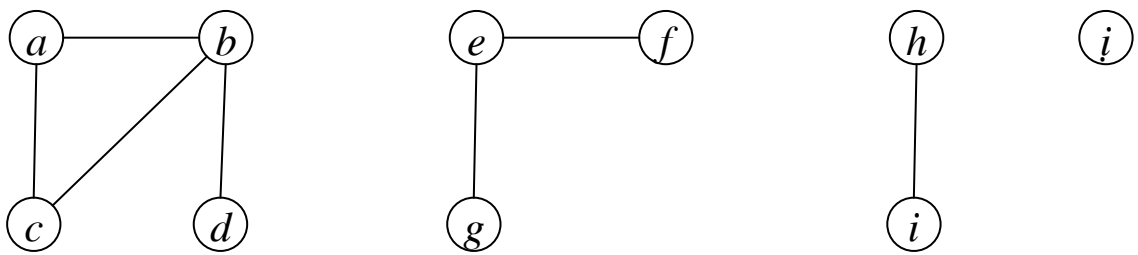
Lecture 10: Các tập rời nhau

- **Bài toán ứng dụng:** Các tạp rời nhau được áp dụng trong các bài toán trên đồ thị
  - Xác định các thành phần liên thông của một đồ thị vô hướng
  - Tìm cây khung tối thiểu
  - Tìm đường đi ngắn nhất
  - …



Các thao tác lên cấu trúc dữ liệu các tập rời nhau

- Cấu trúc dữ liệu *các tập rời nhau* được định nghĩa bởi
  - Một tập S của các tập động rời nhau, $S = \{S_1, S_2, ..., S_k\}$
    - Mỗi tập S_i được tượng trưng bởi một phần tử *đại diện* là một phần tử nào đó của nó.
  - Các thao tác
    - MAKE-SET(x): tạo một tập mới chỉ gồm x. Vì các tập là rời nhau nên x không được nằm trong một tập khác.
    - UNION(s_x, s_y): tạo tập hội của các tập động Sx và Sy lần lượt chứa các phần tử của Sx và Sy, với điều kiện là Sx và Sy là rời nhau.
    - FIND-SET(x): trả về một con trỏ chỉ đến phần tử đại diện của tập chứa x.
- Để cho gọn, sẽ dùng "các tập rời nhau" để gọi "cấu trúc dữ liệu các tập rời nhau".

k1

Một ứng dụng của các tập rời nhau

- Xác định các thành phần liên thông của một đồ thị vô hướng
  - Thủ tục CONNECTED-COMPONENTS xác định các thành phần liên thông của một đồ thị vô hướng.

$V[G]$ là tập các đỉnh của đồ thị G, $E[G]$ là tập các cạnh của G.

```
CONNECTED-COMPONENTS(G)
1  for mỗi đỉnh v ∈ V[G]
2       do MAKE-SET(v)
3  for mỗi cạnh (u, v) ∈ E[G]
4       do if FIND-SET(u) ≠ FIND-SET(v)
5             then UNION(u, v)
```

Bài tập: Chứng tỏ rằng sau khi tất cả các cạnh đã được xử lý bởi CONNECTED-COMPONENTS thì hai đỉnh ở trong cùng một thành phần liên thông nếu và chỉ nếu chúng ở trong cùng một tập.

k1

khmt-hung, 1/10/2007

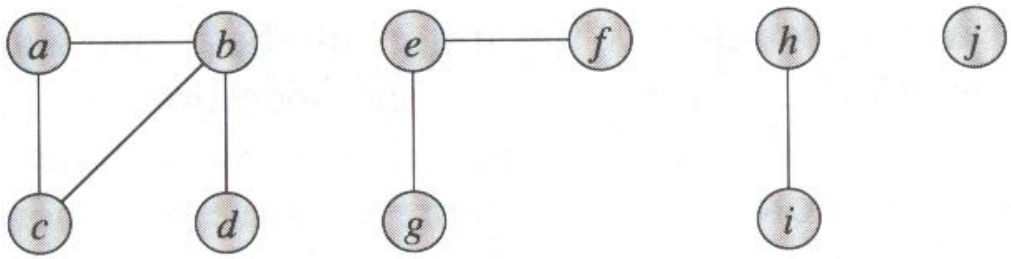
Một ứng dụng của các tập rời nhau (tiếp)

– Thủ tục SAME-COMPONENT xác định hai đỉnh có cùng một thành phần liên thông hay không.

