COMP S265F Lab 10: Union-Find Disjoint Sets

Dr. Keith Lee

School of Science and Technology

The Open University of Hong Kong

Overview

- Array implementation
- *Improvement:* Circular linked list implementation
- *Improvement 2:* Union that always updates the smaller set
- Time complexity of all union operations = $O(n \log n)$

Union-Find Disjoint Sets

We need to maintain a collection of disjoint sets from **n** elements.

- Given any two elements x, y, we need to determine

  Find-Set(x) = **Find-Set**(y)

  i.e., to determine whether the set that x belongs is equal to the set that y belongs.
- Given any two sets in the current collection, we need to replace these two sets by its union:

  Union(u, v)

  where u and v are elements in the two sets, respectively.

Note: The actual name of a set is not important. We can give any names to the sets as long as at any time, no two sets have the same name.

Array Implementation

- We declare an array **T** of size **n** for the n elements.
- For any element **x**, **T[x]** stores the name of the set **x** belongs.
- Thus, **Find-Set**(**x**) = **T[x]**.

What is the name of a set?

- Initially, every element x is in a set of itself; we call this set x.

T	a	b	c	d	e	f	g	h
	a	b	c	d	e	f	g	h

Try 1: How to implement Union?

- To execute **Union**(**x**, **y**) we let

  T[**e**] = **x** for all elements **e** in **y**.

- Example: Given

T	a	b	c	d	e	f	g	h
	a	b	c	d	e	f	g	h

- Union(**a**, **h**) ⇒

T	a	b	c	d	e	f	g	a
	a	b	c	d	e	f	g	h

Try 1: How to implement Union? (cont')

- Another example:

T	a	b	c	d	e	f	g	a
	a	b	c	d	e	f	g	h

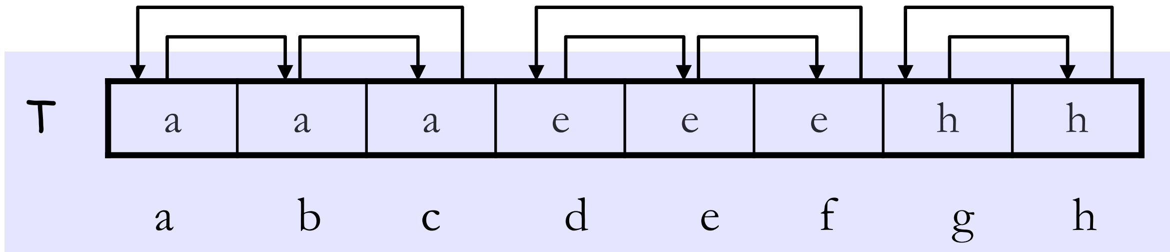
- Union(**b**, **a**) $\Rightarrow$

T	**b**	**b**	c	d	e	f	g	**b**
	a	b	c	d	e	f	g	h

- **Time complexity for Union:** **O(n)** because we have to scan through the whole array to make all the changes.

Improvement: Circular Linked List

- Improvement: Add a circular linked list to join all the elements in a set.

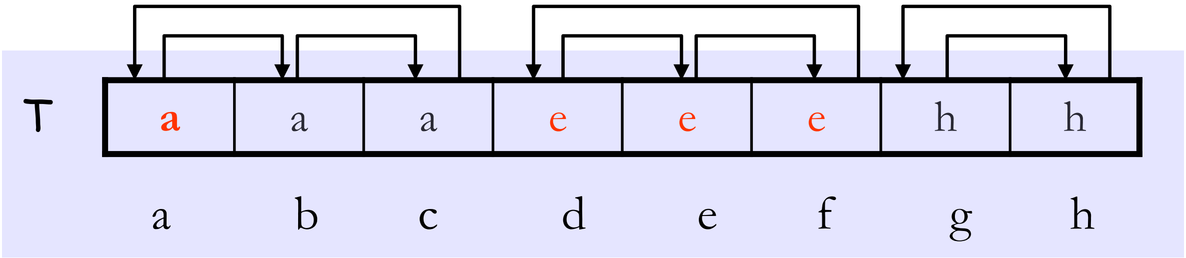


- Or more precisely, add an array ***next*** to remember the next element in the list (or equivalently, in the set).

