Java II
Natural Language Algorithms in Java
Data Structures for Disjoint Sets

Bernd Kiefer

{Bernd.Kiefer}@dfki.de

Deutsches Forschungszentrum für künstliche Intelligenz

Disjoint Set Data Structures

- Problem: a set with n elements and a (total) equivalence relation $\equiv$
- Implement the following operations efficiently:
  - ➤ do elements a and b belong to the same class?
  - ➤ put a into the equivalence class of b
  - ➤ merge the equivalence classes of a and b (union)
- assume the elements are numbered consecutively
- use a vector $\mathcal{V}$ of n elements containing integers
- if $\mathcal{V}[n] = n$, n is the *representative* of the class
- otherwise, $\mathcal{V}[n]$ points directly or indirectly to the representative

<u>**proc**</u> *find-representative*$(a)\ \equiv$

  <u>**while**</u> $\mathcal{V}[a] \neq a$ <u>**do**</u> $a := \mathcal{V}[a]$

  return a

<u>**proc**</u> *equiv*$(a, b)\ \equiv$

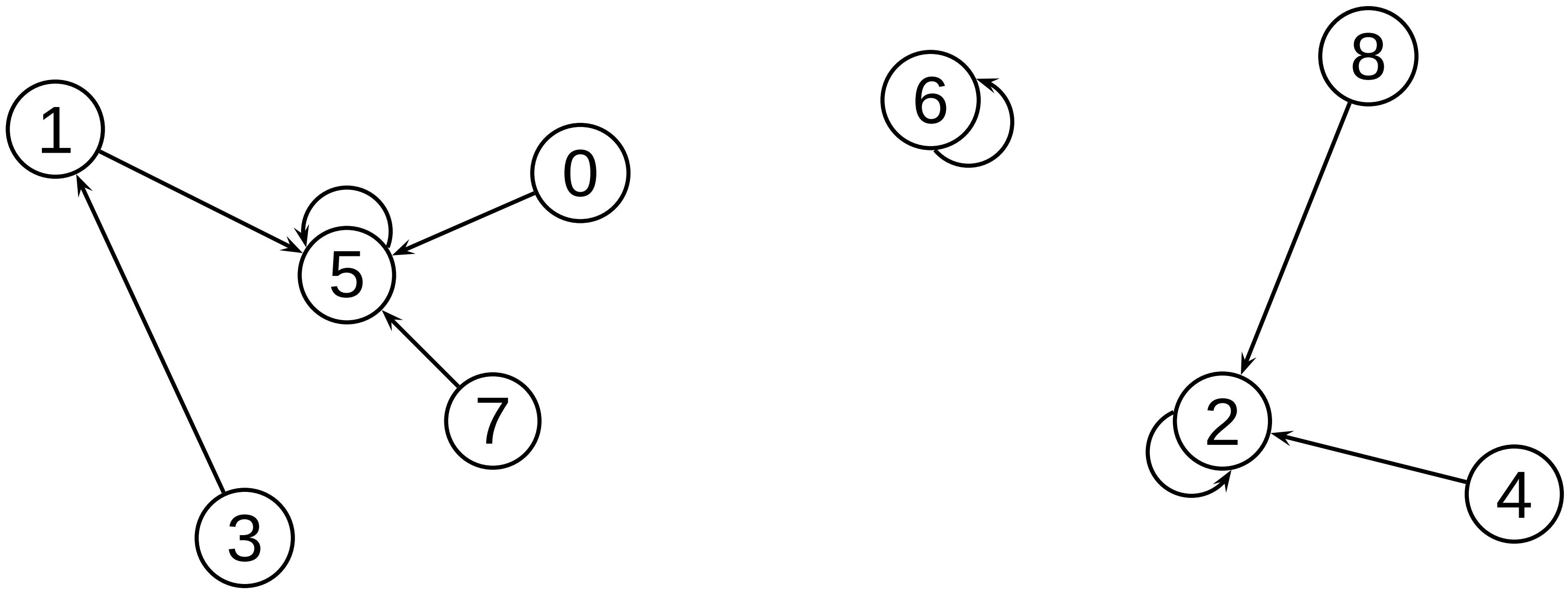
  return *find-representative*$(a) =$ *find-representative*(b)