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

Một ứng dụng của các tập rời nhau (tiếp)

- Ví dụ: một đồ thị với 4 thành phần liên thông



(a)

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)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

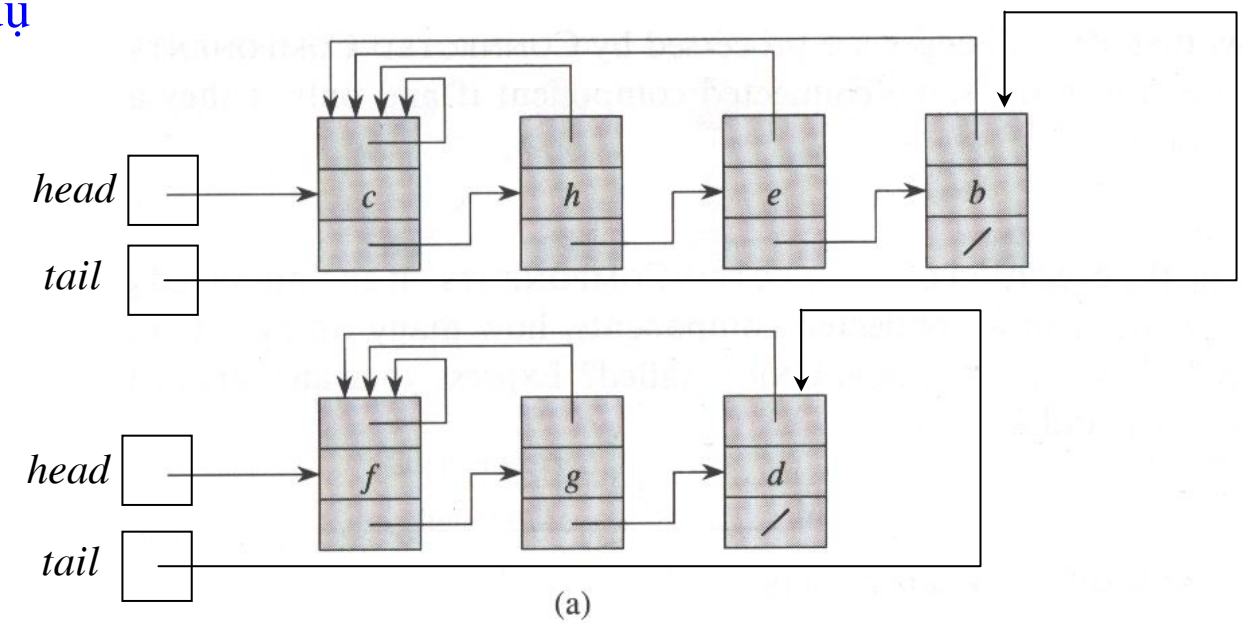
(b)

Biểu diễn các tập rời nhau dùng danh sách liên kết

- Biểu diễn các tập rời nhau dùng danh sách liên kết (linked-list representation of disjoint sets):
  - Biểu diễn mỗi tập bằng một danh sách liên kết. Trong mỗi danh sách liên kết
    - Đối tượng đứng đầu được dùng làm phần tử đại diện của tập.
    - Mỗi đối tượng trong danh sách liên kết chứa
      - phần tử của tập
      - con trỏ chỉ đến đối tượng chứa phần tử kế tiếp
      - con trỏ chỉ đến phần tử đại diện của tập.
    - Quản lý danh sách bằng hai con trỏ *head* chỉ đến đại diện của tập. Con trỏ *tail* chỉ đến phần tử cuối trong danh sách.

Biểu diễn tập bằng danh sách liên kết

- Ví dụ



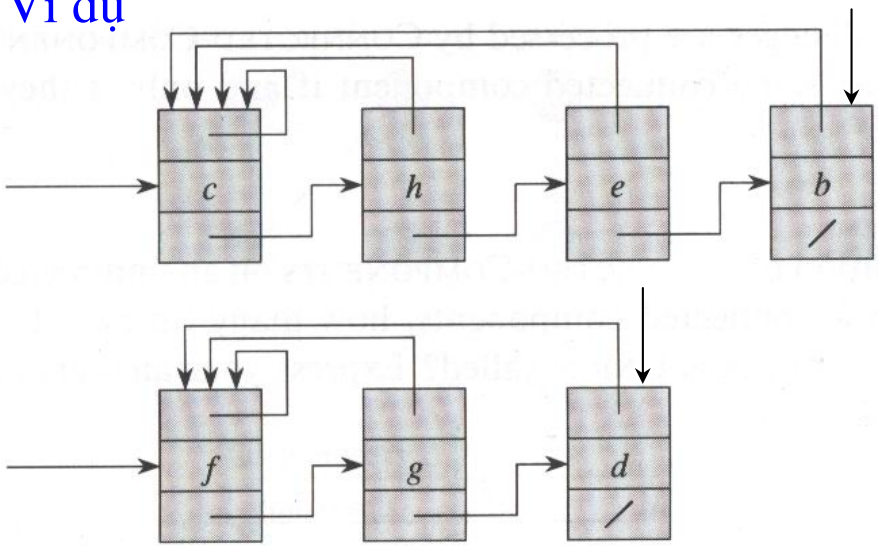
(a)

