جامعة التنمية البشرية
UNIVERSITY OF HUMAN DEVELOPMENT

Advanced Data Structures and Algorithms

Associate Professor Dr. Raed Ibraheem Hamed

Computer Science Department
College of Science and Technology
University of Human Development,

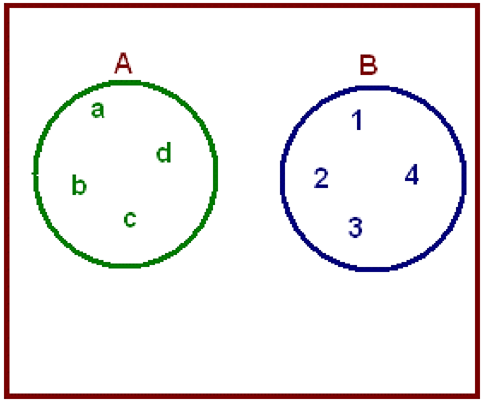
2015 – 2016

What this Lecture is about:

- A disjoint-set data structure
- Disjoint Set Operations
- An Application of Disjoint-Set
- An Application of Disjoint-Set Data Structures
- Linked-List Implementation
- Linked-lists for two sets
- Disjoint-set Implementation: Forests
- Algorithm for Disjoint-Set
- Example

A disjoint-set data structure

- A disjoint-set is a collection **S**=$\{S_1, S_2, \ldots, S_k\}$ of distinct dynamic sets.
- Is a data structure that keeps track of a set of elements partitioned into a number f disjoint subsets.
- Each set is identified by a member of the set, called *representative*.



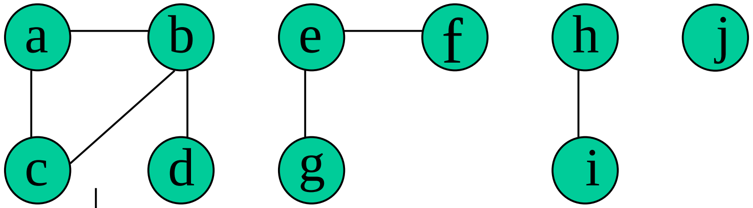
Two disjoint sets A and B

Disjoint set operations:

- MAKE-SET(*x*): create a new set with only *x*. assume *x* is not already in some other set.
- UNION(x,y): combine the two sets containing *x* and *y* into one new set. A new representative is selected.
- FIND-SET(x): returns a pointer to the representative of the unique set containing x. *Find: Determine which subset a particular element is in.*

An Application of Disjoint-Set Data Structures

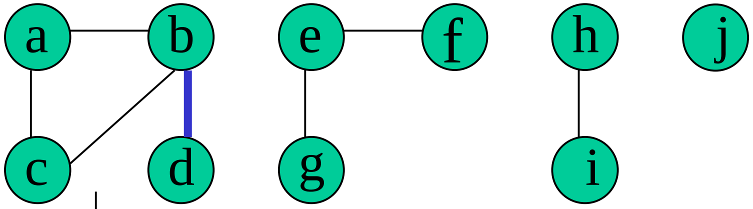
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}

An Application of Disjoint-Set Data Structures

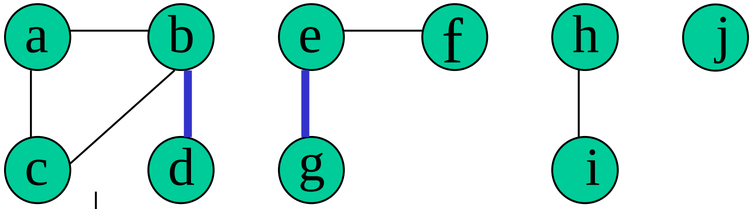
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}

An Application of Disjoint-Set Data Structures

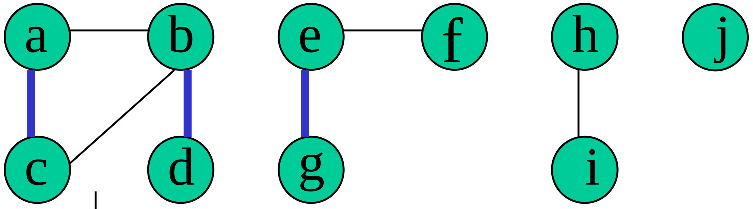
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e, g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}