<u>**proc**</u> *union*(a, b)

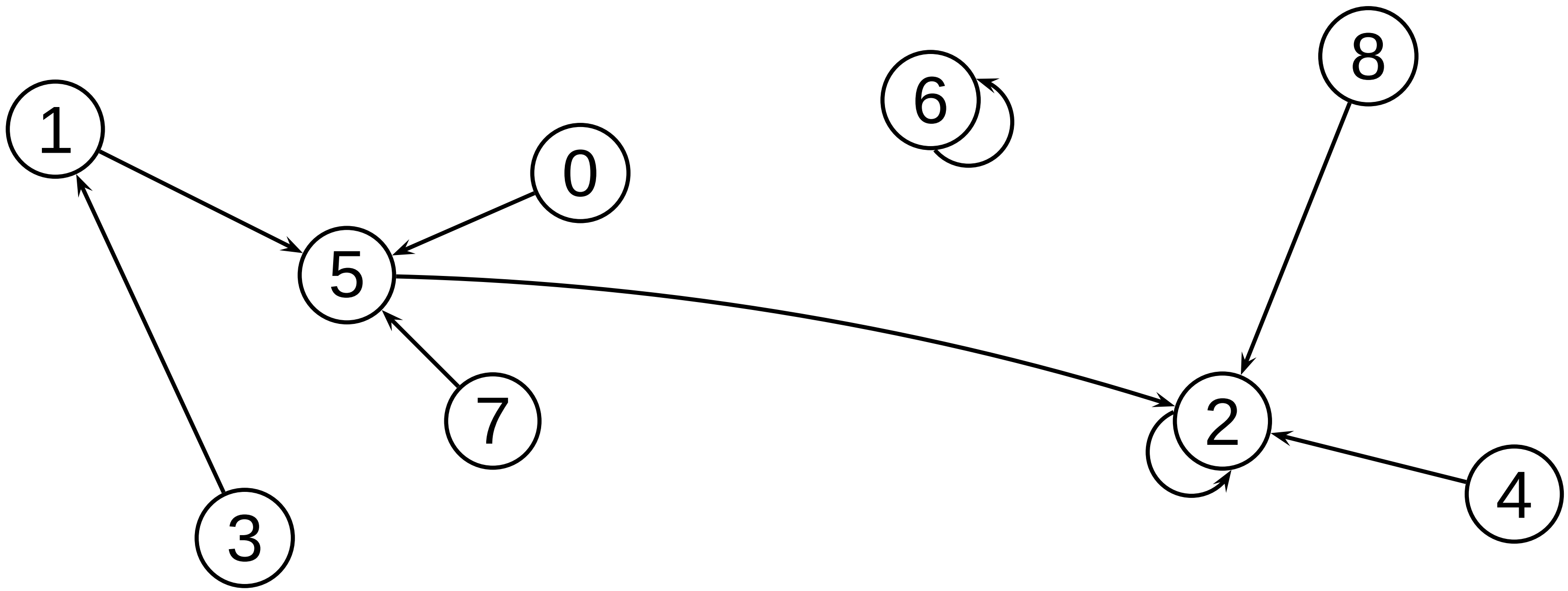
  $a :=$ *find-representative*(a)

  $\mathcal{V}[a] :=$ *find-representative*(b)

