Algorithms
Chapter 21
DS for Disjoint Sets

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Disjoint-set operations** 介紹什麼是 disjoint set
- Linked list representation of disjoint sets 如何用 linked list 來實作
- Disjoint-set forests 用 disjoint forest set 來實作
- Analysis of union by rank with path compression

Overview

S = { {a,b,c}, {e}, {f,g} }

- **Disjoint-set data structures**
  - Also known as “union find.” 又稱為 "union find", 用來維護一個集合的集合
  - Maintain a collection $S = \{S_1, S_2, \ldots, S_k\}$ of **disjoint dynamic** sets.
  - Each set is identified by a **representative**, which is some member of the set. 每一個集合都有一個代表者, 為集合中的一個元素
  - Doesn’t matter which member is the representative, as long as if we ask for the representative twice without modifying the set, we get the same answer both times.

    代表者是誰不重要, 重要的是在集合沒有變動情形下,
    代表者也不能變

Operations

- **Disjoint-set data structures** support the following three operations. make-set(x): 產生一個只包含x的集合,並放到S中
  - MAKE-SET(x): create a new set $S_i = \{x\}$, and add S_i to S.
  - UNION(x, y): unite the dynamic sets that contain x and y, say S_x and S_y, into a new set. union(x,y): 將包含x的集合Sx和包含y的集合合併
    - if $x \in S_x$, $y \in S_y$, then $S \leftarrow S - S_x - S_y \cup \{S_x \cup S_y\}$. S = S - Sx - Sy ∪ {Sx∪Sy}
    - The representative of the resulting set is any member of $S_x \cup S_y$.
    - Since we require the sets in the collection to be disjoint, we "destroy" sets S_x and S_y.
  - FIND-SET(x): return the representative of the set containing x.
    find-set(x): 回傳x所在集合的代表者

Analyzing the running times

- Two parameters:
  - n = number of elements = number of MAKE-SET operations.
  - m = total number of MAKE-SET, UNION, and FIND-SET operations.
- Analysis:
  - $m \geq n$.
  - Have at most $n - 1$ UNION operations.
  - Assume that the first n operations are MAKE-SET.

n：元素的個数，也就是 make-set 动作的次数
m：make-set，union，find-set 动作的次数
最多有 n-1 個 union
假設前 n 個动作為 make-set

An application 可用"union find"來檢測图是否連通

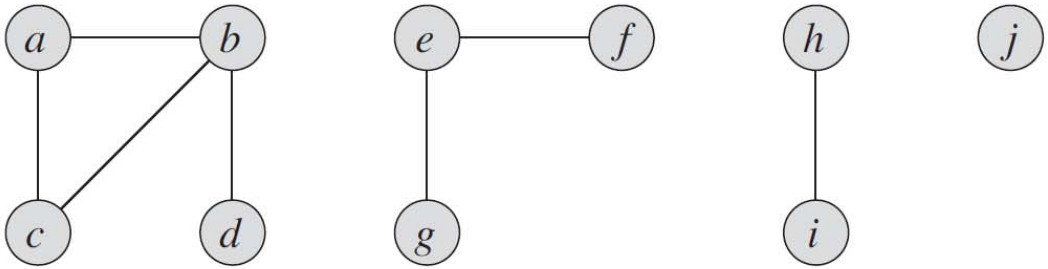
- **Determining the connected components**
  - For a graph $G = (V, E)$, vertices u, v are in same connected component if and only if there's a path between them.
    u和v在同一個連通图上 ⇔ u和v之間有一條path
  - Connected components partition vertices into equivalence classes.

```
CONNECTED-COMPONENTS(G)
1.  for each vertex v∈V[G]
2.      MAKE-SET(v)
3.  for each edge (u, v)∈E[G]
4.      if FIND-SET(u) ≠ FIND-SET(v)
5.          UNION(u, v)
```

```
SAME-COMPONENT(u, v)
1.  if FIND-SET(u) = FIND-SET(v)
2.      return TRUE
3.  else return FALSE
```

可用"union find"來檢測圖是否連通

u和v在同一個連通圖上 ⇔ u和v之間有一條path



Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(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)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

