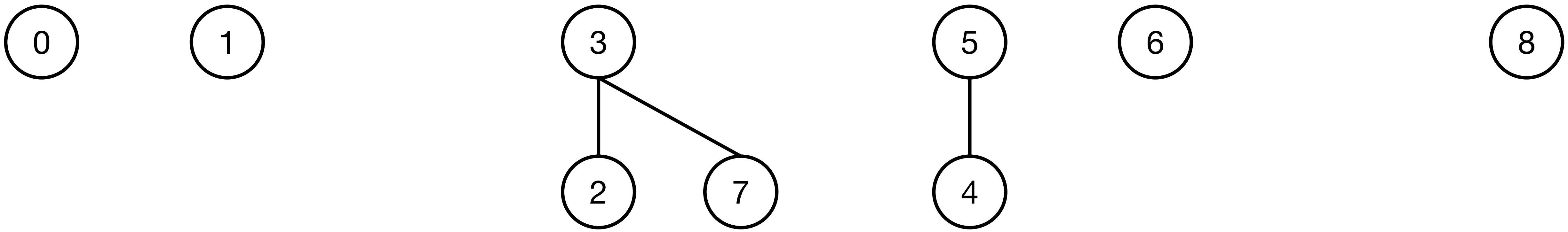
UNION(2,7)



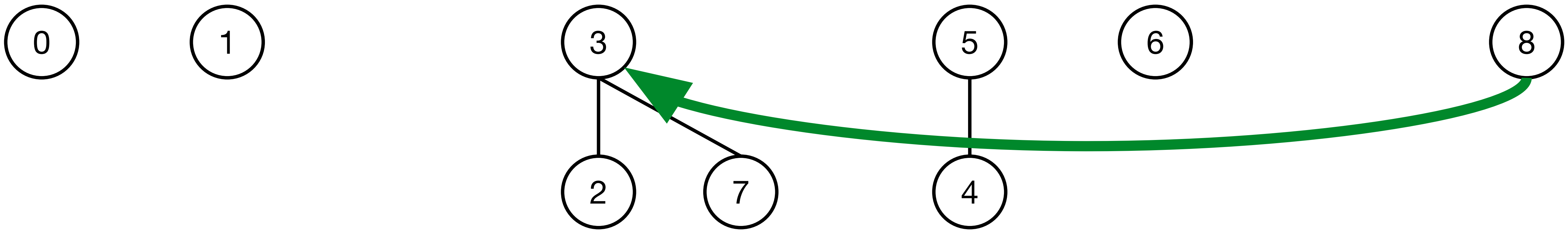
UNION(5,4)



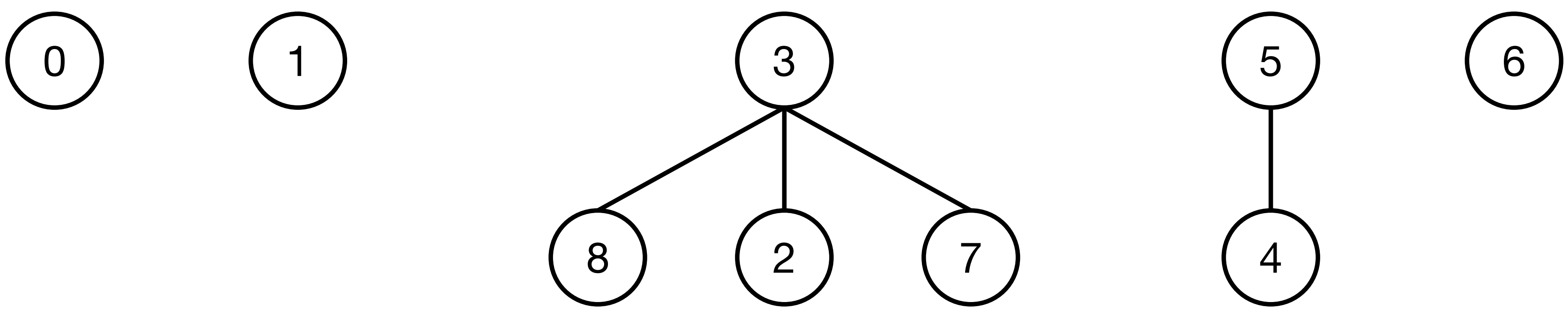
UNION(5,4)



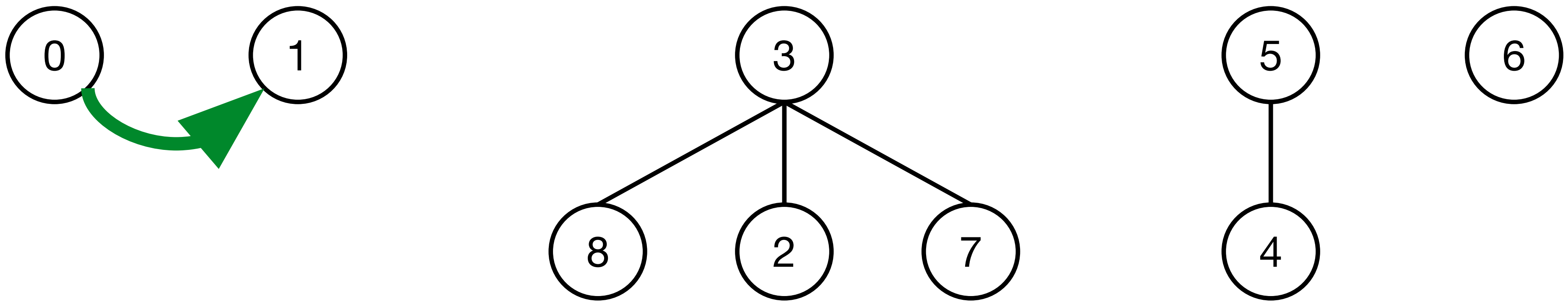
UNION(8,3)