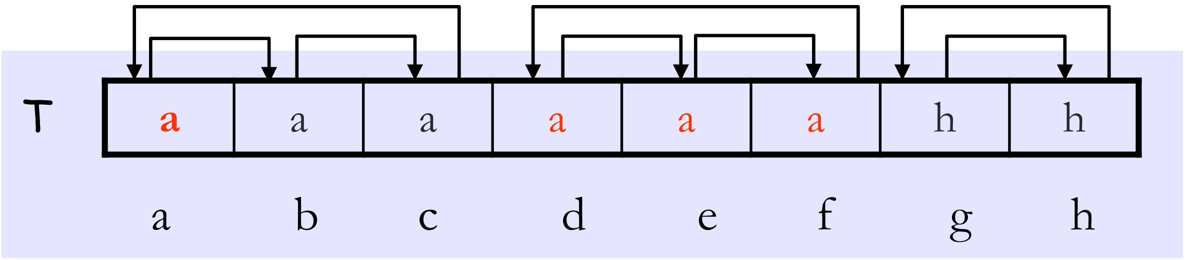
next	b	c	a	e	f	d	h	g
T	a	a	a	e	e	e	h	h
	a	b	c	d	e	f	g	h

Improvement: How to implement Union?

- Union(**a**, **e**)

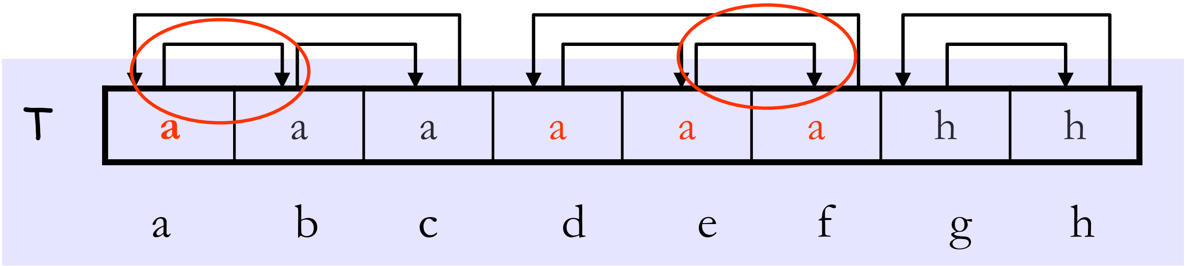


- Update the elements in **e** (by traversing the circular list).

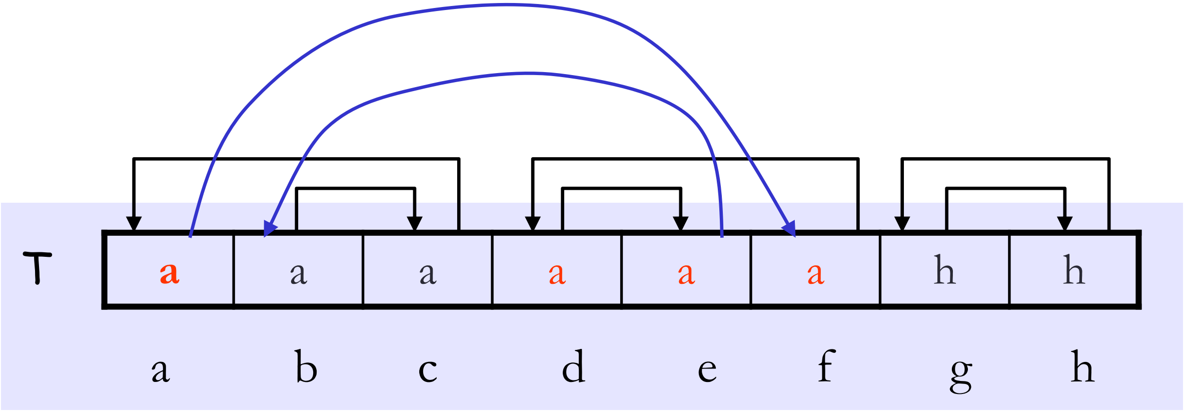


Improvement: How to implement Union? (con't)

- Union(**a**, **e**)



- Update the elements in **e** (by traversing the circular list).