Example I

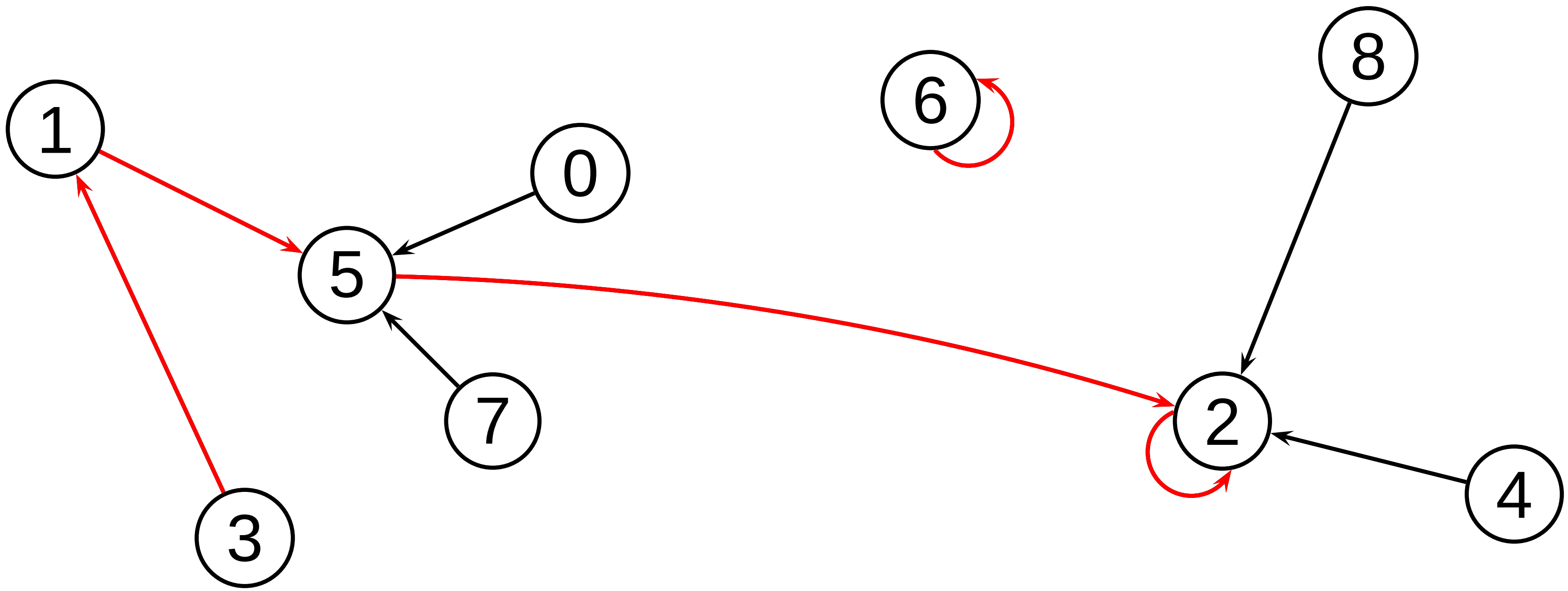


union$(3, 8)$



union$(3, 8)$

Example I



union$(3, 8)$
equiv$(6, 3)$

Improving Asymptotic Complexity

- the tree can degenerate into a spine of length $O(n)$
- idea: use the freedom in merging two sets
  - for every representative, maintain the size of the set it represents
  - always merge the smaller set into the bigger
  - instead maintaining the rank (an approximation of the tree height) gives the same asymptotic results
  - Any tree of height h must then at least containt 2^h elements
- additionaly, shorten the paths during each equiv operation

Improved Implementation

proc *find-representative*$(a) \;\equiv$

  while $\mathcal{V}[a] \neq \mathcal{V}[\mathcal{V}[a]]$ **do**

    $a := \mathcal{V}[a] := \mathcal{V}[\mathcal{V}[a]]$ // path compression

  return $\mathcal{V}[a]$

proc *union*(a, b)

  $a :=$ *find-representative*(a)