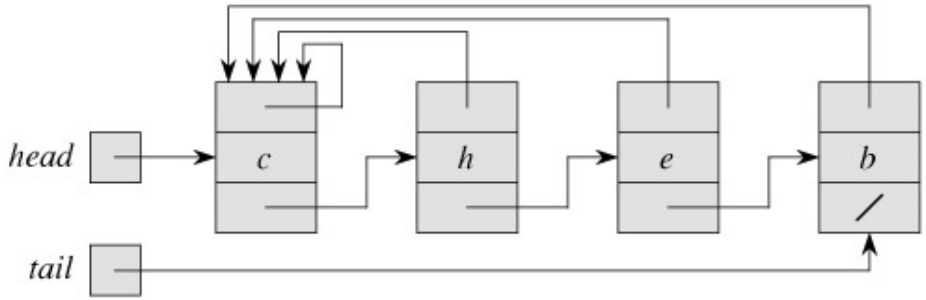
Outline

- Disjoint-set operations
- **Linked list representation of disjoint sets**
- Disjoint-set forests
- Analysis of union by rank with path compression

如何用 linked list 来实作

Linked list representation

- The first object in each linked list serves as its set's representative. list中的第一個元素就是集合中的代表者
- Each object in the linked list contains 注意：每個人都有指標指到代表者
  - a set member,
  - a pointer to the next set member, and
  - a pointer back to the representative.
- Each list maintains pointers **head**, to the representative, and **tail**, to the last object in the list. 有head 和 tail 兩個指標



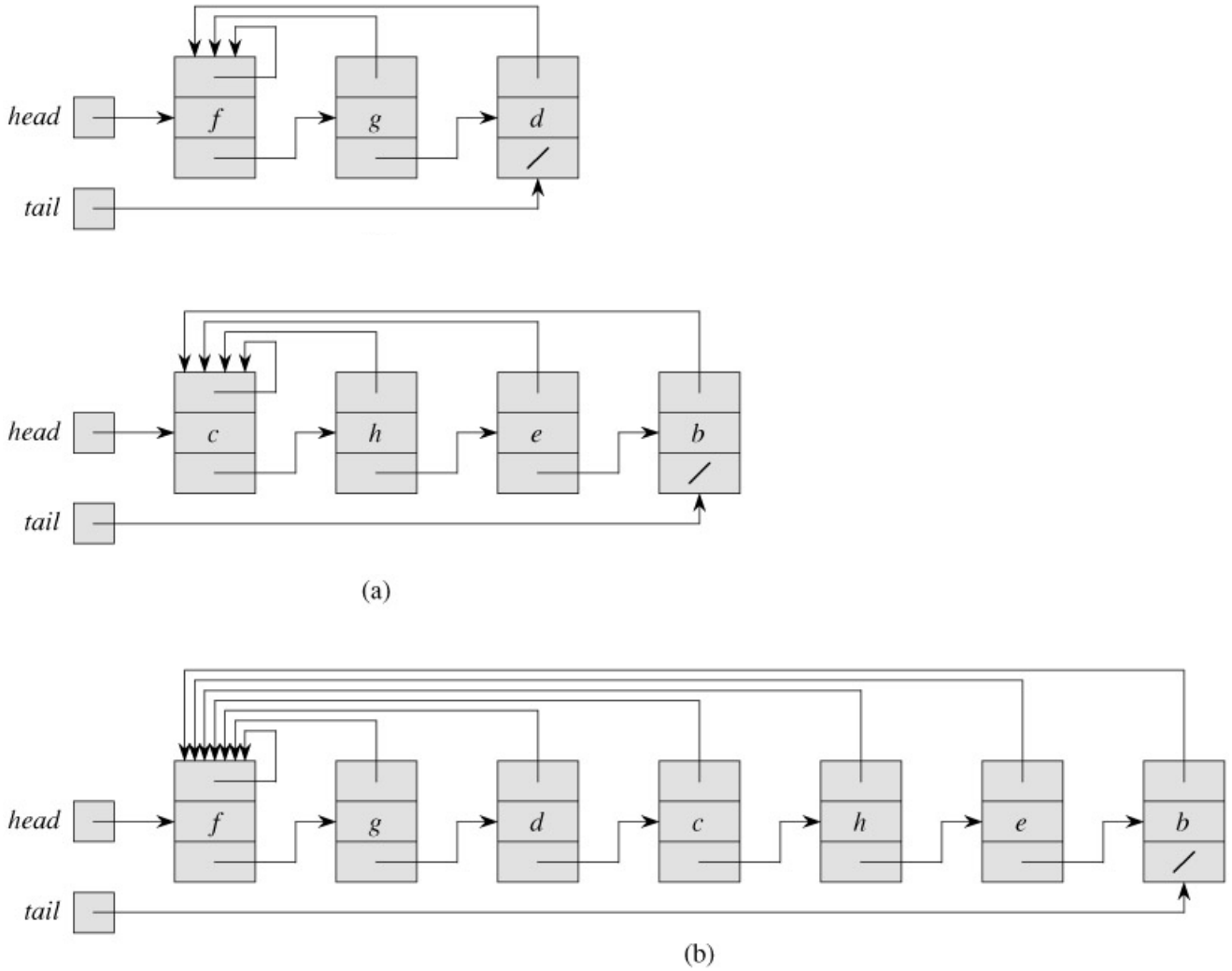
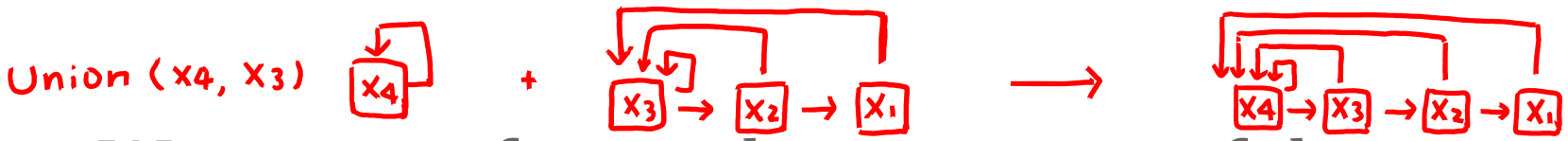