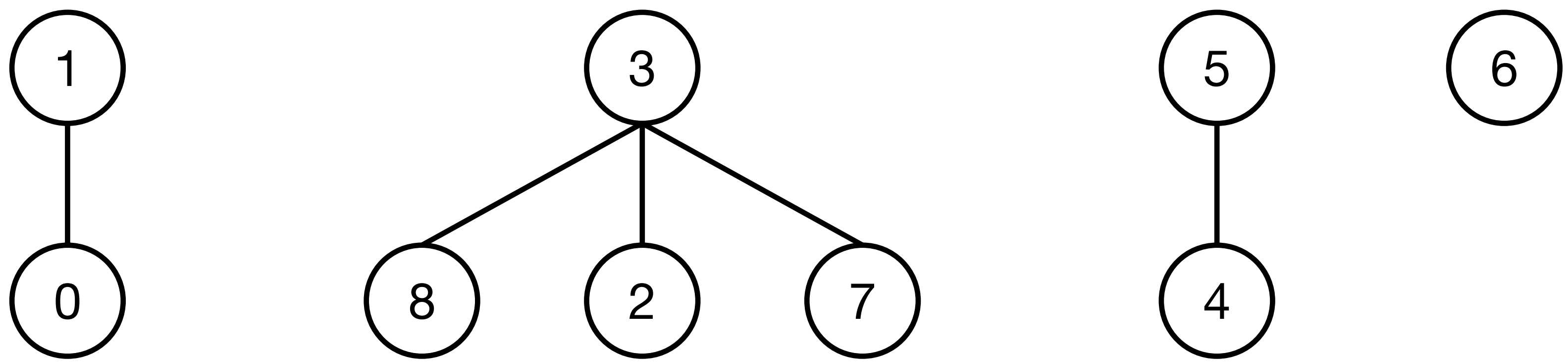
UNION(8,3)



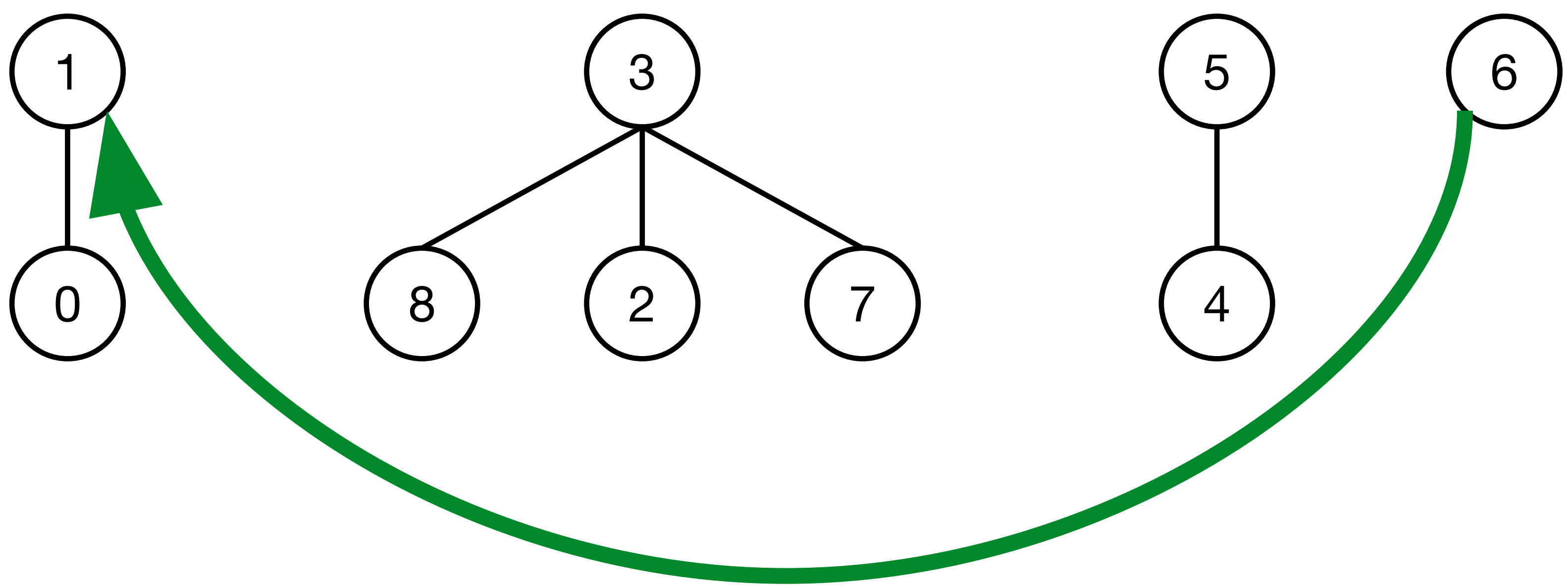
UNION(1,0)



