بسم الله الرحمن الرحیم

طراحی و تحلیل الگوریتم‌ها

مبحث نهم

مباحث پیشرفته در ساختمان داده‌ها

ساختمان‌داده‌هایی برای مجموعه‌های مجزا

Data Structures for Disjoint Sets

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

http://courses.fouladi.ir/algorithm

مجموعه‌های مجزا

Disjoint Sets

برخی کاربردها، نیازمند گروه‌بندی n عنصر مجزا در قالب تعدادی مجموعه‌ی مجزا هستند.

دو عملیات مورد نیاز عبارتند از:

یافتن *Find*	اجتماع *Union*
یافتن مجموعه‌ای که عنصر داده شده در آن قرار دارد	اجتماع دو مجموعه‌ی داده شده

هدف، طراحی ساختمان‌داده‌ای برای پشتیبانی کارآمد از دو عملیات فوق است.

مجموعه‌های مجزا

ساختمان داده

Disjoint Sets

ساختمان داده‌ی مجموعه‌های مجزا، خانواده‌ای از مجموعه‌های پویای مجزای را نگهداری می‌کند.

$$S = \{S_1, S_2, S_3, \ldots, S_k\}$$

هر مجموعه توسط نماینده‌ی آن که عضوی از آن مجموعه است، مشخص می‌شود.

Representative

> در برخی کاربردها اینکه کدام عضو به عنوان نماینده استفاده می‌شود مهم نیست؛ فقط اینکه اگر نماینده‌ی مجموعه‌ی پویا را چند بار بدون تغییر دادن مجموعه درخواست کنیم، باید به پاسخ‌های یکسانی برسیم.

هر عنصر از مجموعه در قالب یک شیئ نمایش داده می‌شود (x)

ایجاد مجموعه	یافتن	اجتماع
MAKE-SET(x)	FIND-SET(x)	UNION(x,y)
یک مجموعه‌ی جدید که تنها عضو آن x است را ایجاد می‌کند. (x نباید عضو مجموعه‌ی دیگری باشد.)	اشاره‌گری به نماینده‌ی مجموعه‌ی شامل x را برمی‌گرداند.	دو مجموعه‌ی پویای مجزا شامل x و y که S_x و S_y نام دارد را در یک مجموعه‌ی جدید که اجتماع دو مجموعه است قرار می‌دهد و سپس S_x و S_y را حذف می‌کند. نماینده‌ی مجموعه‌ی جدید یکی از اعضای S_x یا S_y است.

مجموعه‌های مجزا

کاربرد در یافتن مؤلفه‌های همبند یک گراف بدون جهت

Connected Components

روال Connected-Components

- در آغاز هر رأس v در مجموعه‌ی خودش قرار می‌گیرد.
- سپس برای هر یال (u,v)، اجتماع مجموعه‌هایی که شامل u و v هستند، تعیین می‌شود.

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

روال Same-Component

- پس از اجرای روال قبلی، این روال مشخص می‌کند که دو رأس در یک مؤلفه‌ی همبند قرار دارند یا خیر.

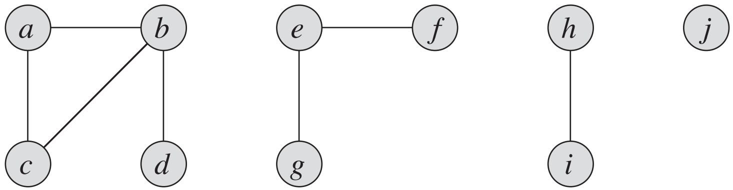
```
SAME-COMPONENT(u, v)
if FIND-SET(u) == FIND-SET(v)
    return TRUE
else return FALSE
```

مجموعه‌های مجزا

کاربرد در یافتن مؤلفه‌های همبند یک گراف بدون جهت : مثال

CONNECTED COMPONENTS

مثال از یک گراف بدون جهت با چهار مؤلفه‌ی همبند:



خانواده‌ی مجموعه‌های مجزا پس از پردازش هر یال توسط روال CONNECTED-COMPONENTS :

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

مجموعه‌های مجزا

بازنمایی با لیست پیوندی

Disjoint Sets

- نمایش هر مجموعه با یک لیست پیوندی
- اولین شیئ در هر لیست پیوندی به عنوان نماینده‌ی مجموعه‌ی آن عمل می‌کند.
- هر لیست دو اشاره‌گر $head$ و $tail$ (اشاره‌گر به سر و ته لیست) دارد.

ساختار هر گره‌ی لیست پیوندی

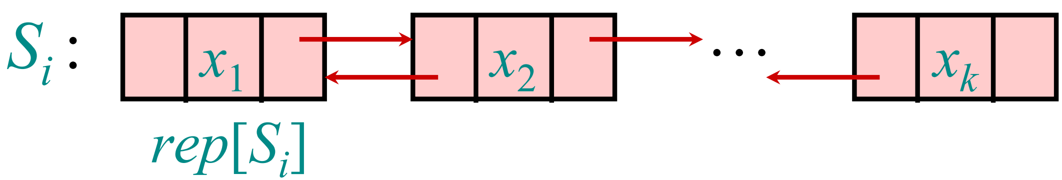
نماینده *Representative*	اشاره‌گری به عضو نماینده $rep[S_x]$
کلید *Key*	عضو مجموعه
بعدی *Next*	اشاره‌گری به شئی حاوی عضو بعدی مجموعه