Figure 21.2 (a) Linked-list representations of two sets. (b) The result of UNION(*g*, *e*).

Operations

- MAKE-SET(x): create a new linked list whose only object is x.
  - $O(1)$ time. make-set : O(1)
- FIND-SET(x):return the pointer from x back to the representative.
  - $O(1)$ time. find-set : O(1)
- UNION(x, y): append y's list onto the end of x's list.
  - Use x's tail pointer to find the end.
  - Need to update the representative for each object on y's list.
  - Take time linear in the length of y's list.

Union (x, y) : 因需要更新 y-list 中每一個元素的代表者，所以時間為 O(y-list 個數)



Worst case for implementation of the union

- Suppose that we have objects $x_1, x_2, \ldots, x_n$ and execute the following sequence of operations.

Operation	Number of objects updated	
MAKE-SET(x_1)	1	
MAKE-SET(x_2)	1	
⋮	⋮	
MAKE-SET(x_n)	1	
UNION(x_2, x_1)	1	O(1)
UNION(x_3, x_2)	2	O(2)
UNION(x_4, x_3)	3	O(3)
⋮	⋮	
UNION(x_n, x_{n-1})	$n-1$	O(n-1)

- The running time for the $2n - 1$ operations is $\Theta(n^2)$.
- The amortized time of an operation is $\Theta(n)$.

改善方法：將個数少的放到個数多的 list 上

A weighted-union heuristic$_{1/2}$

- Append the smaller list onto the longer.
- With this simple **weighted-union heuristic**, a single union can still take $\Omega(n)$ time, e.g., if both sets have $n/2$ members.
- **Theorem 21.1** Using the weighted-union heuristic, a sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, takes $O(m+n\lg n)$ time.

單一 union 仍会花 $\Omega(n)$ 的時間

時間：$O(m+n\lg n)$

Proof:

- Each MAKE-SET and FIND-SET still takes $O(1)$, and there are $O(m)$ of them.

make-set 和 find-set 还是 $O(1)$ ⇒ 最多 $O(m)$

- How many times can each object's representative pointer be updated?

注意：每次更新較少的集合

  - It must be in the smaller set each time.

A weighted-union heuristic$_{2/2}$

更新次數	times updated	size of resulting set
	1	≥ 2
	2	≥ 4
	3	≥ 8
	$\vdots$	$\vdots$
	k	$\geq 2^k$
	$\vdots$	$\vdots$
	$\lg n$	$\geq n$