An Application of Disjoint-Set Data Structures

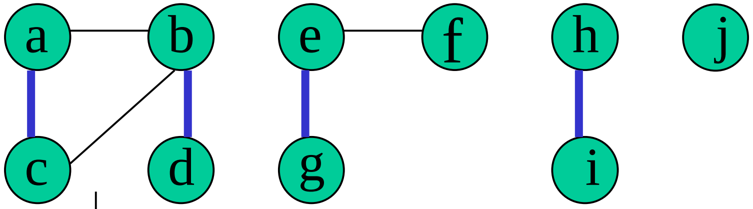
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e, g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}
(a, c)	{a, c}	{b, d}			{e, g}	{f}		{h}	{i}	{j}

An Application of Disjoint-Set Data Structures

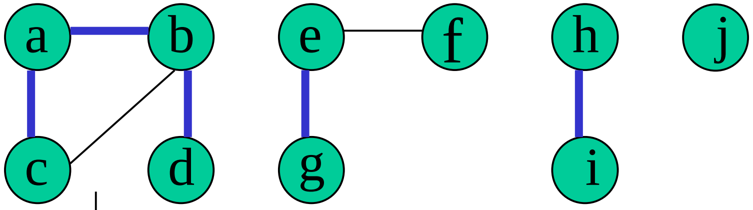
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e, g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}
(a, c)	{a, c}	{b, d}			{e, g}	{f}		{h}	{i}	{j}
(h, i)	{a, c}	{b, d}			{e, g}	{f}		{h, i}		{j}

An Application of Disjoint-Set Data Structures

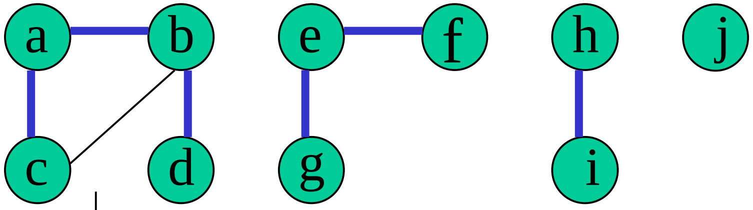
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e, g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}
(a, c)	{a, c}	{b, d}			{e, g}	{f}		{h}	{i}	{j}
(h, i)	{a, c}	{b, d}			{e, g}	{f}		{h, i}		{j}
(a, b)	{a, b, c, d}				{e, g}	{f}		{h, i}		{j}

An Application of Disjoint-Set Data Structures

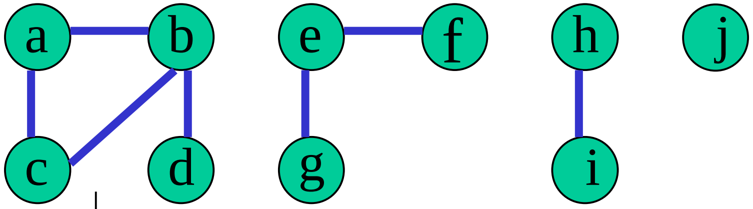
Determining the connected components of an undirected graph G=(V,E)



Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e, g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}
(a, c)	{a, c}	{b, d}			{e, g}	{f}		{h}	{i}	{j}
(h, i)	{a, c}	{b, d}			{e, g}	{f}		{h, i}		{j}
(a, b)	{a, b, c, d}				{e, g}	{f}		{h, i}		{j}
(e, f)	{a, b, c, d}				{e, f, g}			{h, i}		{j}

An Application of Disjoint-Set Data Structures

Determining the connected components of an undirected graph G=(V,E)



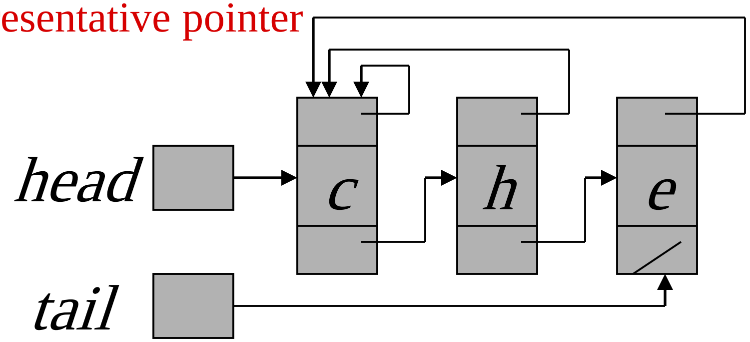
Initial	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b, d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e, g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}
(a, c)	{a, c}	{b, d}			{e, g}	{f}		{h}	{i}	{j}
(h, i)	{a, c}	{b, d}			{e, g}	{f}		{h, i}		{j}
(a, b)	{a, b, c, d}				{e, g}	{f}		{h, i}		{j}
(e, f)	{a, b, c, d}				{e, f, g}			{h, i}		{j}
(b, c)	{a, b, c, d}				{e, f, g}			{h, i}		{j}