مجموعه‌های مجزا

بازنمایی با لیست پیوندی: مثال

Disjoint Sets

$$S_i = \{x_1, x_2, \ldots, x_k\}$$

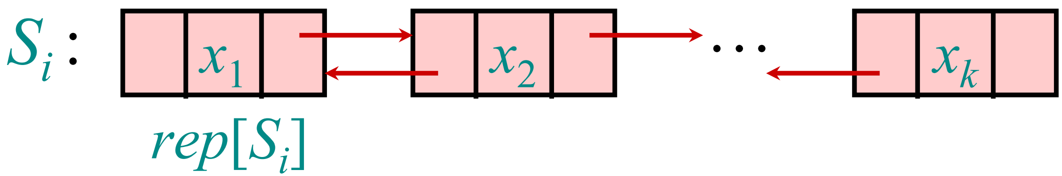


مجموعه‌های مجزا

بازنمایی با لیست پیوندی: تحلیل زمان اجرا

DISJOINT SETS

$$S_i = \{x_1, x_2, \ldots, x_k\}$$



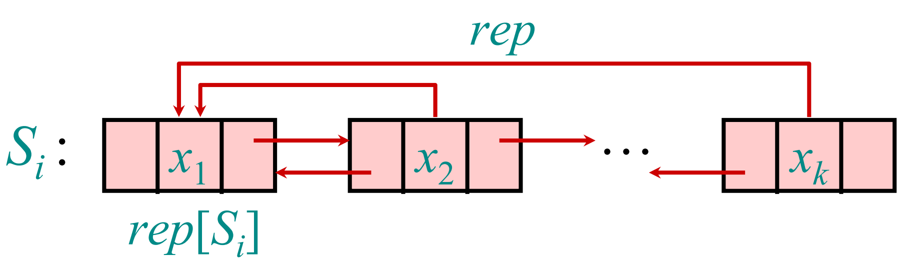
- MAKE-SET(x) initializes x as a lone node. – $\Theta(1)$
- FIND-SET(x) walks left in the list containing x until it reaches the front of the list. – $\Theta(n)$
- UNION(x, y) concatenates the lists containing x and y, leaving rep. as FIND-SET[x]. – $\Theta(n)$

مجموعه‌های مجزا

بازنمایی با لیست پیوندی افزوده

AUGMENTED LINKED-LIST

Store set $S_i = \{x_1, x_2, \ldots, x_k\}$ as unordered doubly linked list. Define $rep[S_i]$ to be front of list, x_1. Each element x_j also stores pointer $rep[x_j]$ to $rep[S_i]$.



- FIND-SET(x) returns $rep[x]$. – $\Theta(1)$
- UNION(x, y) concatenates the lists containing x and y, and updates the rep pointers for all elements in the list containing y. – $\Theta(n)$

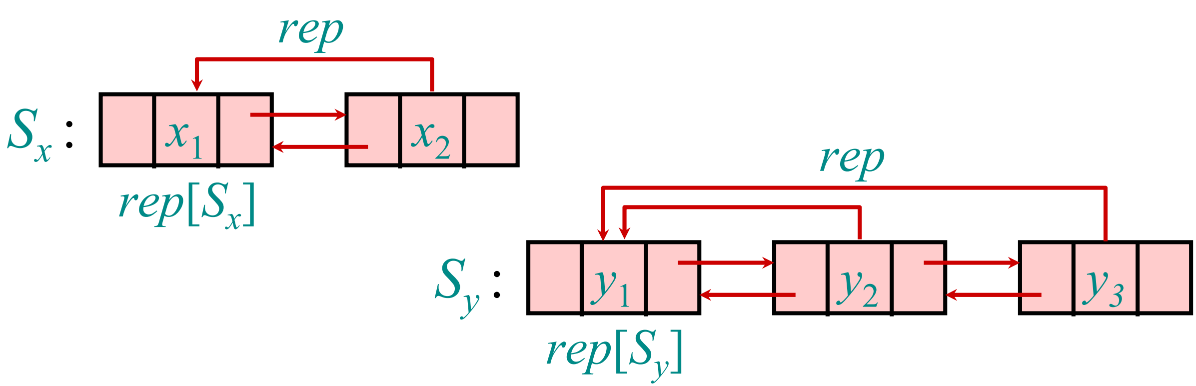
مجموعه‌های مجزا

بازنمایی با لیست پیوندی افزوده: عملیات اجتماع (۱ از ۳)

AUGMENTED LINKED-LIST

Each element x_j stores pointer $rep[x_j]$ to $rep[S_i]$.

UNION(x, y)
- concatenates the lists containing x and y, and
- updates the *rep* pointers for all elements in the list containing y.



مجموعه‌های مجزا

بازنمایی با لیست پیوندی افزوده: عملیات اجتماع (۲ از ۳)

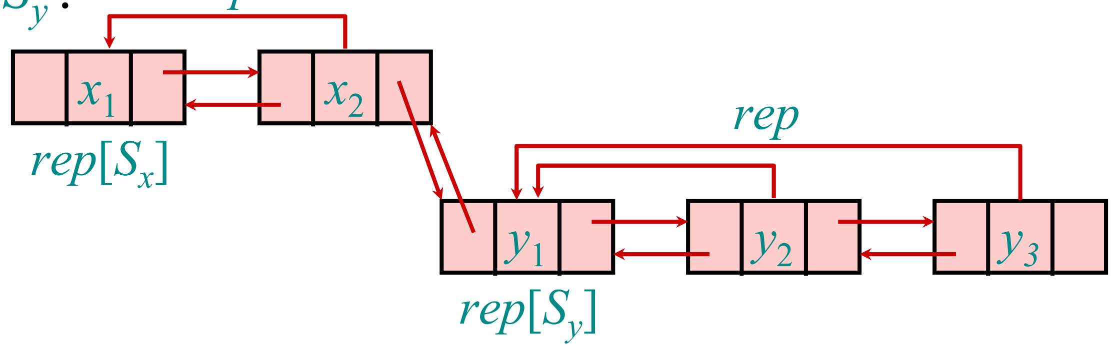
AUGMENTED LINKED-LIST

Each element x_j stores pointer $rep[x_j]$ to $rep[S_i]$.