Biểu diễn tập bằng danh sách liên kết (tiếp)

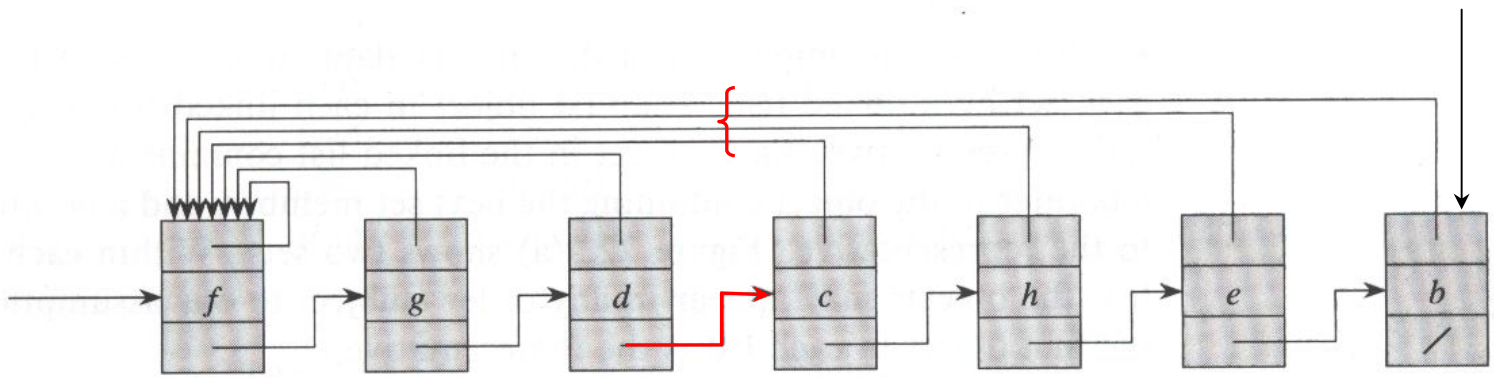
- Các thao tác
  - MAKE-SET(x): tạo một danh sách liên kết chỉ gồm đối tượng x.
  - FIND-SET(x): trả về con trỏ đến đại diện của tập chứa x.
  - UNION(x, y):
    - gắn danh sách của x vào đuôi của danh sách của y
    - cập nhật các con trỏ của các đối tượng trong danh sách cũ của x để chúng chỉ đến đại diện của tập, tức là đầu của danh sách cũ của y.

Biểu diễn tập bằng danh sách liên kết (tiếp)

- Ví dụ



(a)



(b)

Thao tác UNION không dùng heuristic

- Ví dụ một chuỗi gồm $2n - 1$ thao tác lên n đối tượng cần $\Theta(n^2)$ thời gian.

Thao tác	Số các đối tượng được cập nhật	
MAKE-SET(x_1)	1	n
MAKE-SET(x_2)	1	
.		
.		
.		
MAKE-SET(x_n)	1	
UNION(x_1 , x_2)	1	$\Theta(n^2)$
UNION(x_2 , x_3)	2	
UNION(x_3 , x_4)	3	
.		
.		
.		
UNION(x_{n-1} , x_n)	$n - 1$	

Heuristic để tăng tốc của UNION

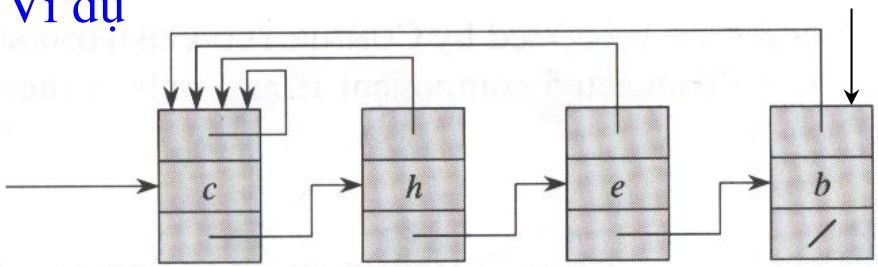
- Nhận xét: Khi hợp hai danh sách trong UNION, mọi con trỏ (chỉ đến đại diện mới) của các phần tử trong danh sách được gắn vào đuôi của danh sách kia phải được cập nhật.