x的代表者被更新後，x所在集合個數

每一個元素的代表者最多更新 $\lg n$ 次
⇒ Union 最多花 $O(n \lg n)$ 時間

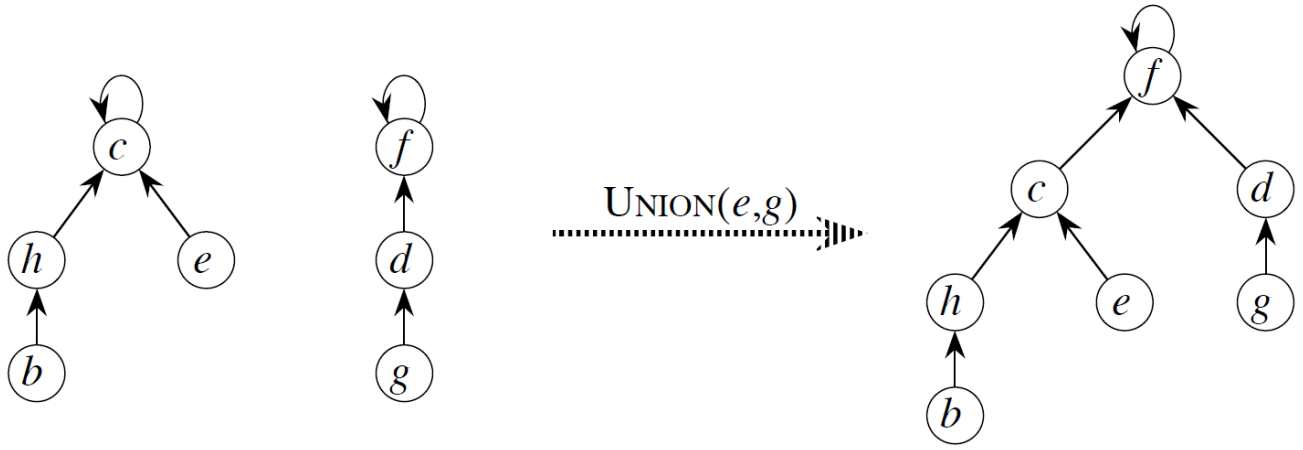
- The first time x's representative pointer was updated, the resulting set must have had at least 2 members.
- Therefore, each representative is updated $\leq \lg n$ times.
- The total time used in updating pointers over all UNION operations is thus $O(n \lg n)$.
- The total time for the entire sequence is thus $O(m + n \lg n)$.

Outline

- Disjoint-set operations
- Linked list representation of disjoint sets
- **Disjoint-set forests** 用disjoint forest set來實作
- Analysis of union by rank with path compression