UNION(x, y)

- concatenates the lists containing x and y, and
- updates the rep pointers for all elements in the list containing y.

$S_x \cup S_y$:



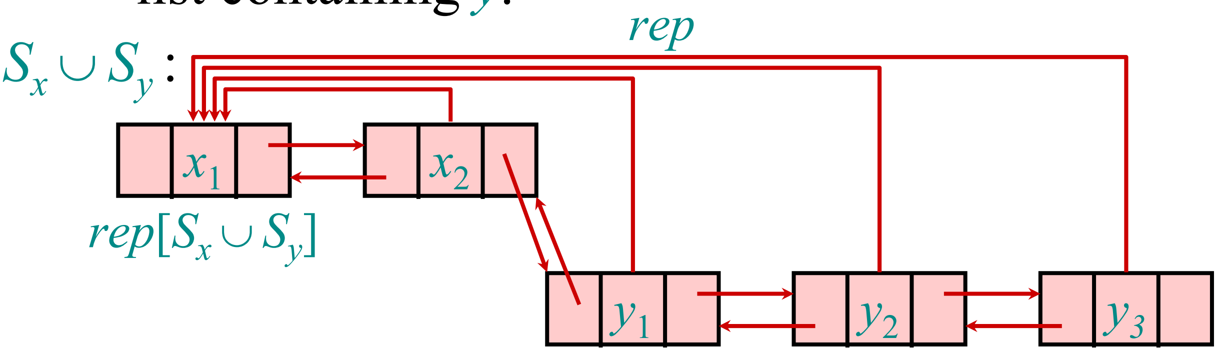
بازنمایی با لیست پیوندی افزوده: عملیات اجتماع (۳ از ۳)

AUGMENTED LINKED-LIST

Each element x_j stores pointer $rep[x_j]$ to $rep[S_i]$.

UNION(x, y)

- concatenates the lists containing x and y, and
- updates the rep pointers for all elements in the list containing y.



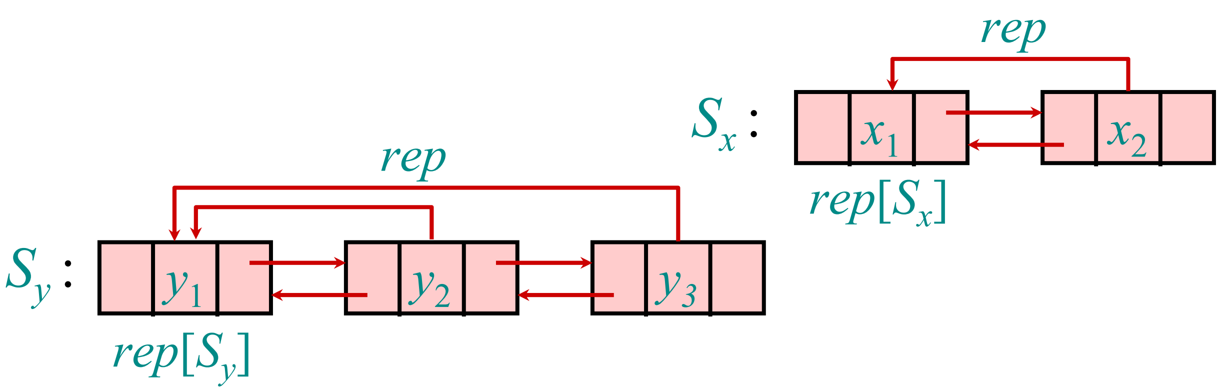
مجموعه‌های مجزا

بازنمایی با لیست پیوندی افزوده: عملیات اجتماع به صورتی دیگر (۱ از ۳)

AUGMENTED LINKED-LIST

UNION(x, y) could instead

- concatenate the lists containing y and x, and
- update the *rep* pointers for all elements in the list containing x.

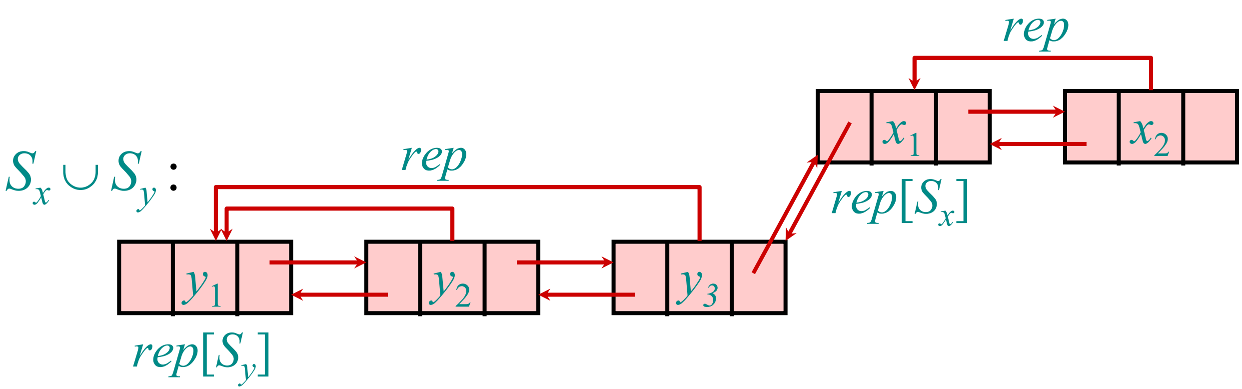


مجموعه‌های مجزا

بازنمایی با لیست پیوندی افزوده: عملیات اجتماع به صورتی دیگر (۲ از ۳)