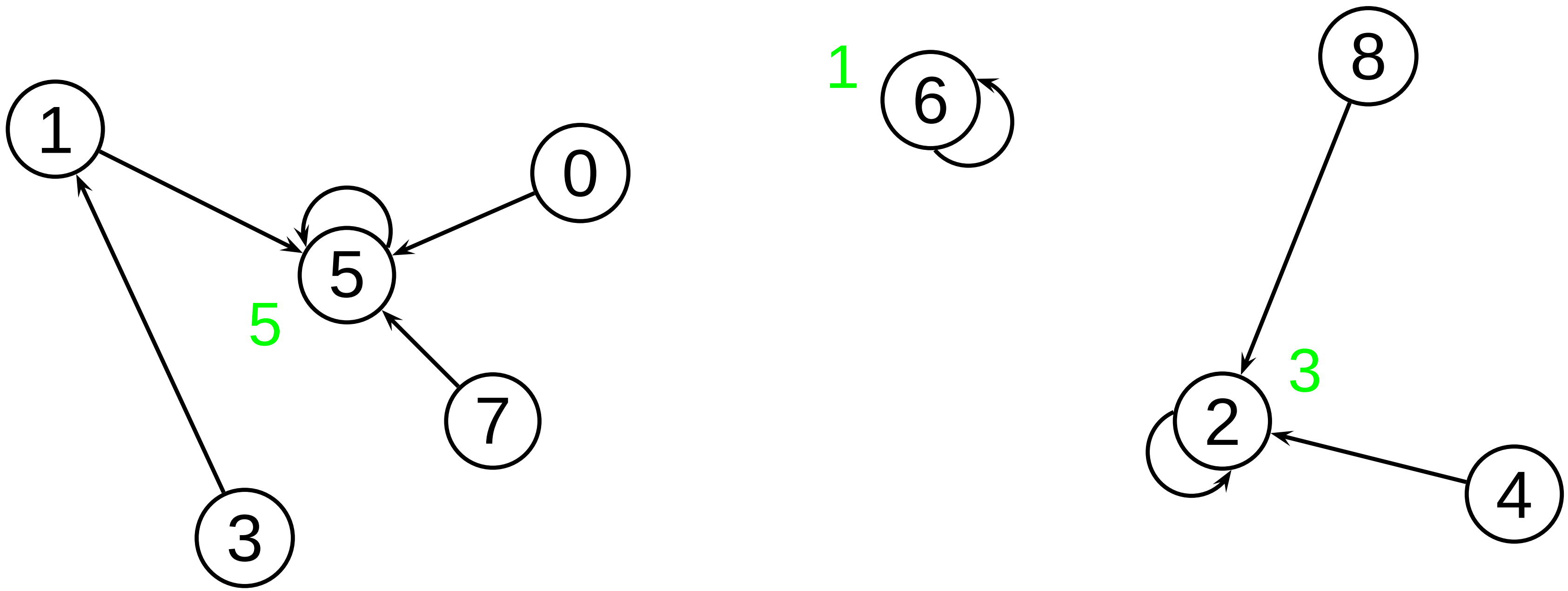
  $b :=$ *find-representative*(b)