  Giả sử mỗi danh sách có chứa thêm chiều dài của nó.

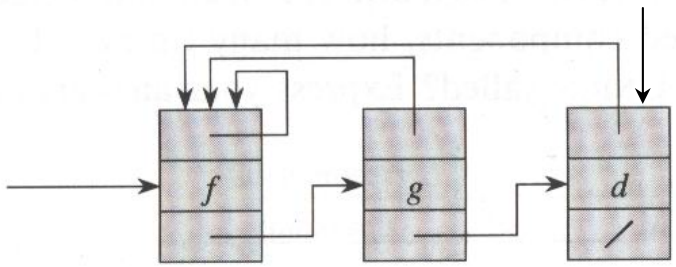
- Heuristic *hợp theo trọng số* (weighted-union heuristic): khi hợp hai danh sách
  - gắn danh sách ngắn hơn vào đuôi của danh sách dài hơn (nếu các danh sách dài như nhau thì có thể gắn tùy ý).

Heuristic hợp theo trọng số

- Ví dụ

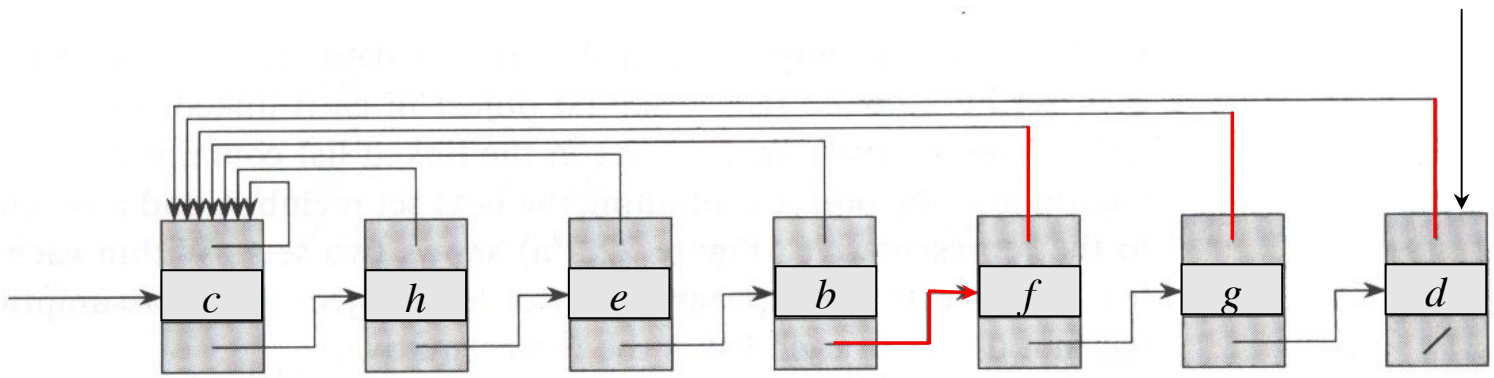


chiều dài = 4



chiều dài = 3

(a)



(b)

Phân tích các thao tác lên các tập rời nhau

- Phân tích thời gian chạy của các thao tác sẽ dựa trên hai tham số sau
  - n, số các thao tác MAKE-SET
  - m, số tổng cộng các thao tác MAKE-SET, UNION, và FIND-SET.
- Nhận xét:
  - Sau $n - 1$ lần gọi UNION lên các tập rời nhau thì còn lại đúng một tập.
  - $m \geq n$.

Biểu diễn tập bằng danh sách liên kết: thời gian chạy

- **Định lý** (Theorem 22.1)

  Bằng cách dùng biểu diễn danh sách liên kết cho các tập rời nhau và heuristic hợp theo trọng số (weighted-union heuristic), một dãy gồm có m thao tác MAKE-SET, UNION, và FIND-SET, trong đó có n thao tác là MAKE-SET, tốn $O(m + n \lg n)$ thời gian.

  Chứng minh

  – Mỗi MAKE-SET chạy trong thời gian $O(1)$

  – Mỗi FIND-SET chạy trong thời gian $O(1)$

  – Xác định thời gian chạy của các thao tác UNION:

  - Thời gian chạy của các thao tác UNION là thời gian tổng cộng cập nhật con trỏ chỉ đến phần tử đại diện của tập chứa phần tử đó.

