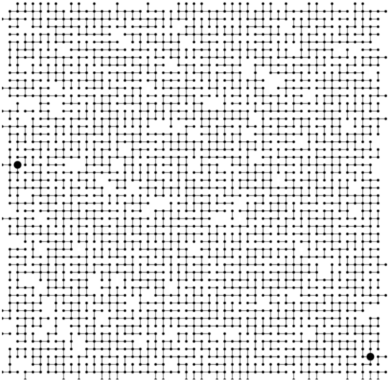
Data Structures for Disjoint Sets

Bernd Kiefer
Jörg Steffen

January 20, 2023

Application: Network Connectivity

- An underlying, unexplored undirected graph
- union: connect two objects
- find query: is there a connection between two objects



Disjoint Set Data Structures: What for?

A set of 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)
- Every union operation reduces the set of classes by one

Disjoint Set Data Structures: What for?

A set of 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)
- Every union operation reduces the set of classes by one

Examples for $\equiv$

- Connected computers in a network
- Variables pointing to the same object in memory (e.g., for garbage collection)
- Similarly colored pictures in a digital image
- Coreferences of feature structures during unification

Some Abstractions

- Objects

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

- Disjoint sets of objects

0	1	{3	5	7}	{6	2}	4	{8	9}	sets of connected points

- Find query: are objects 2 and 9 in the same set?

0	1	{2	3	9}	{5	6}	7	{4	8}	are two points connected?

- Union: merge sets containing 3 and 8

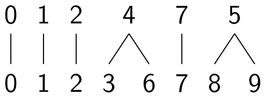
0	1	{2	3	9	4	8}	7	{5	6}	add a connection between two points

- We are looking at cases where n objects are involved, and m operations are performed, both n and m large!

Implementation Basics

- For programming: assume the elements are numbered consecutively
- a symbol table can be used to associate objects to numbers
- 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

	0	1	2	3	4	5	6	7	8	9
$\mathcal{V}$	0	1	2	4	4	5	4	7	5	5



Quick find

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

	0	1	2	3	4	5	6	7	8	9
$\mathcal{V}$	0	1	2	4	4	5	4	7	5	5

Quick find

- $\texttt{find}(a, b) : \mathcal{V}[a] == \mathcal{V}[b]$
- Problem: Merge may require many changes, e.g., merge 6 and 9

	0	1	2	3	4	5	6	7	8	9
$\mathcal{V}$	0	1	2	4	4	5	4	7	5	5

Quick find

- $\texttt{find}(a, b) : \mathcal{V}[a] == \mathcal{V}[b]$
- Problem: Merge may require many changes, e.g., merge 6 and 9
- Merge is linear in n

	0	1	2	3	4	5	6	7	8	9
$\mathcal{V}$	0	1	2	5	5	5	5	7	5	5

Quick Merge

find-representative(a):

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

$a := \mathcal{V}[a]$

return a

equiv(a, b):

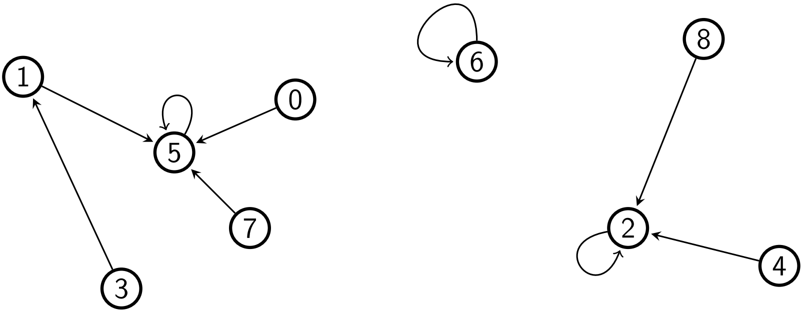
return `find-representative`$(a) =$`find-representative`(b)

union(a, b):

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

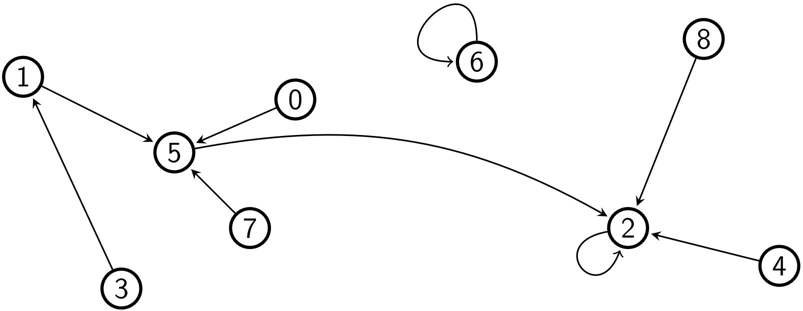
$\mathcal{V}[a] :=$ `find-representative`(b)