Linked-List Implementation

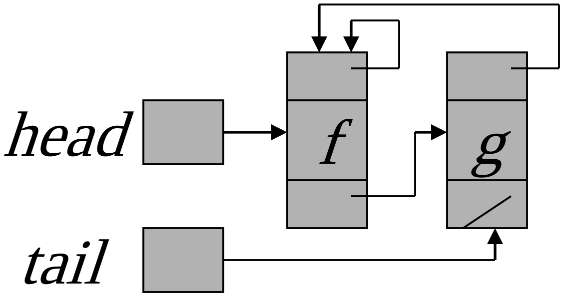
- Each set as a linked-list, with head and tail, and each node contains value, next node pointer and back-to-representative pointer.
- *Example:*
- MAKE-SET costs $O(1)$: just create a single element list.
- FIND-SET costs $O(1)$: just return back-to-representative pointer.

Linked-lists for two sets

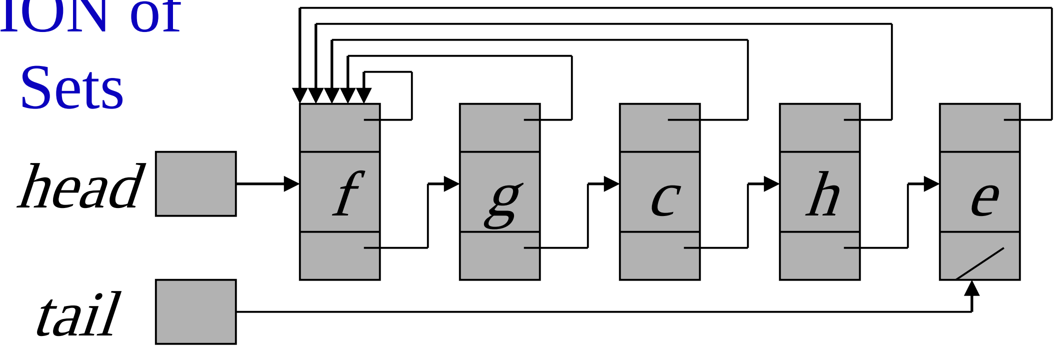
Set {*c*,*h*,*e*}


Representative pointer

Set {*f*, *g*}

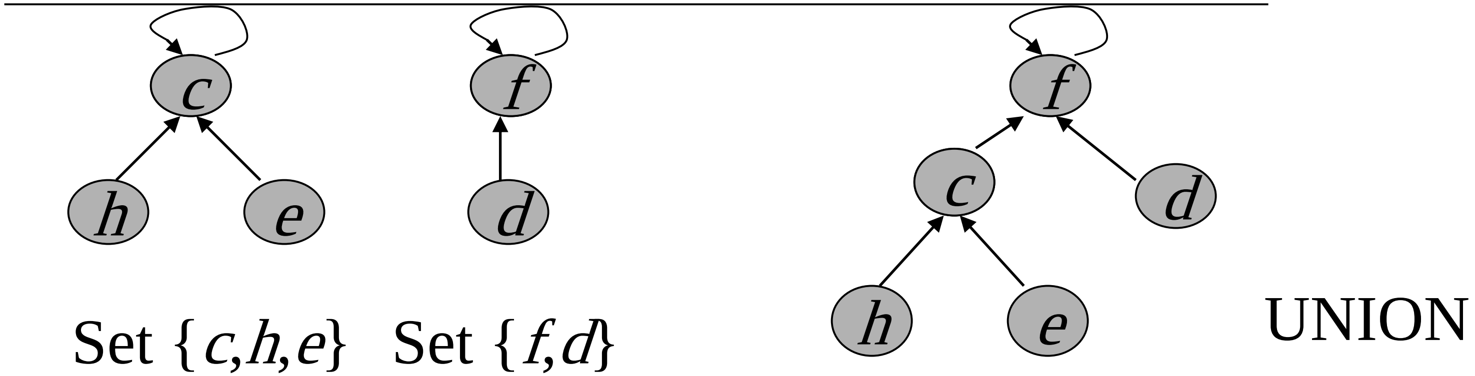


UNION of two Sets



Disjoint-set Implementation: Forests

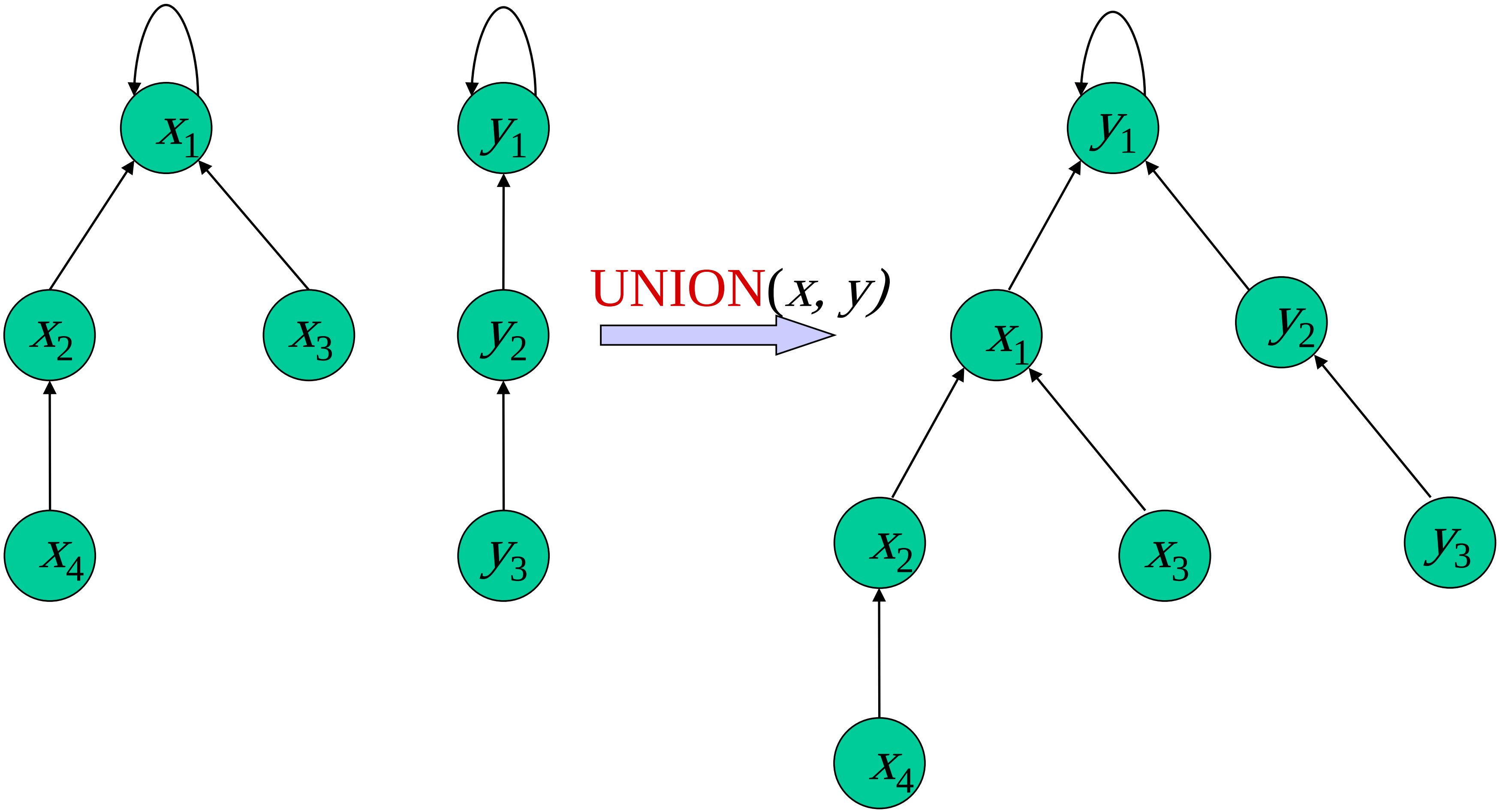
- Rooted trees, each tree is a set, root is the **representative**. Each node points to its parent. Root points to itself.