Biểu diễn tập bằng danh sách liên kết: thời gian chạy

Chứng minh (tiếp theo)

- Xét đối tượng x bất kỳ trong một tập bất kỳ của các tập rời nhau. Mỗi lần con trỏ chỉ đến phần tử đại diện của tập chứa x được cập nhật, thì x phải đã nằm trong tập nhỏ hơn

  $\Rightarrow$

  - Lần 1 cập nhật con trỏ của x: tập kết quả phải có ít nhất 2 phần tử
  - Lần 2 cập nhật con trỏ của x: tập kết quả phải có ít nhất 2^2 phần tử
  - ...
  - Lần k cập nhật con trỏ của x: tập kết quả phải có ít nhất 2^k phần tử.

  Vì tập có nhiều nhất là n phần tử nên $2^k \leq n$. Vậy số lần cập nhật con trỏ của x nhiều nhất là $k \leq \lg n$.

Biểu diễn tập bằng danh sách liên kết: thời gian chạy

Chứng minh (tiếp theo)

- Vì x là phần tử bất kỳ nên thời gian tổng cộng để cập nhật các con trỏ của mọi phần tử là $O(n \lg n)$.

– Thời gian chạy tổng cộng của dãy m thao tác là: $O(m) + O(n \lg n) = O(m + n \lg n)$.

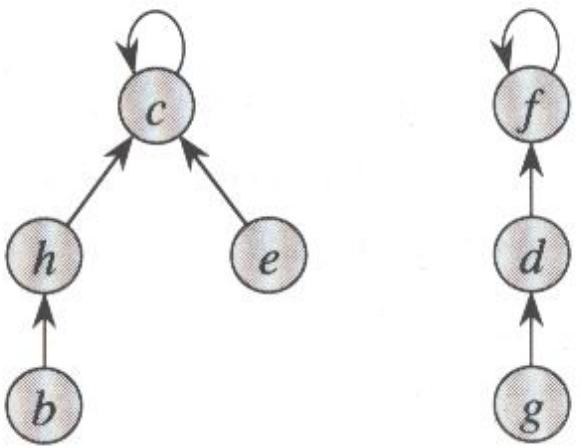
Biểu diễn các tập rời nhau bằng rừng

- Biểu diễn các tập rời nhau bằng rừng (disjoint-set forest)
  - Biểu diễn mỗi tập bằng một cây có gốc:
    - Mỗi nút của cây chứa một phần tử của tập

  Ngoài ra:
    - Mỗi nút chứa một con trỏ chỉ đến cha của nó
    - Gốc của mỗi cây chứa đại diện của tập và là cha của chính nó.

Biểu diễn các tập rời nhau bằng rừng (tiếp)

- Ví dụ
  - Hai cây sau biểu diễn các tập $\{b, c, e, h\}$ và $\{d, f, g\}$.
  - c và f lần lượt là phần tử đại diện của các tập $\{b, c, e, h\}$ và $\{d, f, g\}$.



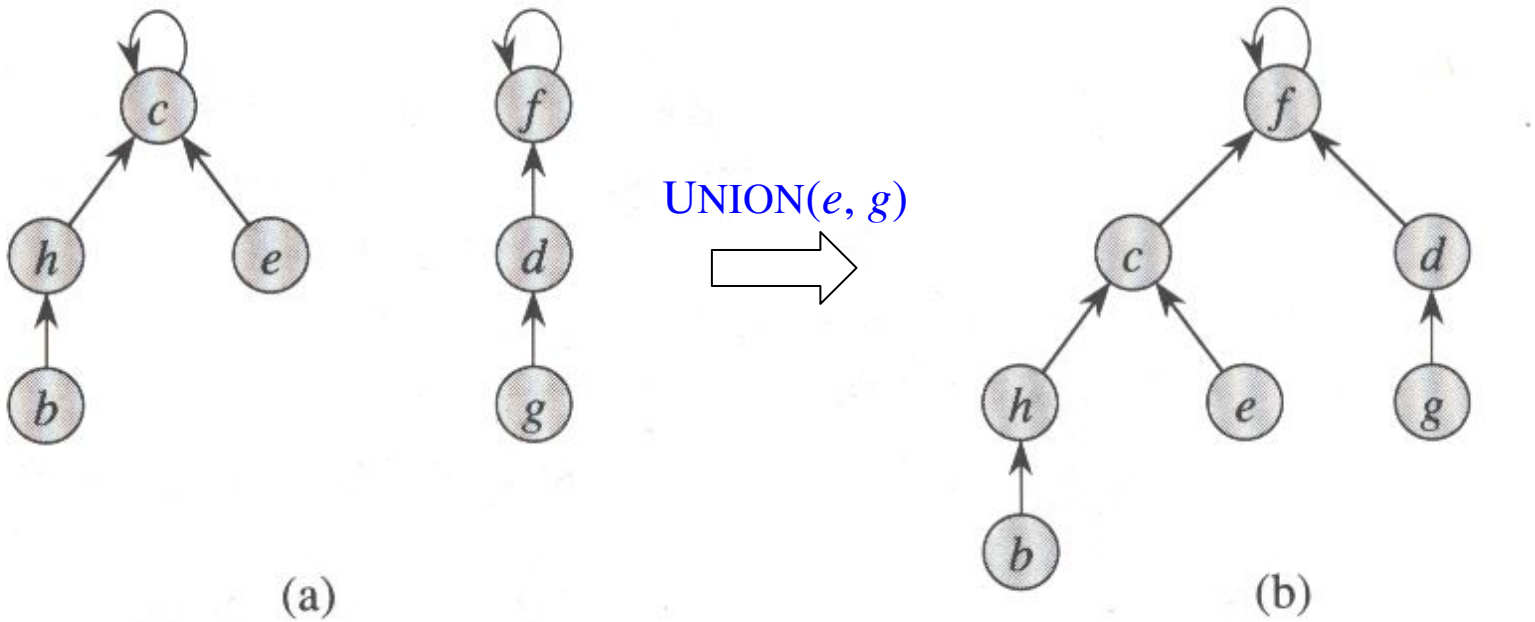
(a)