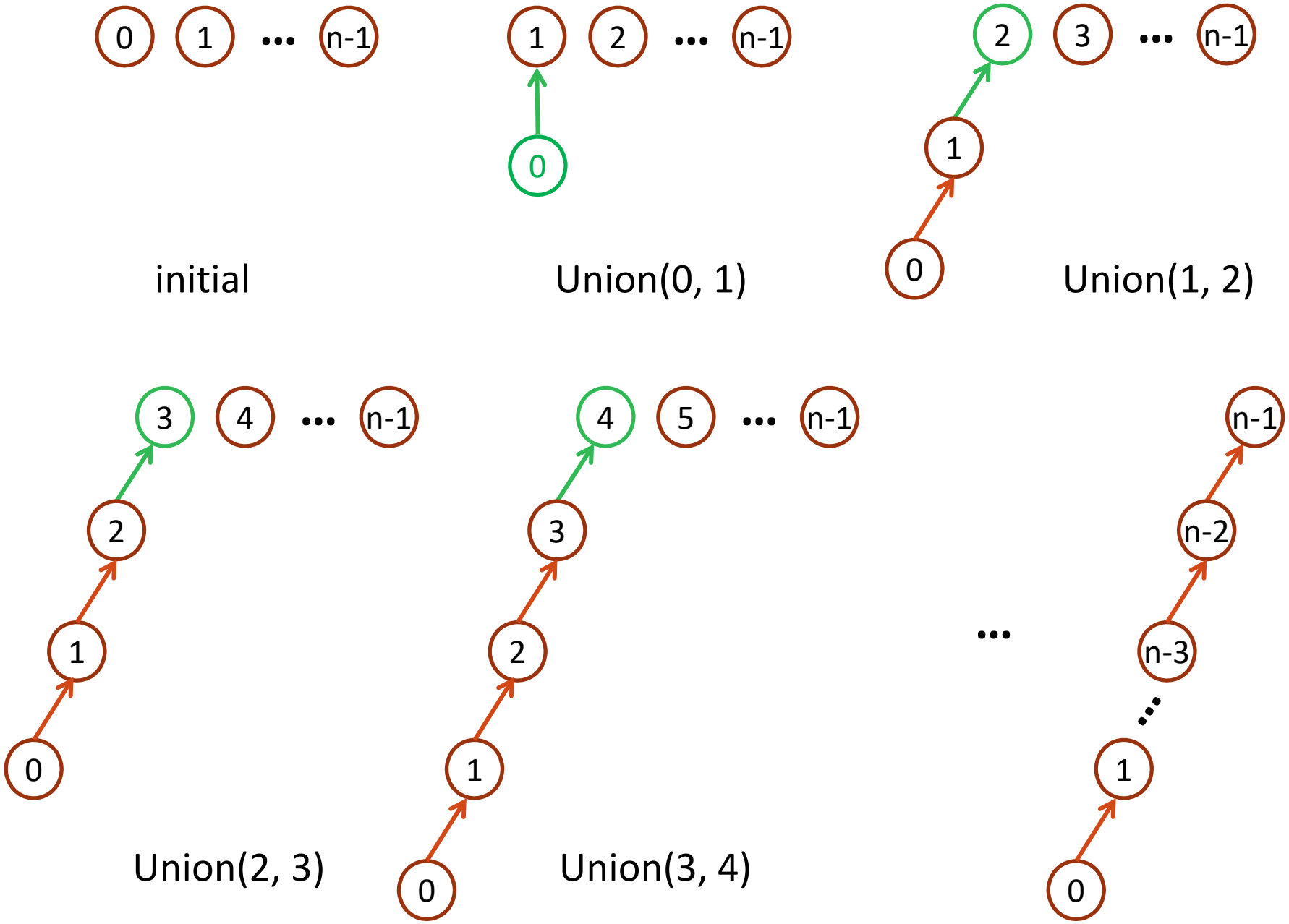
Disjoint-set forest

- In a **disjoint-set forest**:
  - Each tree represents one set; 用tree来表示集合
  - Each member points only to its parent;
  - The root contains the representative; and root就是集合的代表者
  - The root is its own parent. root是自己的parent



Operations

- MAKE-SET(x): creates a tree with just one node.
  - $O(1)$ time. make-set：建一個只有一個元素的樹
- FIND-SET(x): follow parent pointers until we find the root.
  - $O(h)$ time. find-set：找代表者，時間複雜度為O(h)，樹高
  - The nodes visited on this path toward the root constitute the **find path**.
- UNION(x, y): causes the root of one tree to point to the root of the other. Union：將x所在的樹和y所在的樹合併，因要找到代表者，時間為O(h)
  - $O(h)$ time.
- **Problem**: A sequence of $n - 1$ UNION operations may create a tree that is just a linear chain of n nodes. 最差時，會產生一個chain


0
1
...
n-1
initial
1
2
...
n-1
0
Union(0, 1)
2
3
...
n-1
1
0
Union(1, 2)
3
4
...
n-1
2
1
0
Union(2, 3)
4
5
...
n-1
3
2
1
0
Union(3, 4)
...
n-1
n-2
n-3
1
0

Heuristics to improve the running time

- By using two heuristics, however, we can achieve a running time that is almost linear in the total number of operations *m*.