Set $\{c,h,e\}$ Set $\{f,d\}$

UNION

Disjoint Set Forests

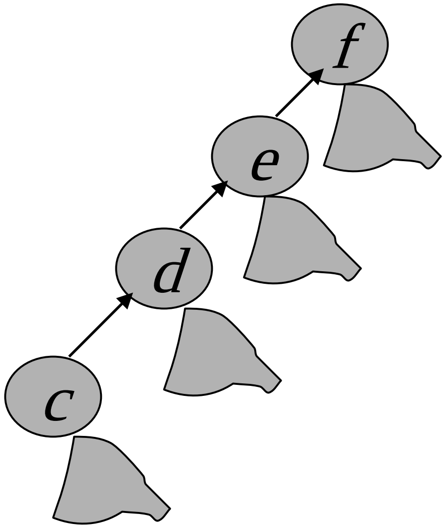


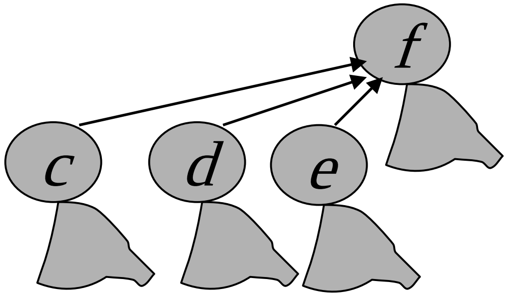
Disjoint Set Forests

Path Compression

Path Compression: used in FIND-SET(*x*) operation, make each node in the path from *x* to the root directly point to the root. Thus reduce the tree height.

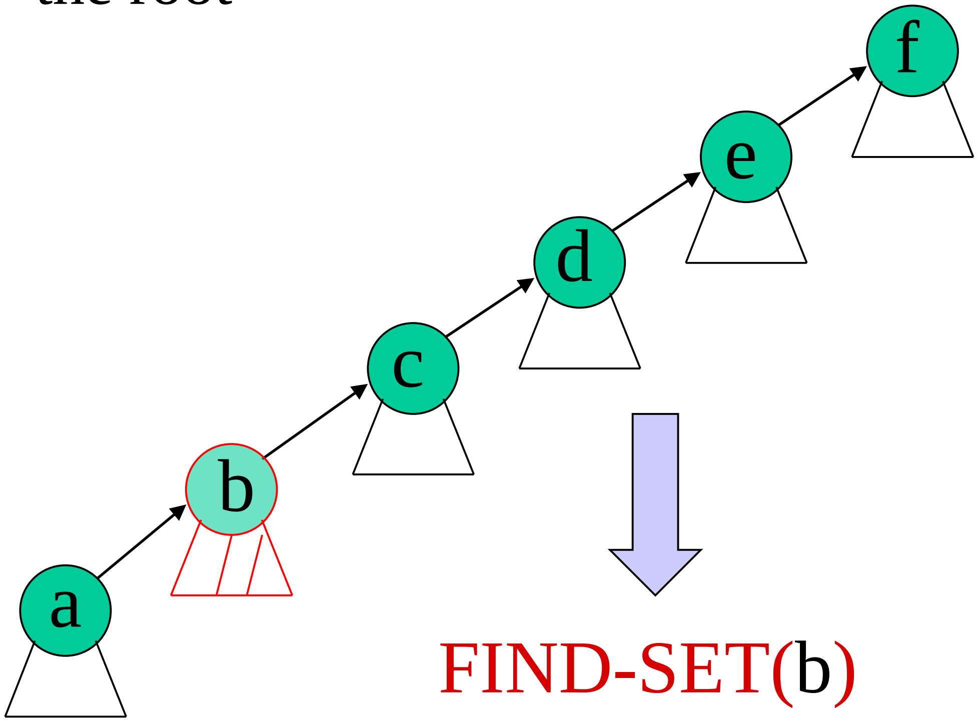
Path Compression





Path Compression

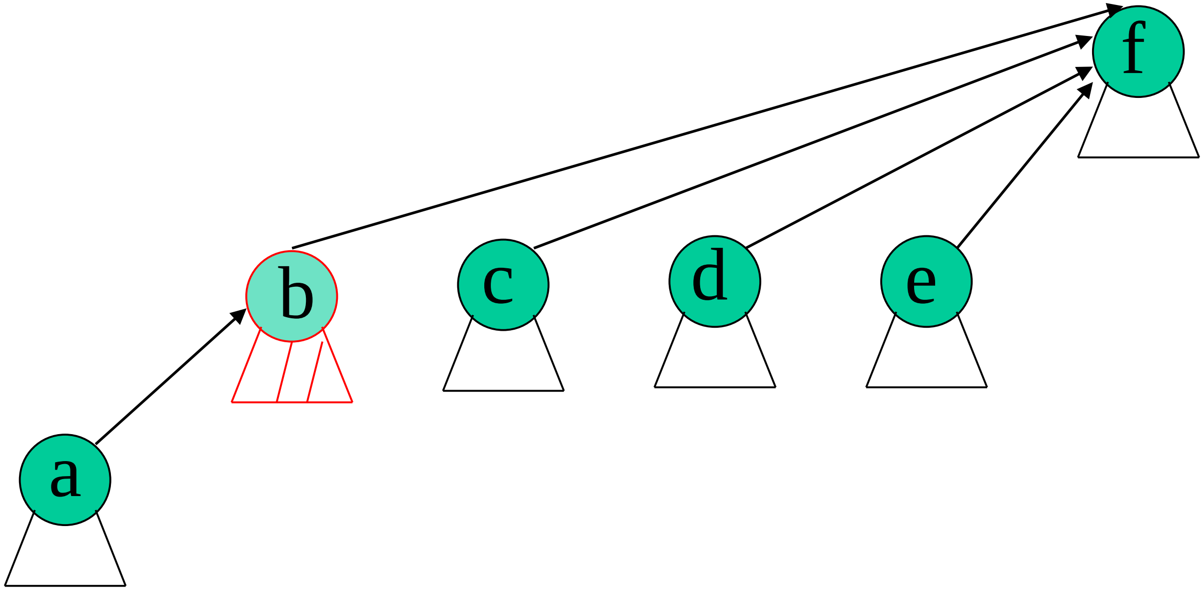
- Use it during the FIND-SET operations
- Make each node on the FIND-PATH to point directly to the root