AUGMENTED LINKED-LIST

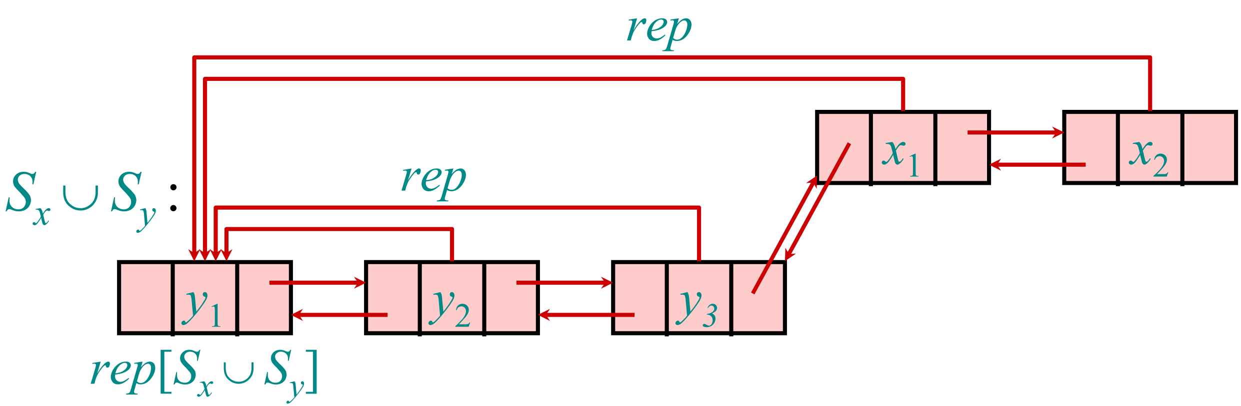
UNION(x, y) could instead
- concatenate the lists containing y and x, and
- update the *rep* pointers for all elements in the list containing x.



بازنمایی با لیست پیوندی افزوده: عملیات اجتماع به صورتی دیگر (۳ از ۳)

AUGMENTED LINKED-LIST

UNION(x, y) could instead
- concatenate the lists containing y and x, and
- update the rep pointers for all elements in the list containing x.

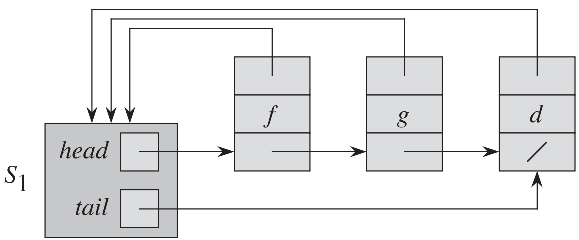


مجموعه‌های مجزا

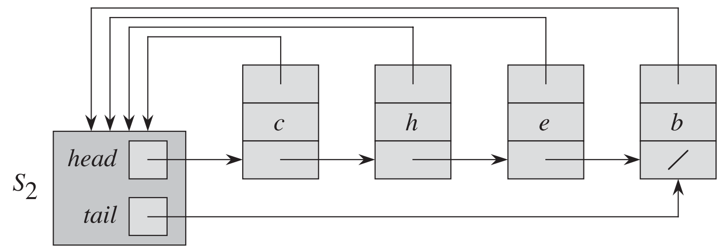
بازنمایی با لیست پیوندی: مثال

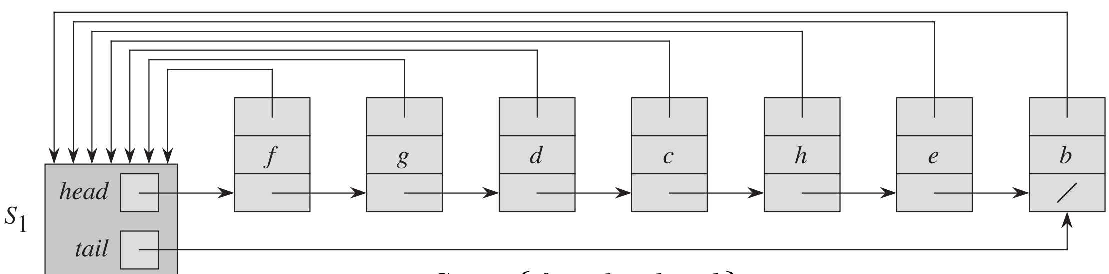
DISJOINT SETS

$$S_1 = \{f, g, d\}$$



$$S_2 = \{c, h, e, b\}$$



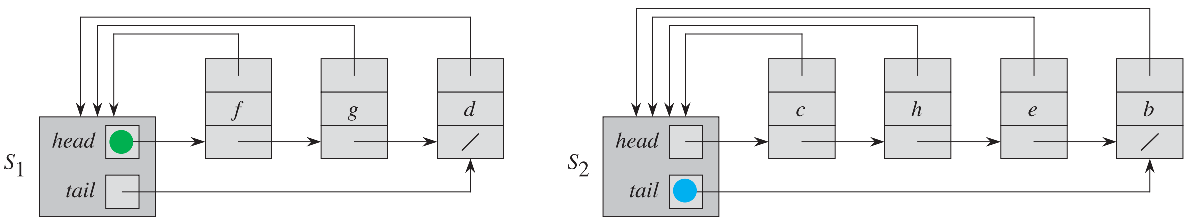


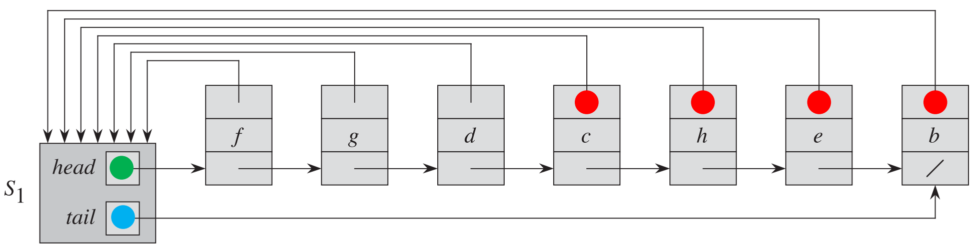
$$S_1 = \{f, g, d, c, h, e, b\}$$

با انجام عمل اجتماع، دو مجموعه‌ی اول حذف شده، و یک مجموعه‌ی جدید از اجتماع آن دو ساخته می‌شود.