  if *size*$(a) >$ *size*(b) **then**

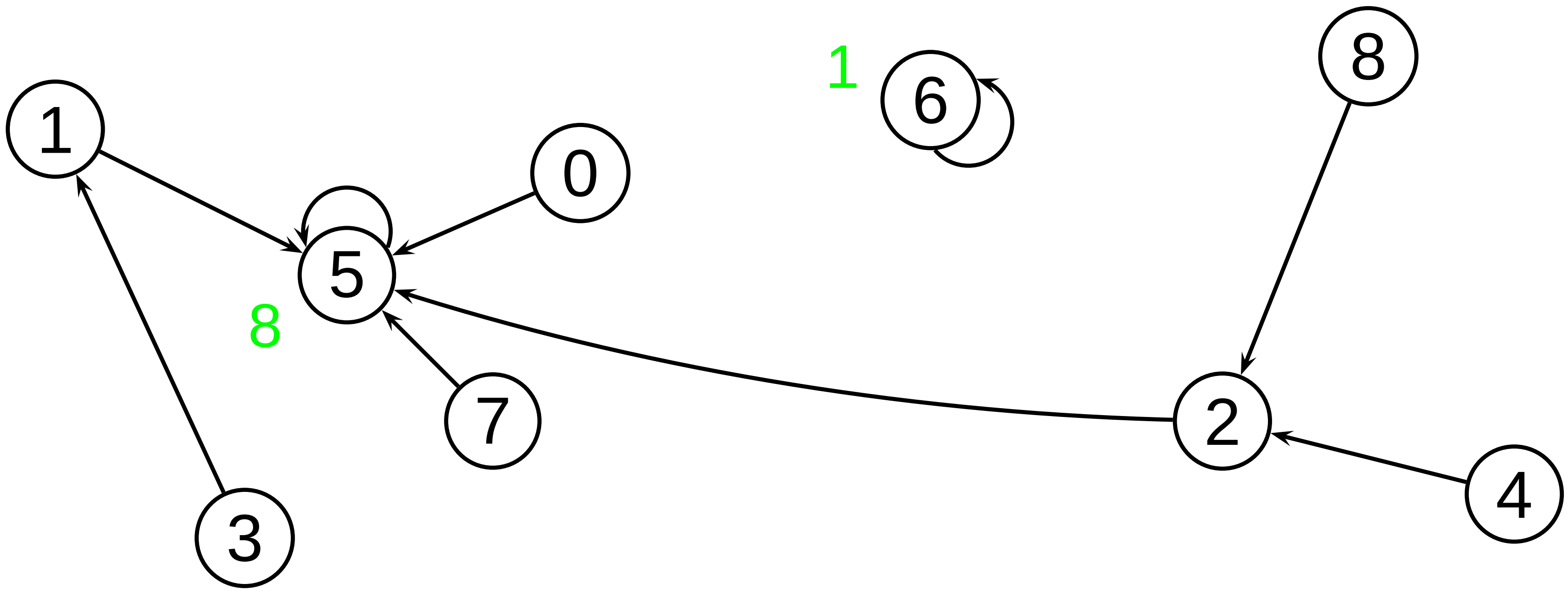
    exchange(a, b) // merge b into a

  $V[a] := b$ // merge a into b

  size$(b) =$ *size*$(a) +$ *size*(b)