Biểu diễn các tập rời nhau bằng rừng: các thao tác

- Các thao tác lên các tập rời nhau khi biểu diễn bằng rừng
  - Thao tác MAKE-SET: tạo một cây chỉ có một nút.
  - Thao tác FIND-SET bằng cách đuổi theo các con trỏ chỉ đến nút cha cho đến khi tìm được nút gốc của cây.
    - Các nút được ghé qua khi gọi FIND-SET tạo thành *đường dẫn* (*find path*).
  - Thao tác UNION: làm cho con trỏ của gốc cây này chỉ đến gốc của cây kia.

Biểu diễn các tập rời nhau bằng rừng

- Ví dụ



Biểu diễn tập bằng cây

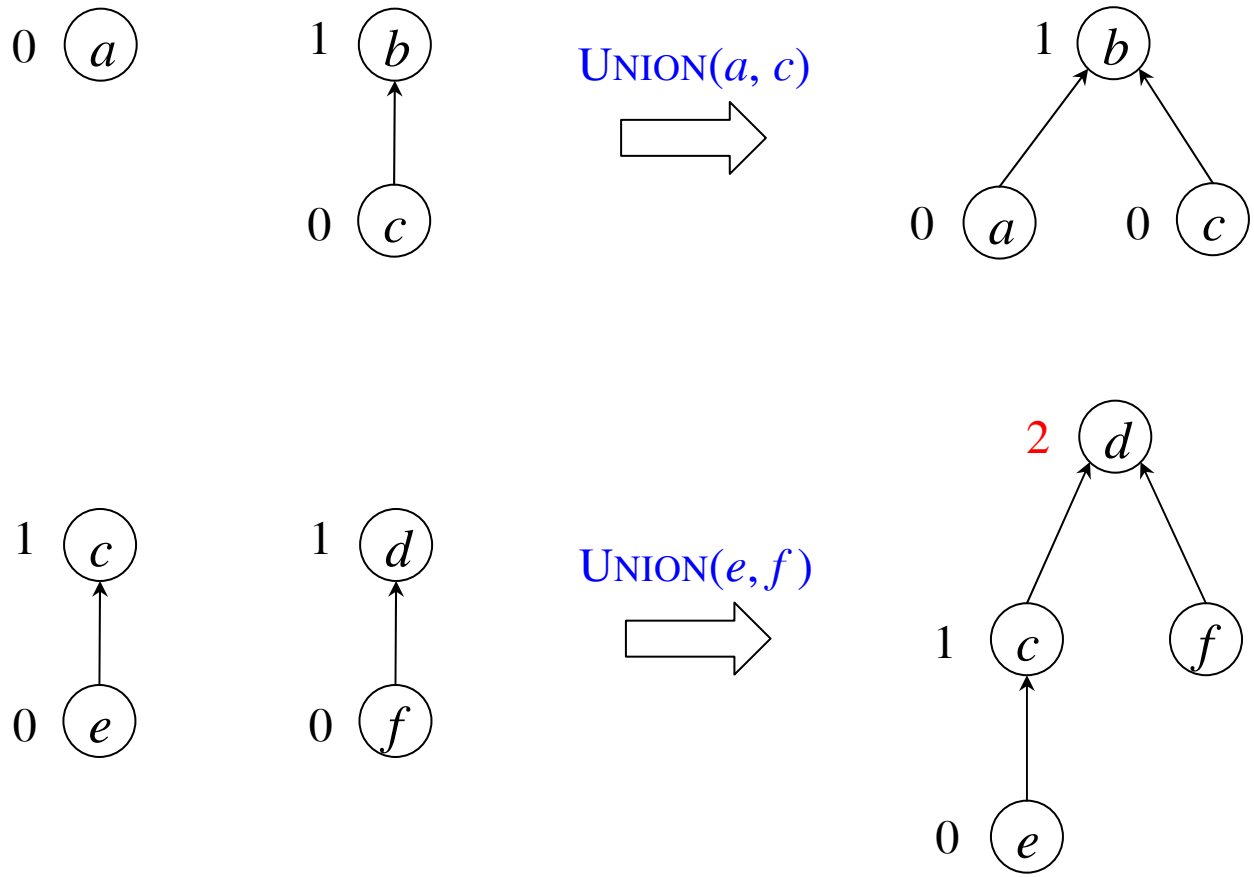
- Dùng hai heuristics để giảm thời gian chạy của các dãy các thao tác lên các tập rời nhau khi hiện thực bằng rừng:
  - Heuristic *hợp theo thứ hạng* (*union by rank*) khi thực thi UNION:
    - duy trì cho mỗi nút một *rank*. *Rank* là cận trên cho độ cao (*) của nút. Mọi nút được khởi tạo với *rank* = 0.
    - khi hợp theo thứ hạng hai cây, nút gốc có *rank* nhỏ hơn được làm thành con của nút gốc có *rank* lớn hơn.