مجموعه‌های مجزا

بازنمایی با لیست پیوندی: پیاده‌سازی عمل اجتماع





عملیات $\text{UNION}(x, y)$

- لیست y را به انتهای لیست x اضافه می‌کند.
- نماینده‌ی مجموعه‌ی جدید، عنصری است که در آغاز نماینده‌ی مجموعه‌ی حاوی x بوده است.
- پس باید اشاره‌گر به نماینده، برای هر شیئی که ابتدا در y بوده است، به‌هنگام شود.
  (زمان این کار بر حسب طول لیست y خطی است.)

مجموعه‌های مجزا

بازنمایی با لیست پیوندی: تحلیل سرشکنی

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

- دنباله‌ای از m عملیات روی n شیئ داریم که $m = 2n - 1$
- n عملیات اول، MAKE-SET است
- $n - 1$ عملیات بعدی UNION است.

- زمان اجرای n عملیات اول $\Theta(n)$ است.
- سپس $n - 1$ عملیات UNION(x_i,x_{i+1}) شیئ iام را به‌هنگام می‌کند.
- تعداد کل اشیایی که توسط UNION به‌هنگام می‌شود عبارت است از:

$$\sum_{i=1}^{n-1} i = \Theta(n^2)$$

- پس پیاده‌سازی UNION به زمان $\Theta(n^2)$ نیاز دارد و هزینه‌ی سرشکنی این m عملیات برابر است با:

$$T(n) = \Theta(n^2) \;/\; (2n-1) = \Theta(n)$$

(زمان متوسط اجتماع در بدترین حالت به‌ازای هر فراخوانی $\Theta(n)$ است، زیرا لیست طولانی‌تر به لیست کوتاه‌تر اضافه می‌شود.)

مجموعه‌های مجزا

بازنمایی با لیست پیوندی: هیوریستیک اجتماع وزن‌دار

WEIGHTED-UNION HEURISTIC

روش هیوریستیک اجتماع وزن‌دار (وزن هر لیست = تعداد عناصر آن لیست)،

همیشه لیست کوتاه‌تر را به لیست طولانی‌تر اضافه می‌کند.

$\Downarrow$

اگر هر دو مجموعه $\Omega(n)$ عضو داشته باشد،
عملیات UNION (اجتماع) $\Omega(n)$ زمان مصرف می‌کند.

قضیه

دنباله‌ای از m عملیات

- MAKE-SET
- FIND-SET
- UNION

داریم که n تای آن، MAKE-SET است.

با استفاده از بازنمایی لیست پیوندی مجموعه‌های مجزا
و روش هیوریستیک اجتماع وزن‌دار،
این دنباله از عملیات به زمان زیر نیاز دارد:

$$O(m + n\log n)$$

WEIGHTED-UNION HEURISTIC

To save work, concatenate smaller list onto the end of the larger list. Cost = Θ(length of smaller list). Augment list to store its ***weight*** (# elements).

Let n denote the overall number of elements (equivalently, the number of MAKE-SET operations).
Let m denote the total number of operations.
Let f denote the number of FIND-SET operations.

Theorem: Cost of all UNION's is $O(n \lg n)$.

Corollary: Total cost is $O(m + n \lg n)$.

بازنمایی با لیست پیوندی: هیوریستیک اجتماع وزن‌دار

WEIGHTED-UNION HEURISTIC

To save work, concatenate smaller list onto the end of the larger list. Cost $= \Theta(1 +$ length of smaller list$)$.

Theorem: Total cost of UNION's is $O(n \lg n)$.

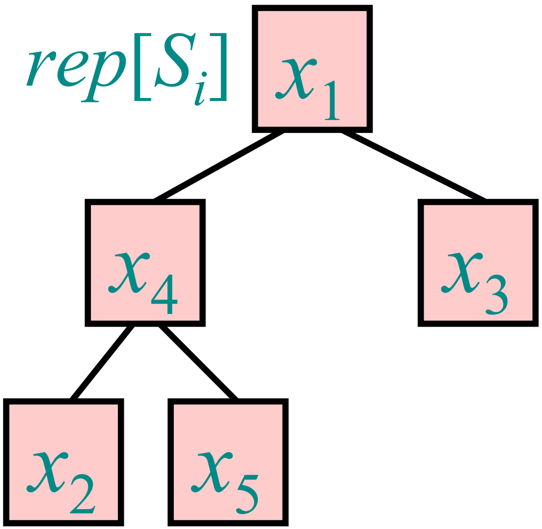
Proof. Monitor an element x and set S_x containing it. After initial MAKE-SET(x), $weight[S_x] = 1$. Each time S_x is united with set S_y, $weight[S_y] \geq weight[S_x]$, pay 1 to update $rep[x]$, and $weight[S_x]$ at least doubles (increasing by $weight[S_y]$). Each time S_y is united with smaller set S_y, pay nothing, and $weight[S_x]$ only increases. Thus pay $\leq \lg n$ for x. $\square$