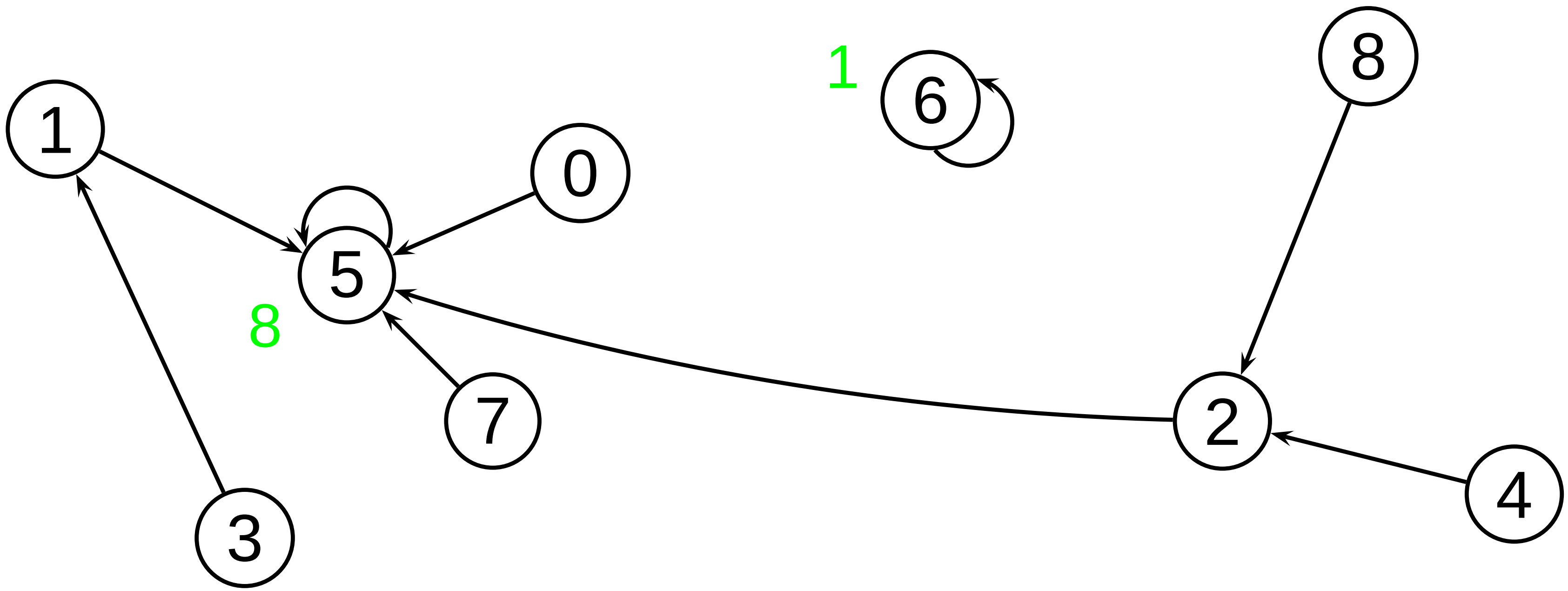


union$(7, 8)$

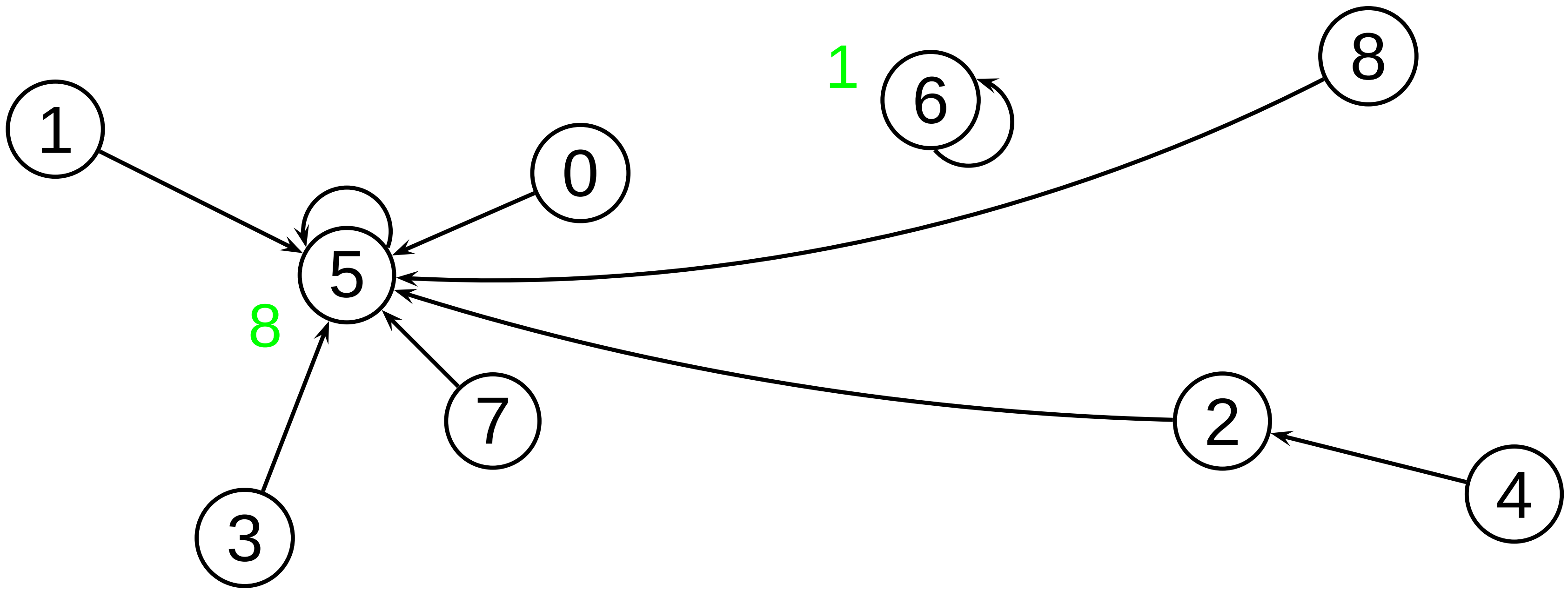


union$(7, 8)$

Size + Path Compression



union$(7, 8)$
equiv$(3, 8)$



union$(7, 8)$
equiv$(3, 8)$