      Nếu hai *rank* bằng nhau thì tùy ý lấy một nút gốc x làm nút cha, tăng thêm 1 *rank* của nút x.
  - Heuristic *nén đường dẫn* (*path compression*)

(*) *Độ cao* của một nút trong một cây là số các cạnh nằm trên đường đi đơn dài nhất từ nút đến một nút lá.

Heuristic hợp theo thứ hạng

- Ví dụ: (số bên cạnh mỗi đối tượng là *rank* của nó.)

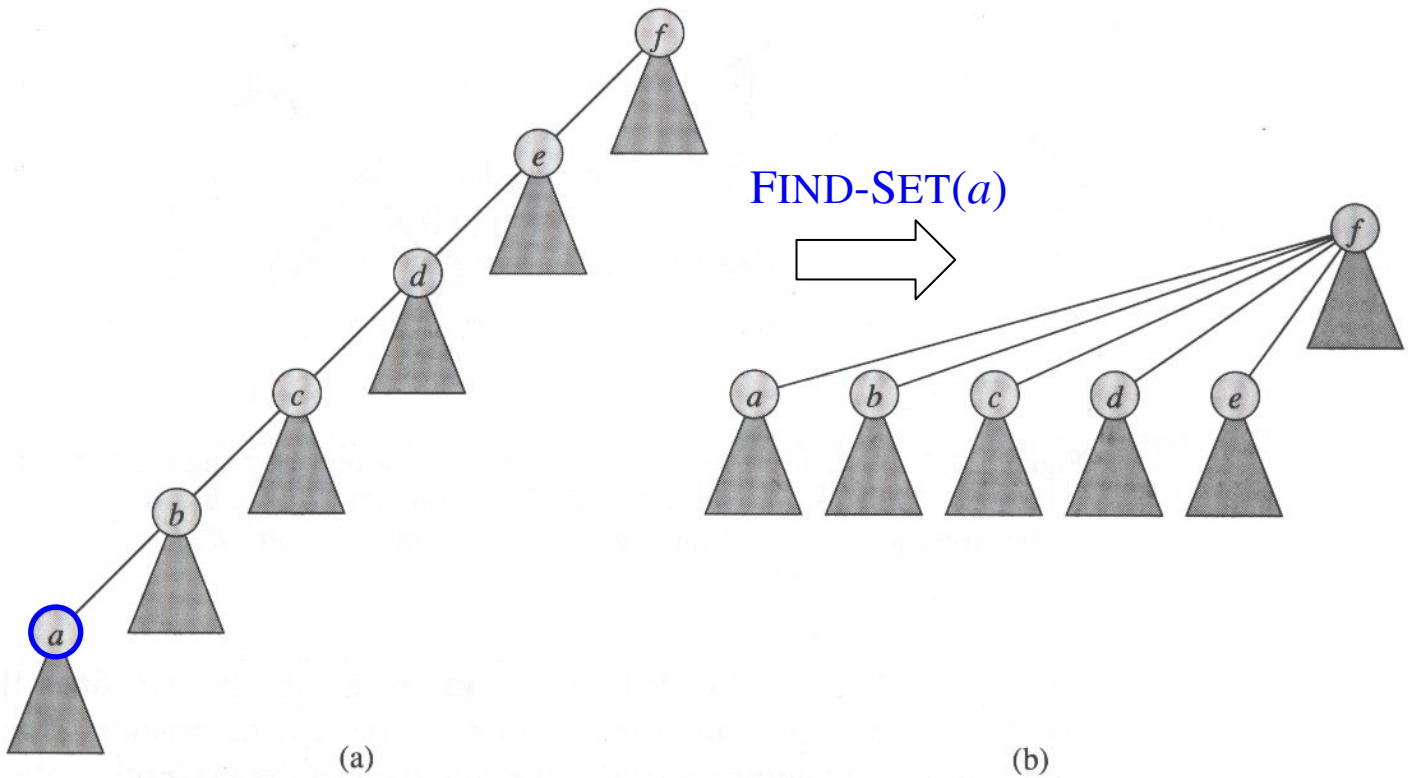


Heuristic nén đường dẫn

– Heuristic *nén đường dẫn* (*path compression*). Chạy qua hai giai đoạn khi thực thi FIND-SET:
  - giai đoạn "chạy lên" để tìm gốc của cây,
  - giai đoạn "chạy xuống" để cập nhật các nút trên đường dẫn để chúng chỉ trực tiếp đến gốc.

Heuristic nén đường dẫn (tiếp)

- Minh họa heuristic nén đường dẫn do thao tác FIND-SET
  - Các hình tam giác tượng trưng các cây con có gốc tại các nút trong hình (a). Mỗi nút có con trỏ chỉ đến nút cha của nó.



Các heuristic hợp theo thứ hạng và nén đường dẫn

- Các thủ tục hiện thực các heuristics hợp theo thứ hạng và nén đường dẫn: MAKE-SET, UNION, và FIND-SET
  - Cha của nút x là $p[x]$.

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

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

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

Các heuristics hợp theo thứ hạng và nén đường dẫn(tiếp)

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

k2

Ảnh hưởng của các heuristic lên thời gian chạy

- Thời gian chạy của một dãy các thao tác gồm m MAKE-SET, UNION, và FIND-SET, trong đó có n thao tác MAKE-SET:
  - Nếu chỉ dùng heuristic hợp theo thứ hạng