Example I



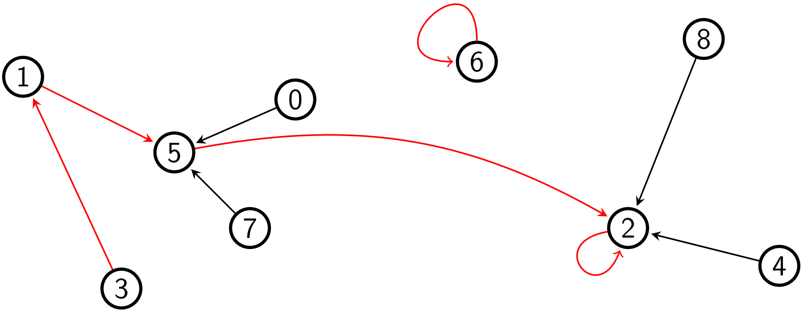
union(3, 8)

Example I



union$(3, 8)$

Example I



$\text{union}(3, 8)$
$\text{equiv}(6, 3)$

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

find-representative(a):

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

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

return $\mathcal{V}[a]$

union(a, b):

$a := \texttt{find-representative}(a)$

$b := \texttt{find-representative}(b)$

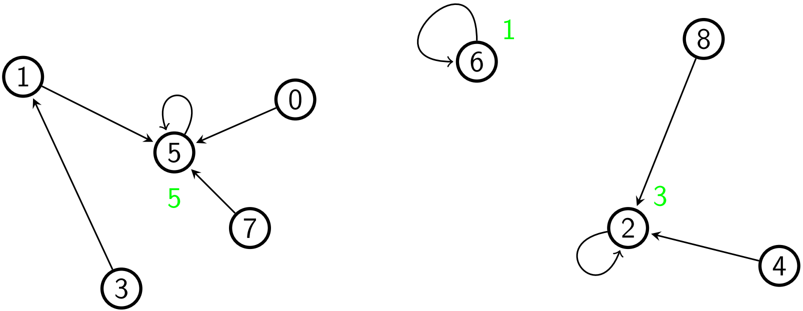
if $\texttt{size}(a) > \texttt{size}(b)$ **then**

$\texttt{exchange}(a, b)$ // merge b into a

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

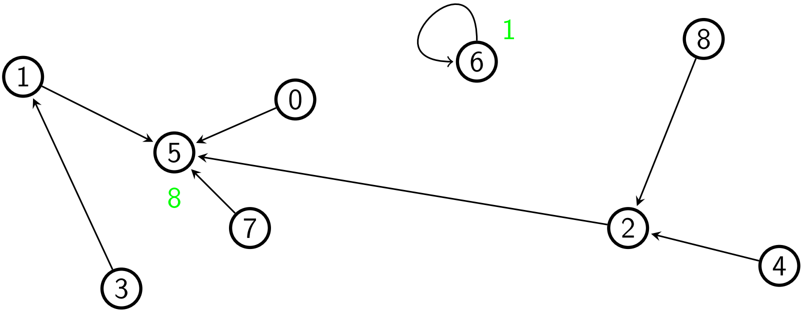
$\texttt{size}(b) = \texttt{size}(a) + \texttt{size}(b)$

Size + Path Compression



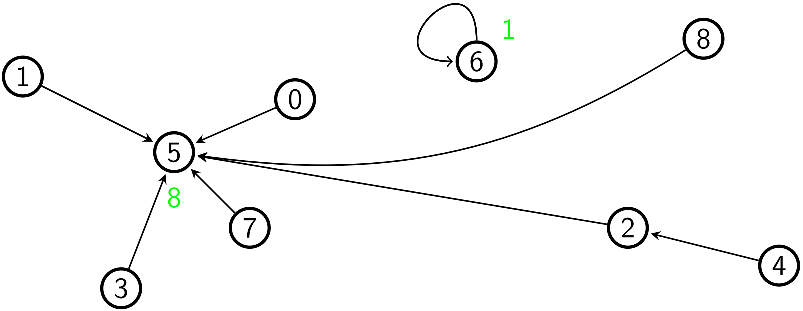
union(7, 8)

Size + Path Compression



union(7, 8)

Size + Path Compression



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