Improvement: Time complexity

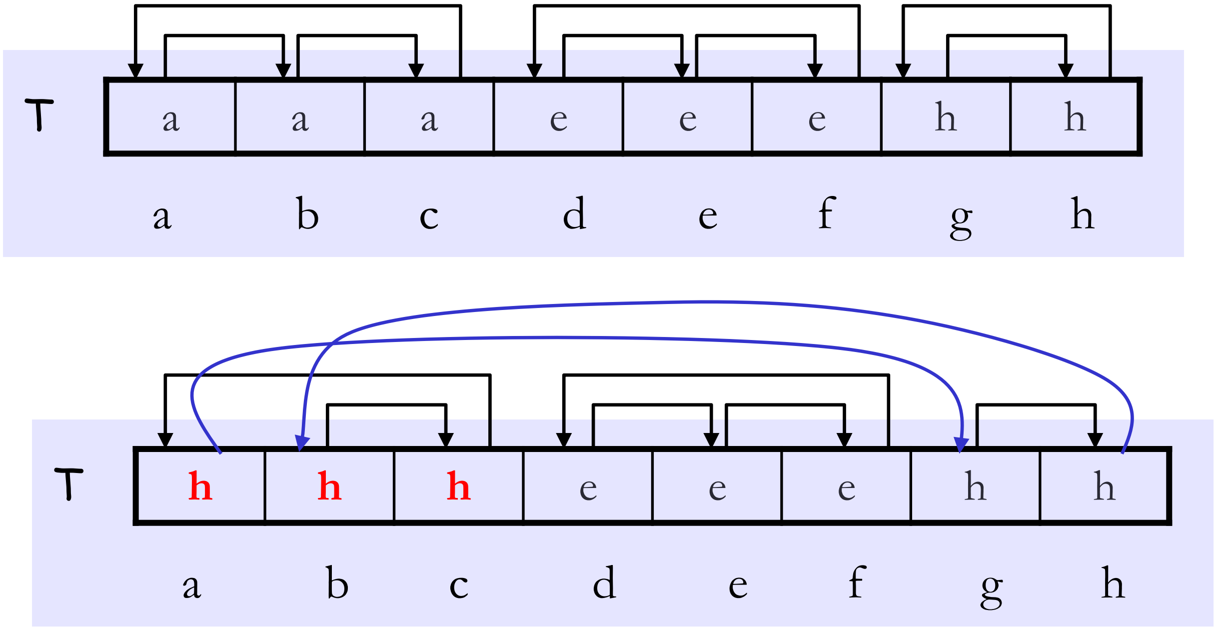
- **Find-Set**(x): O(1) time

- **Union**(x, y):
  - Update of the circular linked list: O(1) time;
  - Update of the set name: |y|.
  - In worst case, it still takes O(n) time.

Total time complexity of all unions = **$O(n^2)$**

Improvement 2: Observation

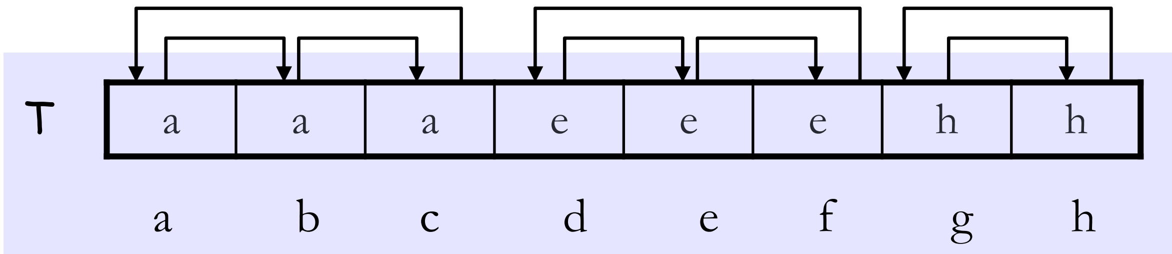
- **Scenario 1:**
  Union(**h**, **a**): elements in **a** changed to **h**.