    - $O(m \lg n)$
  - Nếu chỉ dùng heuristic nén đường dẫn, với f là số thao tác FIND-SET
    - $\Theta(f \log_{(1+f/n)} n)$ nếu $f \geq n$
    - $\Theta(n + f \lg n)$ nếu $f < n$
  - Nếu dùng cả hai heuristics hợp theo thứ hạng và nén đường dẫn
    - $O(m\, \alpha(m, n))$
      - $\alpha(m, n)$ là hàm đảo của hàm Ackermann
      - trong mọi ứng dụng thực tế, $\alpha(m, n) \leq 4$.

- Không chứng minh các chặn đã liệt kê ở trên.

k2

Show that the running time of the disjoint-set forest with union by rank only is O(mlg n).
First notice that the rank of a node is at most the height of the subtree rooted at that node. By exercise 21:4 - 2 every node has rank at most blg nc and therefore the height h of any subtree is at most
blg nc. Clearly, each call to FIND-SET (and thus UNION) takes at most O(h) = O(lg n) time.

khmt-hung, 1/10/2007

Hàm Ackermann

We are now ready to show Ackermann's function, which is defined for integers $i, j \geq 1$ by

$$
\begin{aligned}
A(1,j) &= 2^j && \text{for } j \geq 1 , \\
A(i,1) &= A(i-1,2) && \text{for } i \geq 2 , \\
A(i,j) &= A(i-1,A(i,j-1)) && \text{for } i,j \geq 2.
\end{aligned}
$$

Thuật toán Kruskal tìm cây khung tối thiểu

- **Make_Set (*v*)**
- **FIND_Set**
- **UNION (*u*, *v*)**

MST_KRUSKAL (G, w)

A ← {} // A sẽ chứa các cạnh của cây khung tối thiểu -MST
for each vertex v in V[G]
  do Make_Set (*v*)
Sort edge of E by nondecreasing weights w
for each edge (*u*, *v*) in E
  do if FIND_SET (*u*) ≠ FIND_SET (*v*)
    then A = AU{(*u*, *v*)}
      UNION (*u*, *v*)
Return A