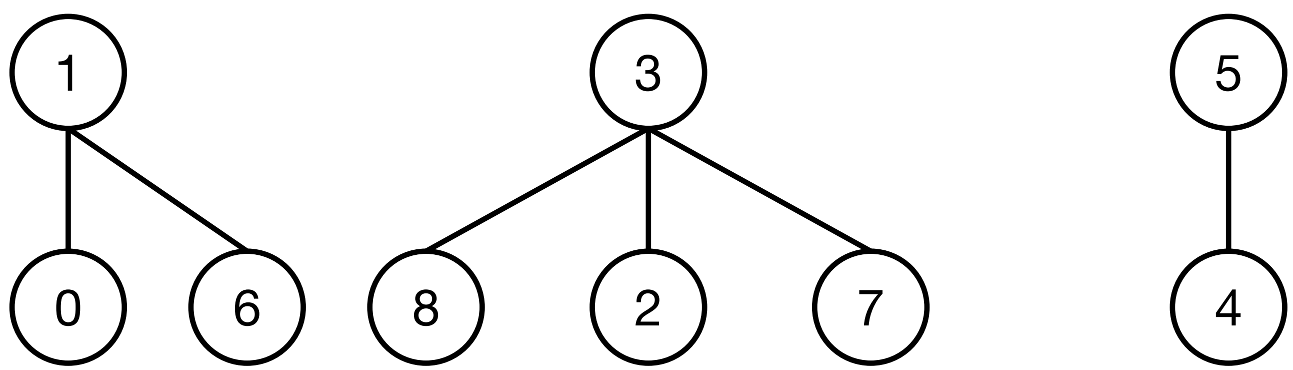
UNION(1,0)



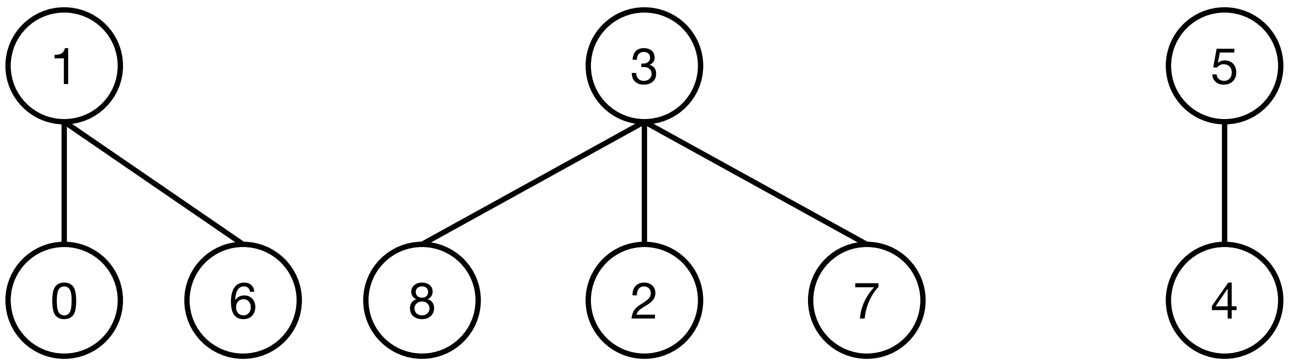
UNION(6,1)



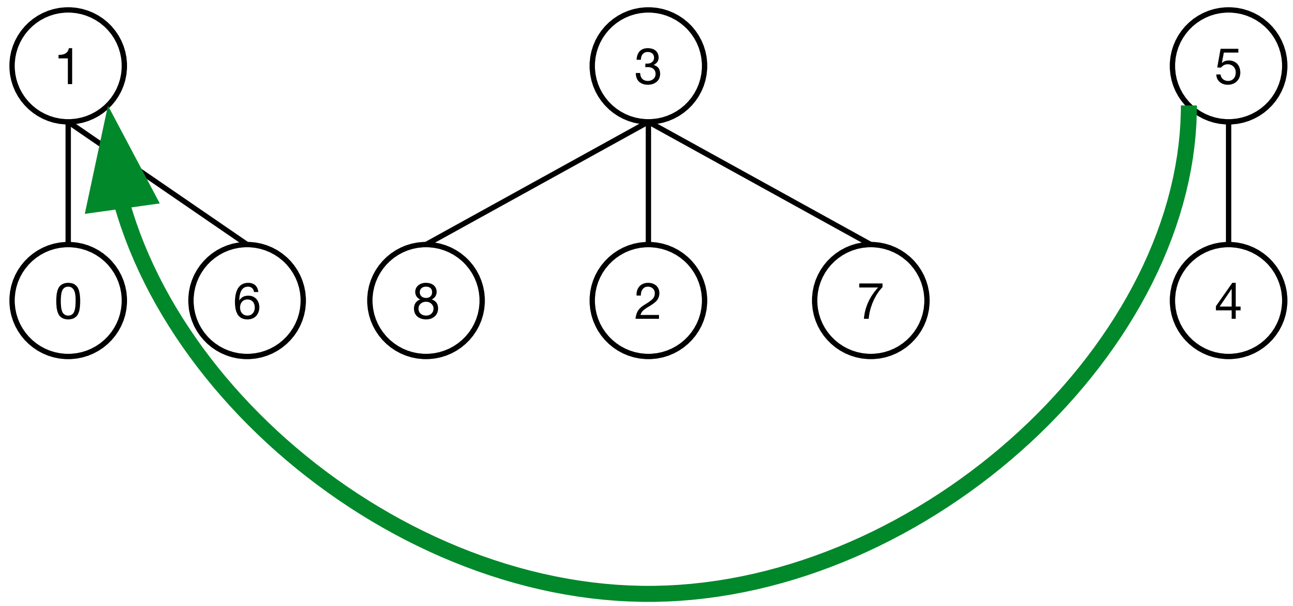
UNION(6,1)