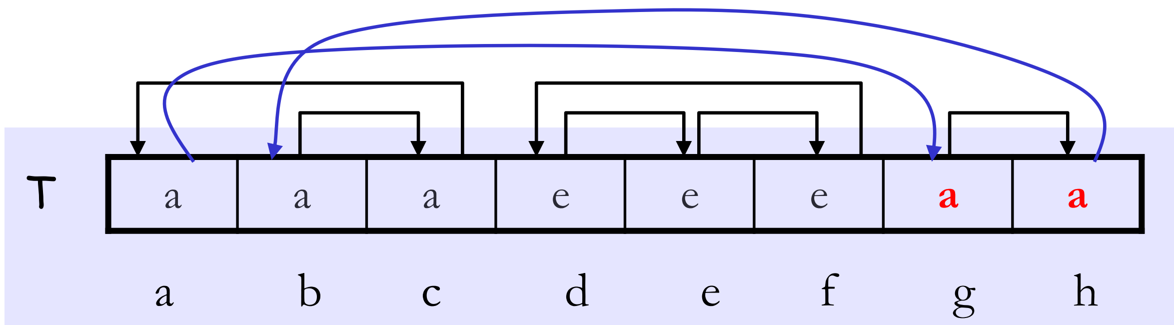


3 set-name updates.

Improvement 2: Observation

- **Scenario 2:**
  Union(**a**, **h**): elements in **h** changed to **a**.

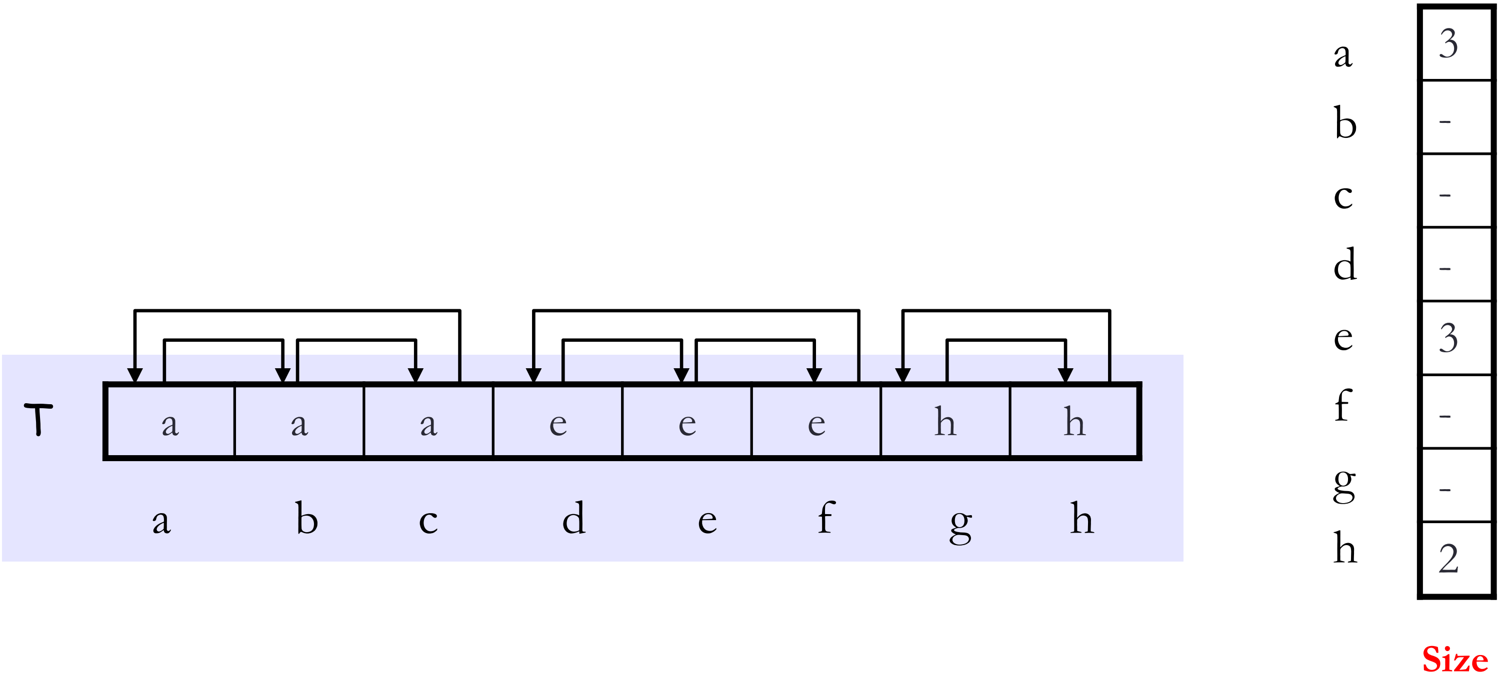




2 set-name updates.