FIND-SET(b)

Path Compression

Path Compression During FIND-SET(b) Operation



Algorithm for Disjoint-Set

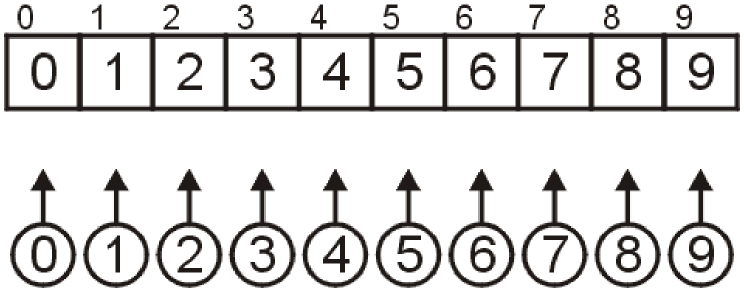
```
function MakeSet(x)
    x.parent := x
```

```
function Find(x)
    if x.parent == x
        return x
    Else
        return Find(x.parent)
```

```
function Union(x, y)
    xRoot := Find(x)
    yRoot := Find(y)
    xRoot.parent := yRoot
```

Example

Consider the following disjoint set on the ten decimal digits:



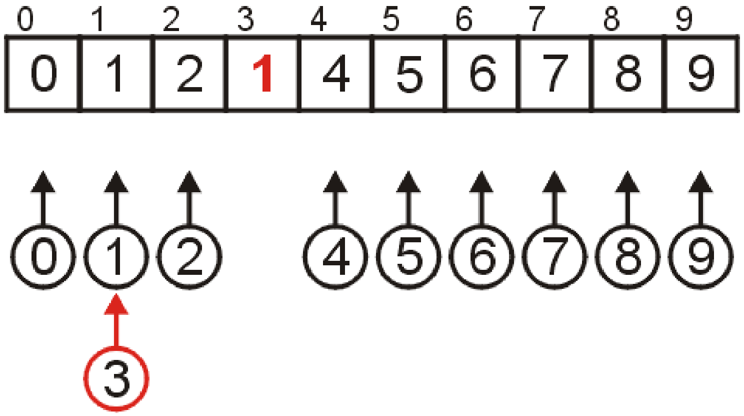
{0}, {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}