مجموعه‌های مجزا

بازنمایی با درخت‌های متوازن

BALANCED-TREE REPRESENTATION

$$S_i = \{x_1, x_2, x_3, x_4, x_5\}$$



مجموعه‌های مجزا

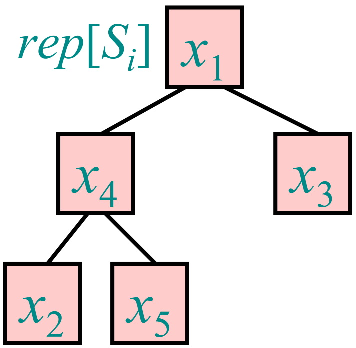
بازنمایی با درخت‌های متوازن: تحلیل زمان اجرا

BALANCED-TREE REPRESENTATION

$$S_i = \{x_1, x_2, \ldots, x_k\}$$

- MAKE-SET(x) initializes x as a lone node. – $\Theta(1)$
- FIND-SET(x) walks up the tree containing x until it reaches the root. – $\Theta(\lg n)$
- UNION(x, y) concatenates the trees containing x and y, changing rep. – $\Theta(\lg n)$

$S_i = \{x_1, x_2, x_3, x_4, x_5\}$



مجموعه‌های مجزا

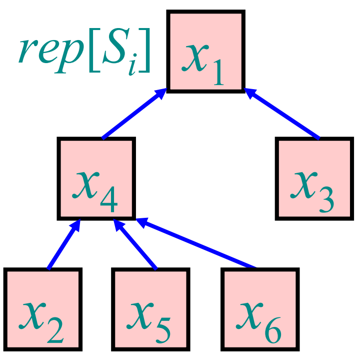
بازنمایی با درخت‌ها

BALANCED-TREE REPRESENTATION

Store each set $S_i = \{x_1, x_2, \ldots, x_k\}$ as an unordered, potentially unbalanced, not necessarily binary tree, storing only *parent* pointers. $rep[S_i]$ is the tree root.

- MAKE-SET(x) initializes x as a lone node. $-\Theta(1)$
- FIND-SET(x) walks up the tree containing x until it reaches the root. $-\Theta(depth[x])$
- UNION(x, y) concatenates the trees containing x and y…

$S_i = \{x_1, x_2, x_3, x_4, x_5, x_6\}$



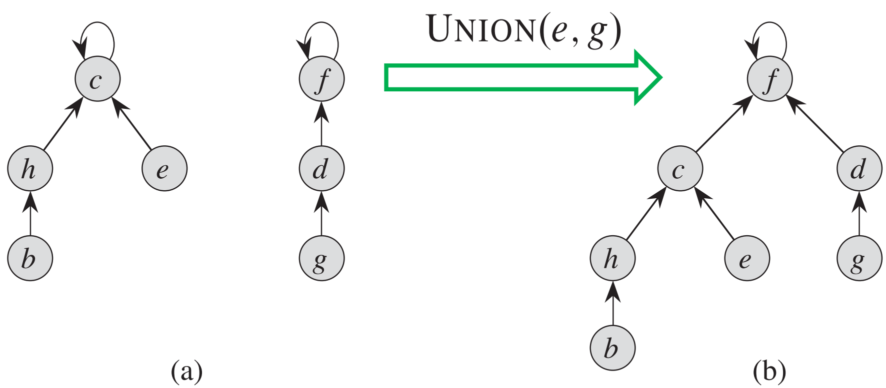
جنگل مجموعه‌های مجزا

Disjoint Sets Forest

یک پیاده‌سازی سریع‌تر مجموعه‌های مجزا،
بازنمایی آنها به صورت درخت‌های ریشه‌دار است:

- هر **گره** نشان‌دهنده‌ی یک عضو است.
- هر **درخت** نشان‌دهنده‌ی یک مجموعه است.

- هر عضو فقط به والدش اشاره می‌کند.
- ریشه‌ی هر درخت، نماینده‌ی مجموعه است.
- ریشه‌ی درخت به خودش اشاره می‌کند.



جنگل مجموعه‌های مجزا

روش هیوریستیک اجتماع بر حسب رتبه

UNION BY RANK

```
MAKE-SET(x)
  x.p = x
  x.rank = 0
```

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

```
LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      if x.rank == y.rank
          y.rank = y.rank + 1
```

ریشه‌ی درخت با تعداد گره‌ی کمتر
به ریشه‌ی درخت با تعداد گره‌ی بیشتر اشاره می‌کند.

برای پیاده‌سازی:
به‌جای ردیابی صریح اندازه‌ی زیردرخت حاصل از هر گره،
برای هر گره‌ی x، رتبه‌ی $rank[x]$ را نگه می‌داریم:
$rank[x]$ = کران بالای ارتفاع گره

- وقتی مجموعه‌ی تک‌عنصری ایجاد می‌شود، رتبه‌ی اولیه‌ی این گره، صفر است.
- هنگام اجرای اجتماع، ریشه‌ای با رتبه‌ی کوچک‌تر به ریشه‌ای با رتبه‌ی بزرگ‌تر اشاره می‌کند. (ولی خود رتبه‌ها تغییر نمی‌کنند.)
- هنگام اجرای اجتماع، اگر رتبه‌ی ریشه‌ها، یکسان بود، به‌طور دلخواه یکی از ریشه‌ها به‌عنوان والد انتخاب می‌شود و رتبه‌ی آن یک واحد افزایش می‌یابد.

رتبه‌ی هر گره در مجموعه‌های مجزا = $\lfloor \log n \rfloor$

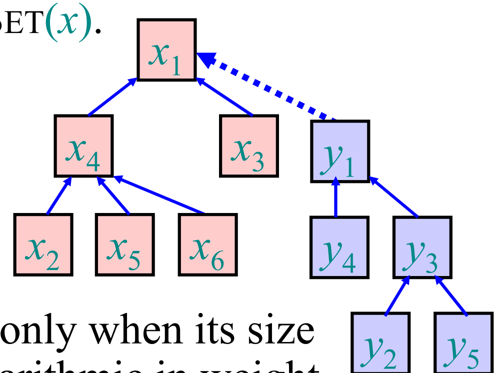
جنگل مجموعه‌های مجزا

روش هیوریستیک اجتماع بر حسب رتبه

UNION BY RANK

Let n denote the overall number of elements (equivalently, the number of MAKE-SET operations).
Let m denote the total number of operations.
Let f denote the number of FIND-SET operations.