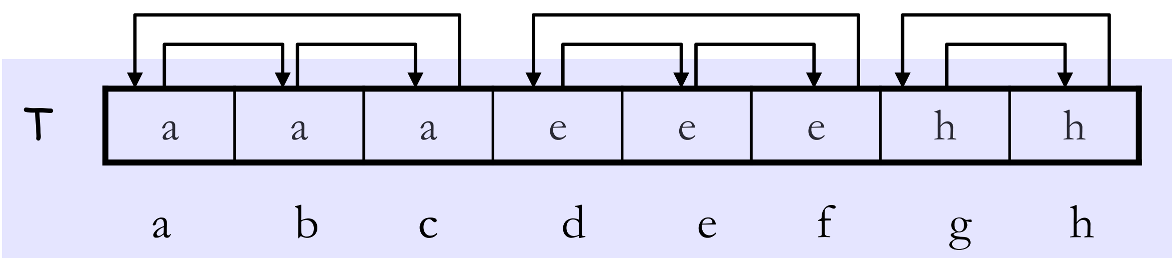
Improvement 2: Always update the smaller set

- **Idea:** Maintain another array that records the size of the sets.



Improvement 2: Always update the smaller set

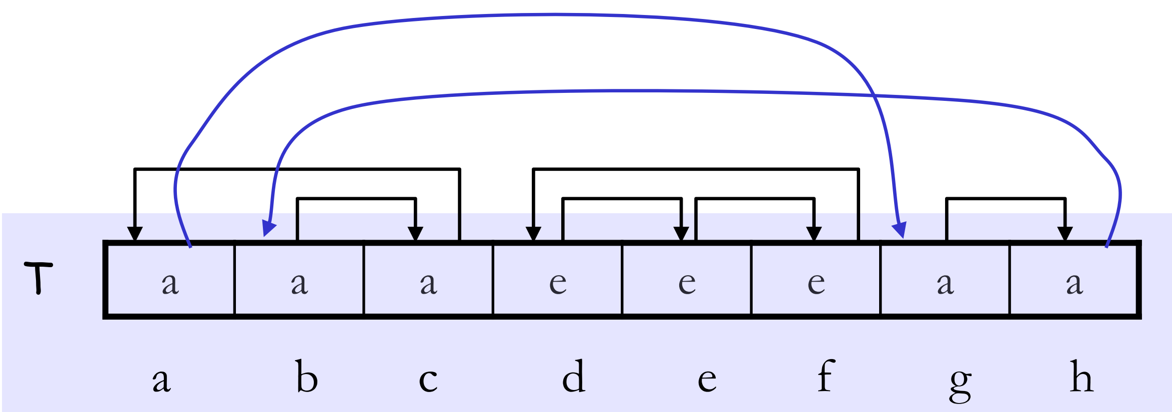
```
Union-by-Size(a,h):
  if size[a] > size[h]:
    Union(a,h)
    size[a]=size[a]+size[h]
  else:
    Union(h,a)
    size[h]=size[a]+size[h]
```



	Size
a	3
b	-
c	-
d	-
e	3
f	-
g	-
h	2

Improvement 2: Always update the smaller set

```
Union-by-Size(a,h):
 if size[a] > size[h]:
    Union(a,h)
    size[a]=size[a]+size[h]
  else:
    Union(h,a)
    size[h]=size[a]+size[h]
```



a	5
b	-
c	-
d	-
e	3
f	-
g	-
h	-

Size

Time complexity for unions: O(n log n)

Theorem. By using **Union-by-Size** to union two ***different*** sets, the time complexity for all unions is **O(n log n).**

Proof. The idea is to consider the number of set-name updates for each element, instead of each Union-by-Size.

- For each element x_0, we know that after an update of **T[x_0]**, the size of the **new set** is **at least double** of that of the old set:

The set in T[x_0]
It's size

x_0	x_1	x_2	x_3	…	x_k
1	≥ 2	≥ 4	≥ 8		$\geq 2^k$

$2^k \leq n$
$\Rightarrow k \leq \log n$

- This follows that we can update **T[x_0]** at most **log n** times because x_k must have size at most n.
- Since **T** has **n** entries, we can update **T** at most **n log n** times.