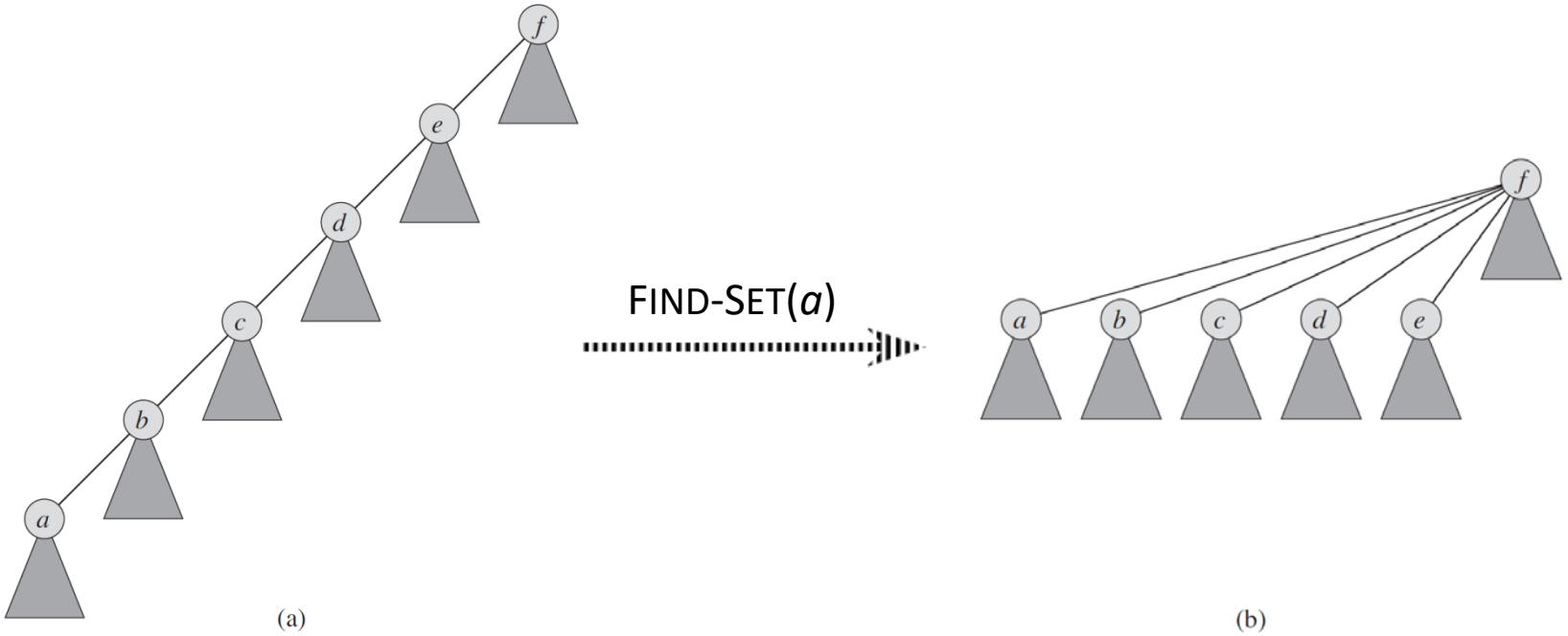
使用以下兩個技巧改善：

- **Union by rank**: make the root of the tree with fewer nodes a child of the root of tree with more nodes.
  - Don't actually use **size**. 將rank小的樹變成大的樹的root的兒子
  - Use **rank**, which is an upper bound on height of node.
  - Make the root with the smaller rank into a child of the root with the larger rank.

rank：樹高的上界　　rank ↓ 樹高

- **Path compression**: make all nodes on the find path direct children of root. 將find path上的点都变成root的兒子

Path compression 將 find path 上的点都变成 root 的兒子



- Triangles represent subtrees whose roots are the nodes shown.
- In Figure b, each node on the find path now points directly to the root after executing FIND-SET(*a*).

Pseudocode for disjoint-set forest

```
MAKE-SET(x)
1.  p[x] ← x
2.  rank[x] ← 0

UNION(x, y)
1.  LINK(FIND-SET(x), FIND-SET(y))

FIND-SET(x)
1.  if x ≠ p[x]
2.      p[x] ← FIND-SET(p[x])
3.  return p[x]
```

```
LINK(x, y)
1.  if rank[x] > rank[y]
2.      p[y] ← x
3.  else p[x] ← y
4.      if rank[x] = rank[y]
5.          rank[y] ← rank[y] + 1
```

比較 rank，大的当爸爸

相等時，rank 加 1

Find-set 分 2 個 pass：

pass 1：先找到 root

pass 2：將 path 上的 node 都变成 root 的兒子

- The FIND-SET procedure is a **two-pass method**:
  - it makes one pass up the find path to find the root; and
  - a second pass back down the find path to update each node to point directly to root.

Effect of the heuristics on the running time 1/2

- **Union by rank** yields a running time of $O(m \lg n)$.

只用union by rank，時間：$O(m \lg n)$

- **Path-compression** gives a worst-case running time

$$\Theta(n + f \cdot (1 + \log_{2+f/n} n)).$$

  - n = number of MAKE-SET operations.
  - f = number of FIND-SET operations.

只用 path-compression，時間：$\Theta(n + f \cdot (1 + \log_{2+f/n} n))$

Effect of the heuristics on the running time 2/2

- When we use both union by rank and path compression, the worst-case running time is $O(m\alpha(n))$, where $\alpha(n)$ is a very slowly growing function. 同時使用 union by rank 和 path-compression
  時間：$O(m\alpha(n))$
- In any conceivable application, $\alpha(n) \le 4$. 在合理的情形下，$\alpha(n) \le 4$

$$\alpha(n) = \begin{cases} 0 & \text{for } 0 \le n \le 2, \\ 1 & \text{for } n = 3, \\ 2 & \text{for } 4 \le n \le 7, \\ 3 & \text{for } 8 \le n \le 2047, \\ 4 & \text{for } 2048 \le n \le A_{4(1)}. \end{cases}$$

$A_{4(1)} >> 10^{80}$

">>" = "much-greater-than"

10^{80}：比地球的砂子多