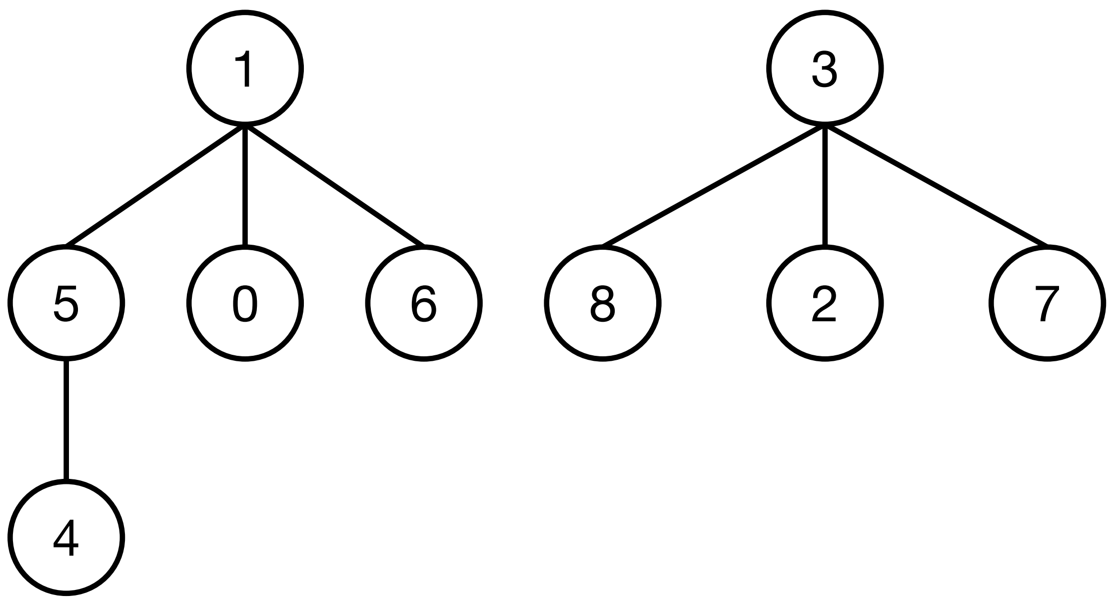
UNION(7,3)



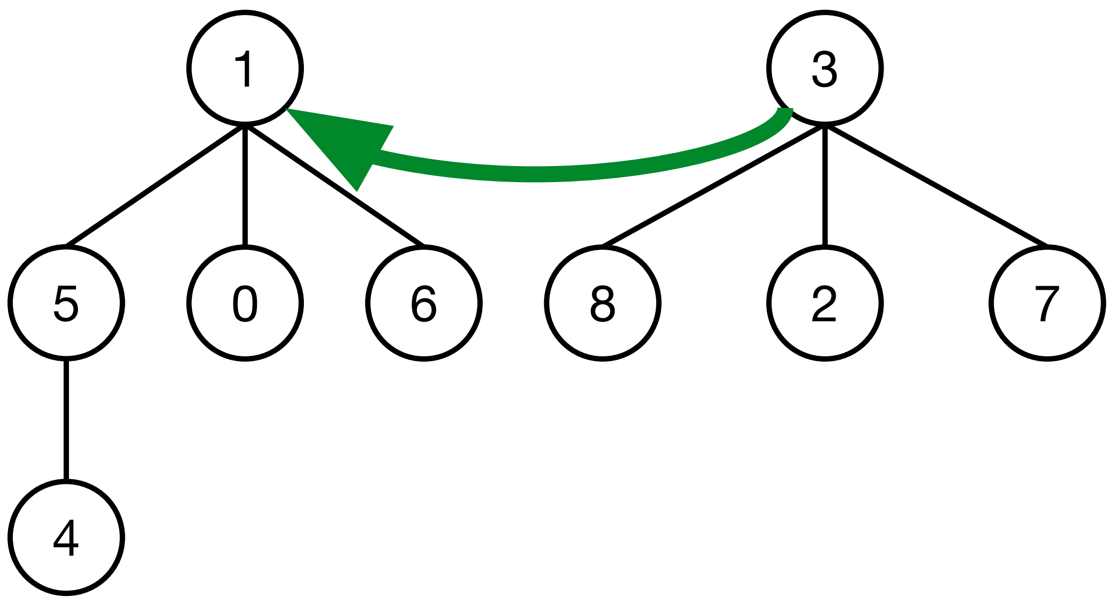
UNION(5,1)