UNION(x, y) can use a simple concatenation strategy:
Make root FIND-SET(y) a child of root FIND-SET(x).
$\Rightarrow$ FIND-SET(y) = FIND-SET(x).



Merge tree with smaller weight into tree with larger weight.

Height of tree increases only when its size doubles, so height is logarithmic in weight.
Thus total cost is $O(m + f \lg n)$.

جنگل مجموعه‌های مجزا

روش هیوریستیک اجتماع بر حسب رتبه: تحلیل زمان اجرا

UNION BY RANK

Theorem: Total cost of FIND-SET's is $O(m \lg n)$.

Proof: Amortization by potential function.
The ***weight*** of a node x is # nodes in its subtree.
Define $\phi(x_1, \ldots, x_n) = \Sigma_i \lg weight[x_i]$.
UNION(x_i, x_j) increases potential of root FIND-SET(x_i) by at most $\lg weight[\text{root FIND-SET}(x_j)] \leq \lg n$.
Each step down $p \rightarrow c$ made by FIND-SET(x_i), except the first, moves c's subtree out of p's subtree.
Thus if $weight[c] \geq ½\ weight[p]$, ϕ decreases by ≥ 1, paying for the step down. There can be at most $\lg n$ steps $p \rightarrow c$ for which $weight[c] < ½\ weight[p]$. □

جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن

فشرده‌سازی مسیر در حین عمل FIND-SET استفاده می‌شود.

باید هر گره در **مسیر یافتن**، مستقیماً به ریشه اشاره کند.

فشرده‌سازی مسیر، رتبه‌ها را تغییر نمی‌دهد.

برای پیاده‌سازی:

- در مرحله‌ی اول، در **مسیر یافتن** به سمت بالا می‌رود تا به ریشه برسد.
- در مرحله‌ی دوم، در **مسیر یافتن** به سمت پایین می‌رود تا هر گره را به‌هنگام کند.

PATH COMPRESSION

```
MAKE-SET(x)
    x.p = x
    x.rank = 0
```

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

```
LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1
```

The FIND-SET procedure with path compression

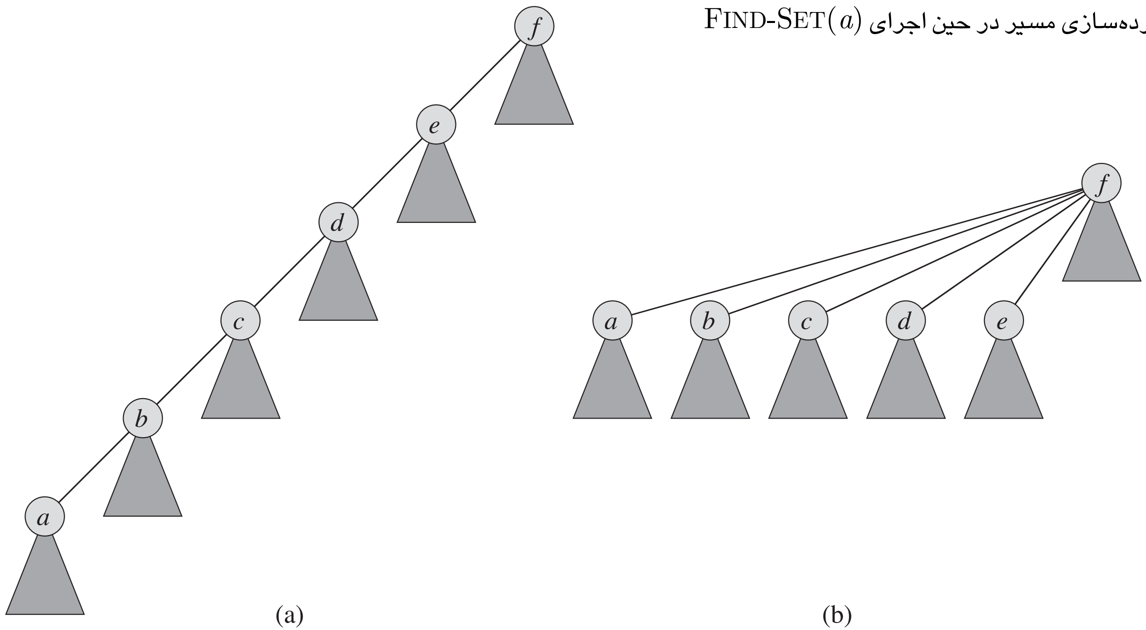
```
FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p
```

جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن

PATH COMPRESSION

فشرده‌سازی مسیر در حین اجرای FIND-SET(a)



درخت بازنمایی کننده‌ی یک مجموعه قبل از اجرای FIND-SET(a)

درخت بازنمایی کننده‌ی همان مجموعه بعد از اجرای FIND-SET(a) با فشرده‌سازی مسیر

جنگل مجموعه‌های مجزا

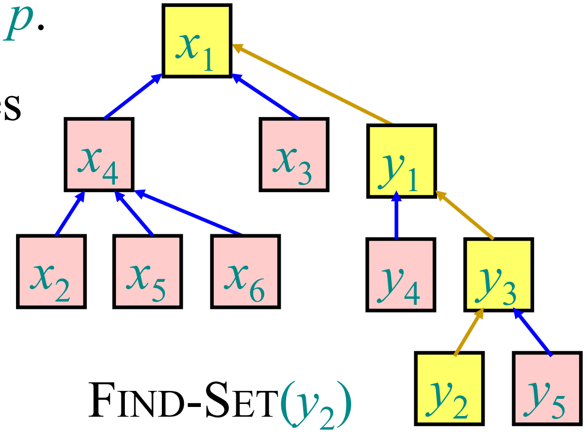
روش هیوریستیک فشرده‌سازی مسیر برای یافتن

PATH COMPRESSION

When we execute a FIND-SET operation and walk up a path p to the root, we know the representative for all the nodes on path p.