Example

If we take the union of the sets containing 1 and 3

```
set_union(1, 3);
```

we perform a find on both entries and update the second

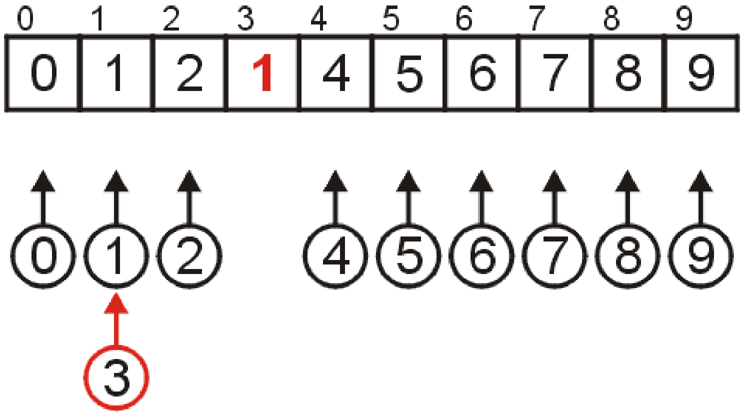


0	1	2	3	4	5	6	7	8	9
0	1	2	1	4	5	6	7	8	9

{0}, {1, 3}, {2}, {4}, {5}, {6}, {7}, {8}, {9}

Example

Now, **`find(1)`** and **`find(3)`** will both return the integer 1



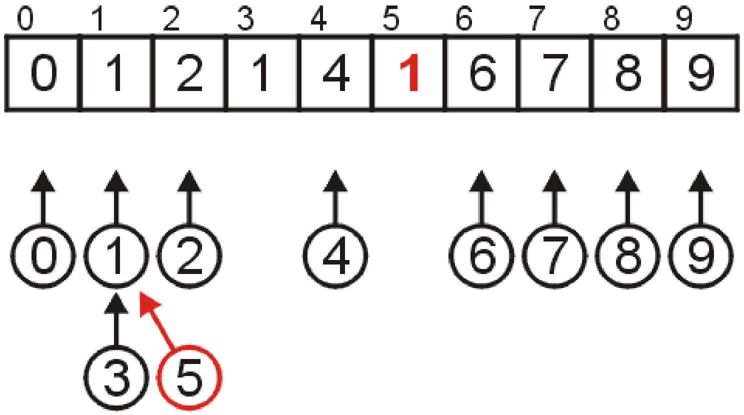
{0}, {1, 3}, {2}, {4}, {5}, {6}, {7}, {8}, {9}

Example

Next, take the union of the sets containing 3 and 5,

`set_union(3, 5);`

we perform a find on both entries and update the second



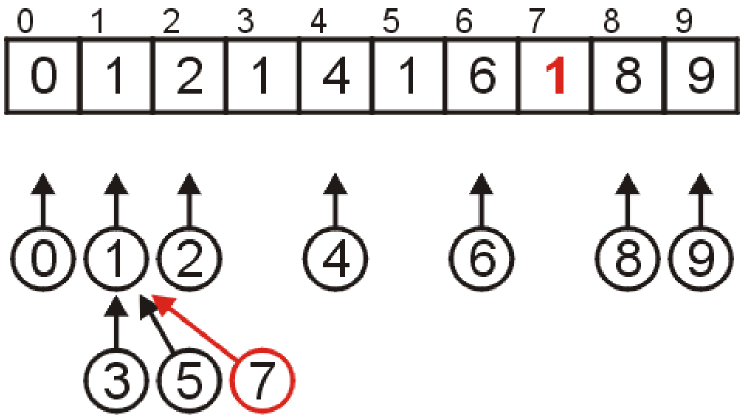
{0}, {1, 3, 5}, {2}, {4}, {6}, {7}, {8}, {9}

Example

Now, if we take the union of the sets containing 5 and 7

```
set_union(5, 7);
```

we update the value stored in `find(7)` with the value `find(5)`:



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

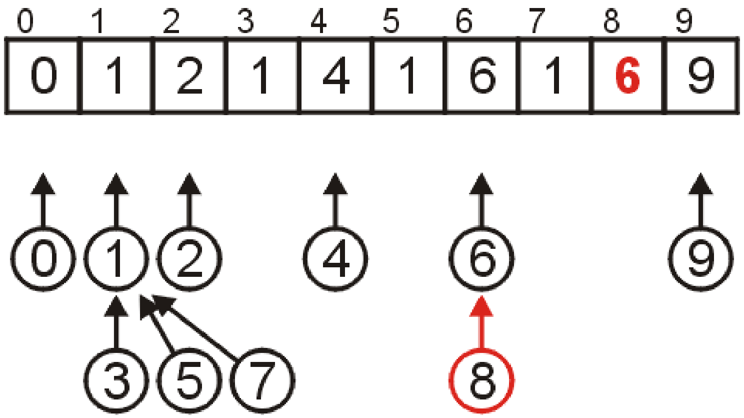
$\{0\}, \{1, 3, 5, 7\}, \{2\}, \{4\}, \{6\}, \{8\}, \{9\}$

Example

Taking the union of the sets containing 6 and 8

`set_union(6, 8);`

we update the value stored in `find(8)` with the value `find(6)`:



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

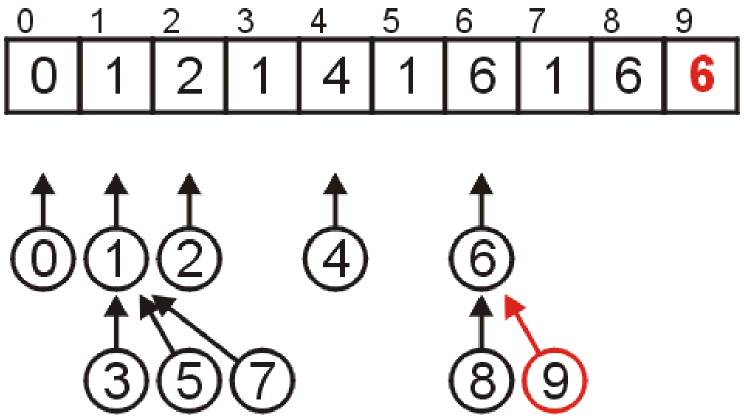
{0}, {1, 3, 5, 7}, {2}, {4}, {6, 8}, {9}

Example

Taking the union of the sets containing 8 and 9

`set_union(8, 9);`

we update the value stored in `find(8)` with the value `find(9)`:



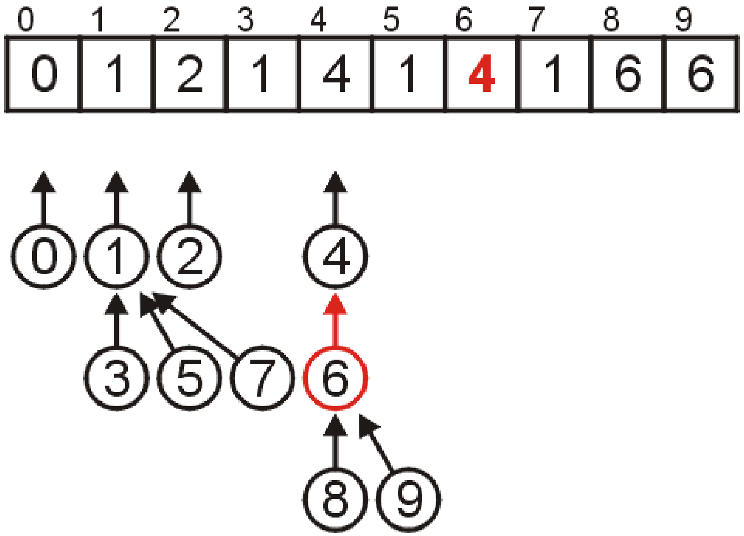
{0}, {1, 3, 5, 7}, {2}, {4}, {6, 8, 9}

Example

Taking the union of the sets containing 4 and 8

`set_union(4, 8);`

we update the value stored in `find(8)` with the value `find(4)`:



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

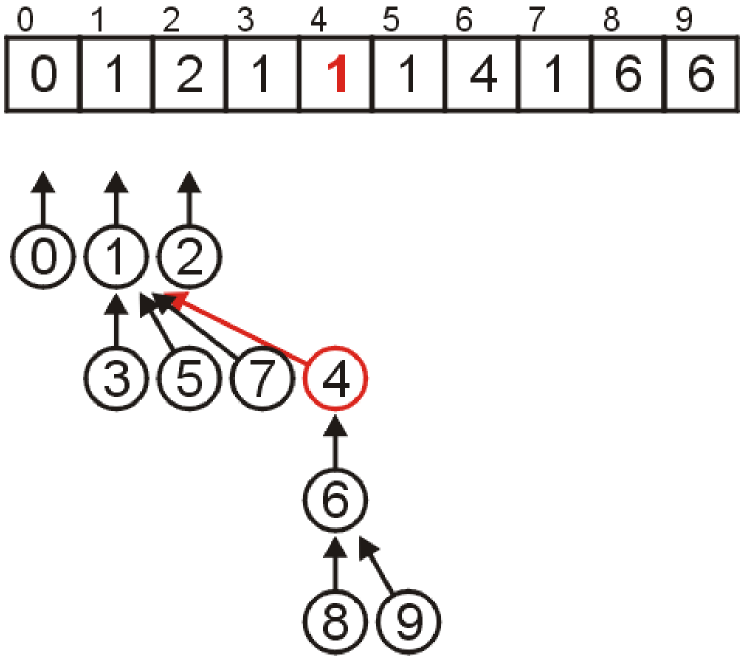
{0}, {1, 3, 5, 7}, {2}, {4, 6, 8, 9}

Example

Finally, if we take the union of the sets containing 5 and 6

```
set_union(5, 6);
```

we update the entry of `find(6)` with the value of `find(5)`:



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

{0}, {1, 3, 4, 5, 6, 7, 8, 9}, {2}

THANK YOU

???