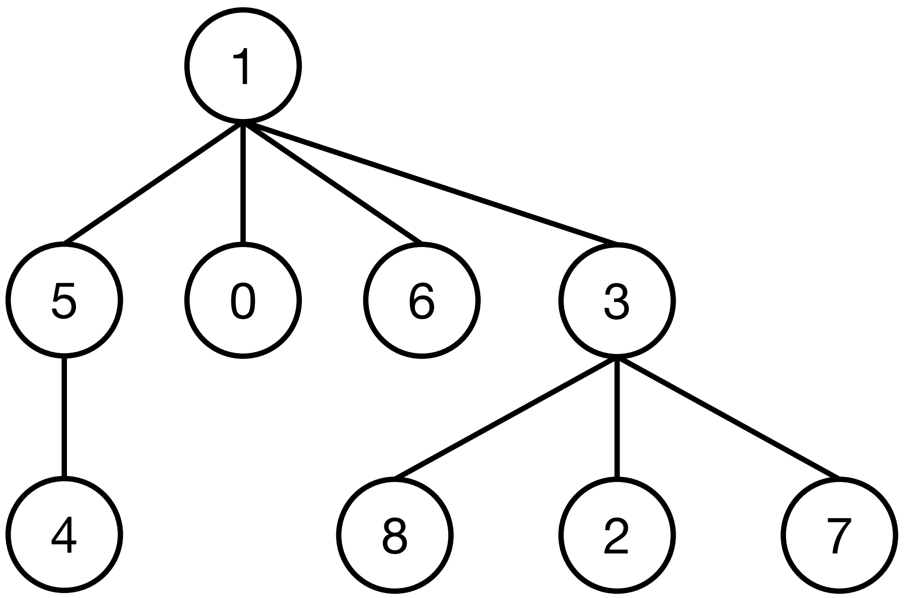
UNION(5,1)



UNION(6,3)

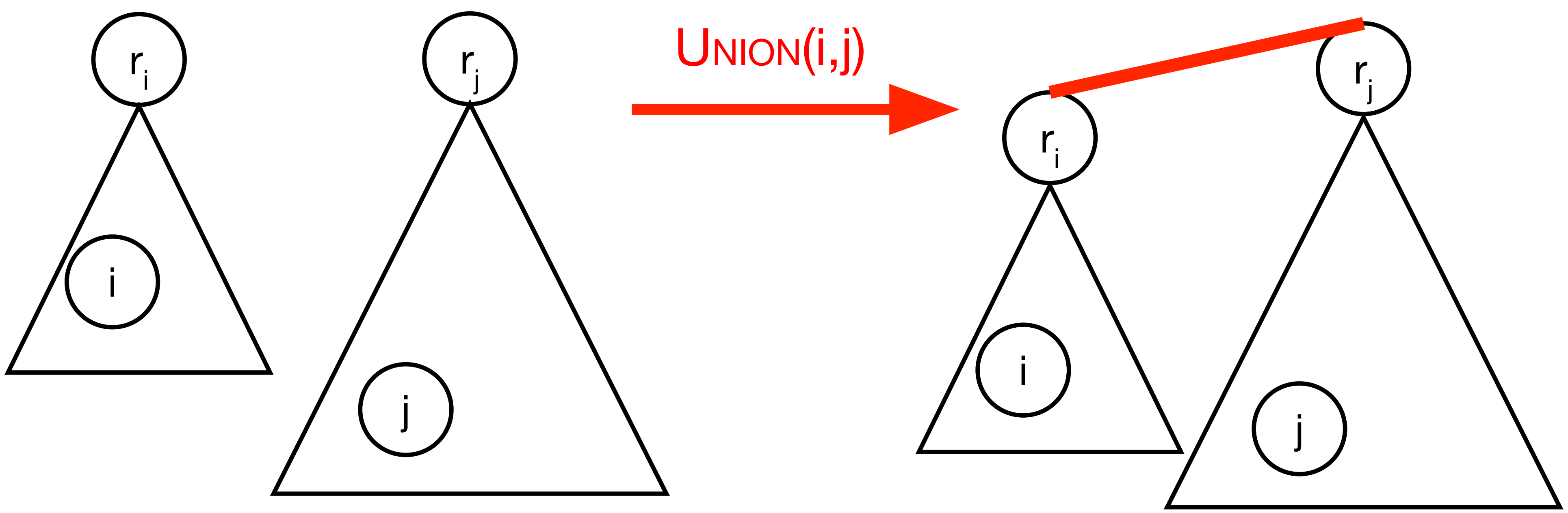


UNION(6,3)