Path compression makes all of those nodes direct children of the root.

Cost of FIND-SET(x) is still $\Theta(depth[x])$.

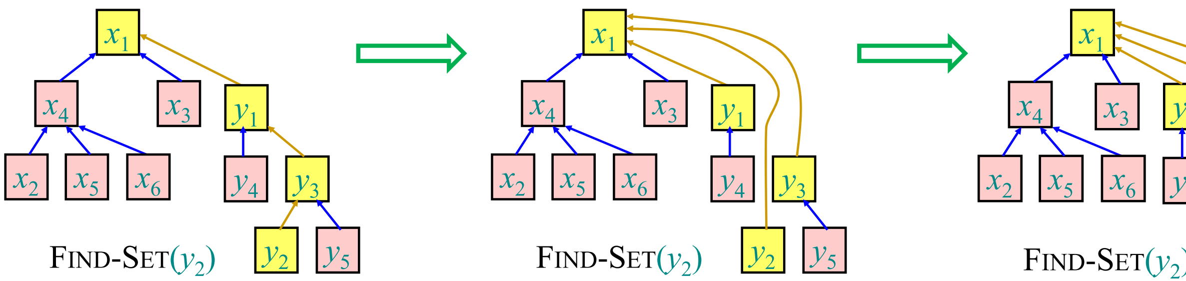


FIND-SET(y_2)

جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن

PATH COMPRESSION



جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن: تحلیل زمان اجرا

PATH COMPRESSION

قضیه

دنباله‌ای از m عملیات

- MAKE-SET
- FIND-SET
- UNION

داریم.
با فرض اینکه تمام عملیات UNION قبل از FIND-SET انجام شود،
با استفاده از بازنمایی درختی مجموعه‌های مجزا
و روش فشرده‌سازی مسیر + اجتماع برحسب رتبه،
این دنباله از عملیات در بدترین حالت به زمان زیر نیاز دارد:

$$O(m)$$

روش هیوریستیک فشرده‌سازی مسیر برای یافتن: تحلیل زمان اجرا

PATH COMPRESSION

Theorem: If all UNION operations occur before all FIND-SET operations, then total cost is $O(m)$.

Proof: If a FIND-SET operation traverses a path with k nodes, costing $O(k)$ time, then $k - 2$ nodes are made new children of the root. This change can happen only once for each of the n elements, so the total cost of FIND-SET is $O(f + n)$. ■

جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن: تحلیل زمان اجرا

PATH COMPRESSION

قضیه

دنباله‌ای از m عملیات
- MAKE-SET
- FIND-SET
- UNION

داریم که
- n تای آن، MAKE-SET است.
- در نتیجه حداکثر $n - 1$ تای آن UNION است.
- f تای آن، FIND-SET است.

با استفاده از بازنمایی درختی مجموعه‌های مجزا
و روش فشرده‌سازی مسیر (به‌تنهایی)،
این دنباله از عملیات در بدترین حالت به زمان زیر نیاز دارد:

$$\Theta\left(n + f \cdot (1 + \log_{2+\frac{f}{n}} n)\right)$$

جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن: تحلیل زمان اجرا

PATH COMPRESSION

قضیه

دنباله‌ای از m عملیات
- MAKE-SET
- FIND-SET
- UNION

داریم که
- n تای آن، MAKE-SET است.
- در نتیجه حداکثر $n-1$ تای آن UNION است.
- f تای آن، FIND-SET است.

با استفاده از بازنمایی درختی مجموعه‌های مجزا
و روش فشرده‌سازی مسیر + اجتماع بر حسب رتبه،
این دنباله از عملیات در بدترین حالت به زمان زیر نیاز دارد:

$$O(m \cdot \alpha(n))$$

$\alpha(n)$ معکوس تابع آکرمن، تابعی با رشد بسیار کند است
که در هر کاربرد واقعی از مجموعه‌های مجزا $\alpha(n) \leq 4$ است.

Ackermann's function A

Define $A_k(j) = \begin{cases} j+1 & \text{if } k = 0, \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1. \end{cases}$ – iterate $j+1$ times

$A_0(j) = j + 1$ $\qquad$ $A_0(1) = 2$

$A_1(j) \sim 2j$ $\qquad$ $A_1(1) = 3$

$A_2(j) \sim 2j\, 2^j > 2^j$ $\qquad$ $A_2(1) = 7$

$A_3(1) = 2047$

$A_3(j) > \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^j}}}}}}\right\} j$

$A_4(1) > \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^{2047}}}}}}}\right\} 2048$

$A_4(j)$ is a lot bigger.

Define $\alpha(n) = \min\,\{k : A_k(1) \geq n\} \leq 4$ for practical n.

Prepared by Kazim Fouladi | Fall 2016 | 2nd Edition
کاظم فولادی

جنگل مجموعه‌های مجزا

روش هیوریستیک فشرده‌سازی مسیر برای یافتن: تحلیل زمان اجرا

PATH COMPRESSION

Theorem: In general, total cost is $O(m\,\alpha(n))$.

(long, tricky proof – see Section 21.4 of CLRS)