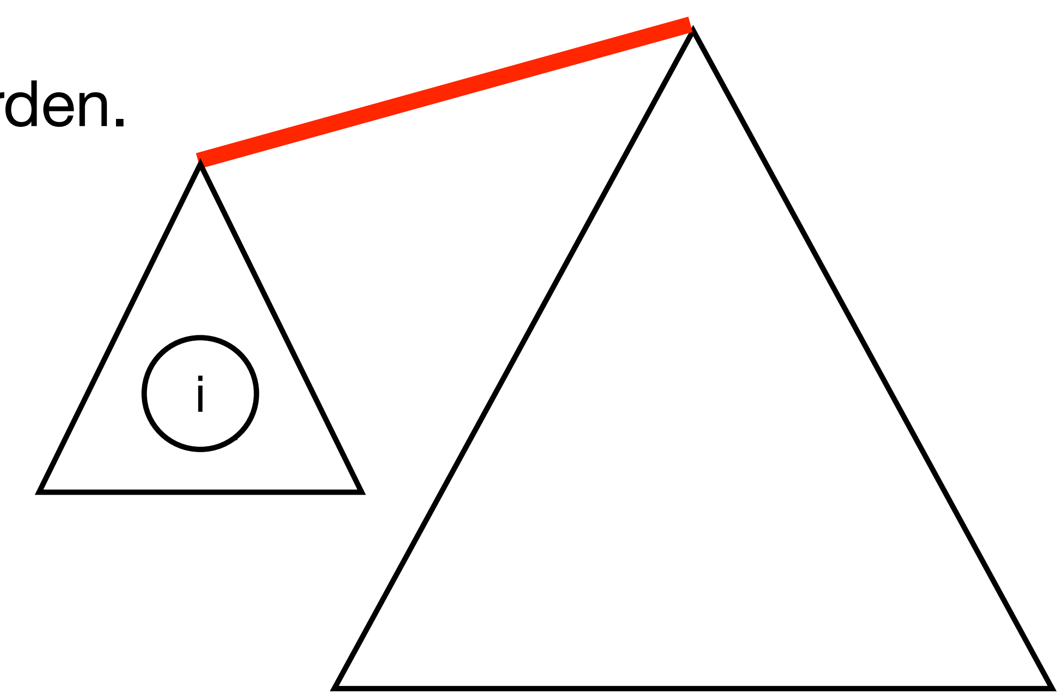
Weighted Quick Union

```
UNION(i,j):
   r_i = FIND(i)
   r_j = FIND(j)
   if (r_i ≠ r_j)
      if (sz[r_i] < sz[r_j])
         p[r_i] = r_j
         sz[r_j] = sz[r_i] + sz[r_j]
      else
         p[r_j] = r_i
         sz[r_i] = sz[r_i] + sz[r_j]
```



Weighted Quick Union

- **Lemma.** Mit Weighted Quick Union bleibt die Tiefe jedes Knoten immer $\leq \log_2 n$.
- **Beweis.**
  - Betrachte Knoten i in Tiefe d_i.
  - Anfangs $d_i = 0$.
  - d_i erhöht sich um 1 wenn jedesmal wenn der Baum mit einem größeren Baum kombiniert wird.
  - Der kombinierte Baum ist mindestens **zweimal** so groß.
  - Die Größe der Bäume kann höchstens $\log_2 n$ mal verdoppelt werden.
  - $\Longrightarrow d_i \leq \log_2 n$.



Union Find

Datenstruktur	UNION	FIND
quick find	O(n)	O(1)
quick union	O(n)	O(n)
weighted quick union	O(log n)	O(log n)

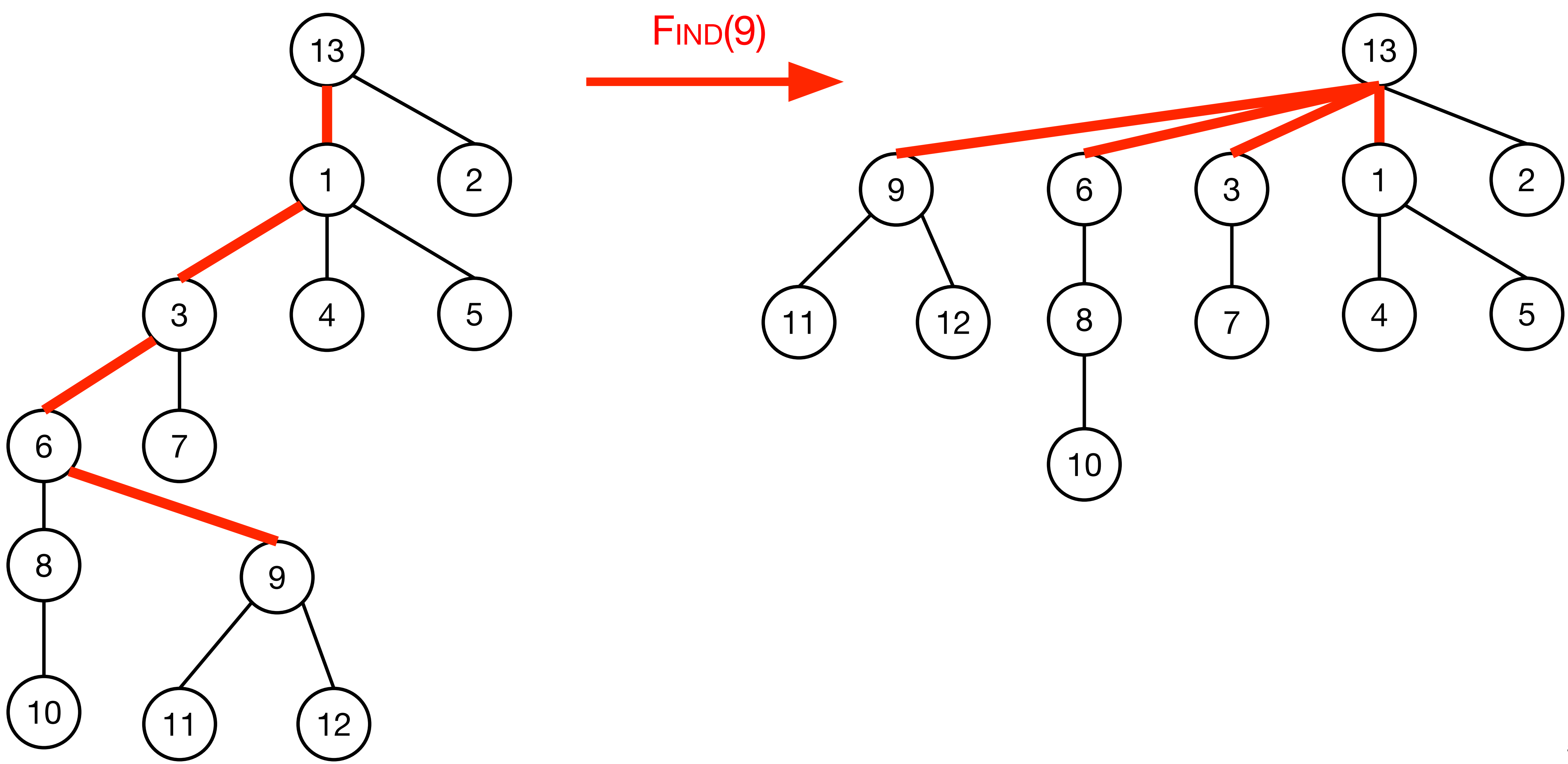
- Frage. Geht es **noch** besser?

Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

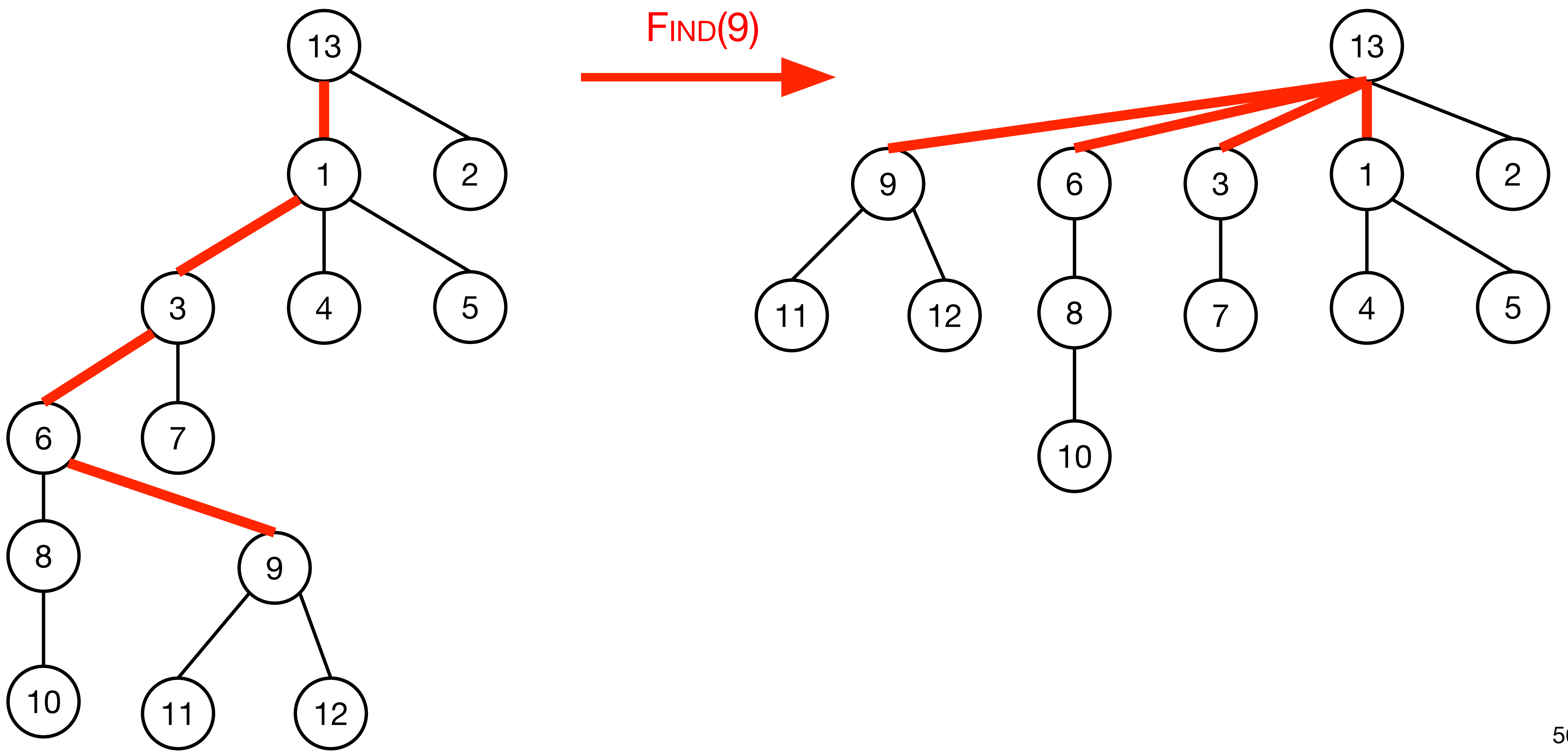
Pfadverkürzung (*path compression*)

- Pfadverkürzung. Verkürze Pfad bei FIND. Mach alle Knoten auf dem Pfad zu Kindern der Wurzel.
- Ändert die Laufzeit von einzelnem FIND nicht. Spätere FIND werden schneller.
- Funktioniert mit Quick Union und Weighted Quick Union.



Pfadverkürzung (*path compression*)

- Satz [Tarjan 1975]. Mit Pfadverkürzung braucht jede Sequenz von m FIND und UNION Operationen auf n Elementen Zeit O(n + m α(m,n)).
- Hierbei ist α(m,n) die inverse Ackermannfunktion. α(m,n) ≤ 5 gilt in der Praxis.
- Satz [Fredman-Saks 1985]. Es ist unmöglich, m FIND und UNION Operationen in Zeit O(n + m) zu implementieren.

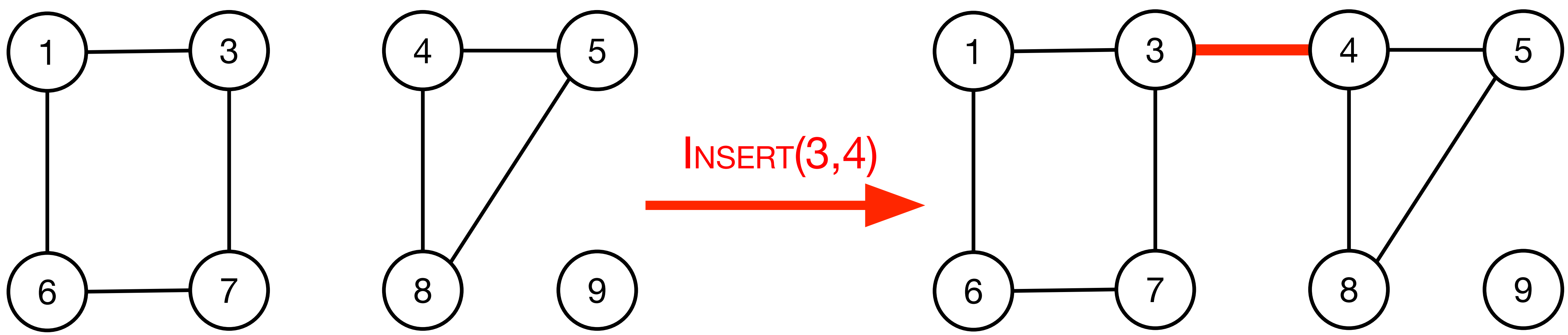


Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang

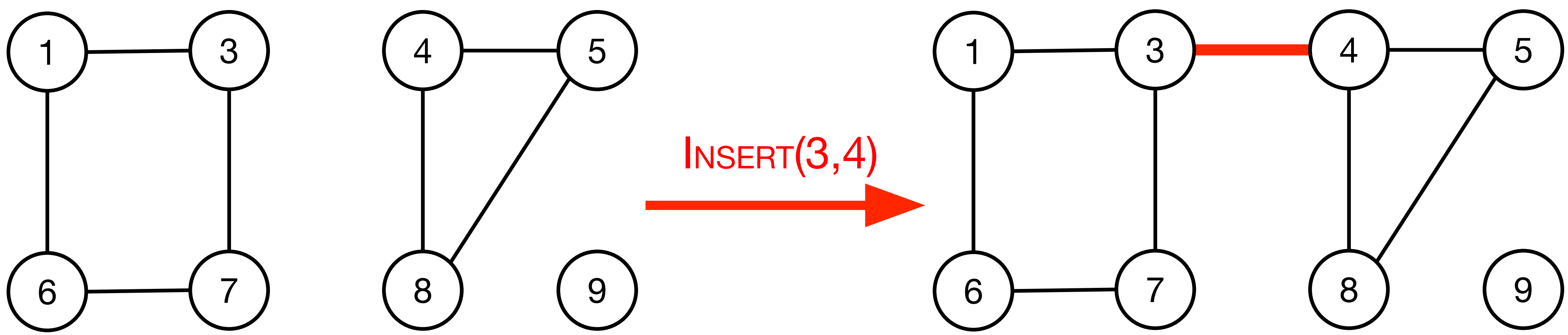
Dynamischer Zusammenhang

- Dynamischer Zusammenhang. Verwalte eine dynamischen Graphen mit den folgenden Operationen:
  - INIT(n): Erzeuge einen Graphen G mit n Knoten und 0 Kanten.
  - CONNECTED(u,v): Entscheide, ob u und v verbunden sind.
  - INSERT(u, v): Füge Kante (u,v) hinzu. Wir nehmen hierbei an, dass (u,v) noch keine Kante ist.



Dynamischer Zusammenhang

- Implementierung mit Union Find.
  - INIT(n): Initialisiere eine Union Find Datenstruktur mit n Elementen.
  - CONNECTED(u,v): FIND(u) == FIND(v).
  - INSERT(u, v): UNION(u,v)



- Zeit
  - O(n) für INIT, O(log n) für CONNECTED und O(log n) für INSERT

Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Pfadverkürzung
- Dynamischer Zusammenhang