# Data Structure for Disjoint Sets

情報知識ネットワーク研究室 B4
大泉翼

# 目次

# 目次

# 内容

- 素集合データ構造は, グループ分けを管理するデータ構造の総称.
  - 無向グラフにおける**連結成分**を求めるアルゴリズムや, 根付き木における**最近共通祖先**を求めるアルゴリズム, **最小全域木**を求めるアルゴリズムなどに応用.

- 本スライドでは以下の二つの表現を紹介.
  - 素集合連結リスト
  - 素集合森

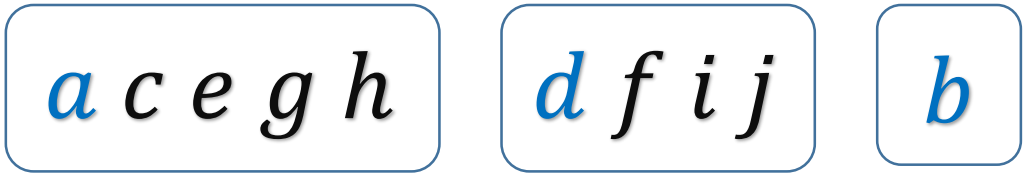
- さらに素集合森の 1 操作あたりのならし時間計算量を解析して, $O(\alpha(n))$ となる事実を示す.

# 素集合データ構造とは



- 素集合データ構造とは, 動的で互いに素な集合族 $\mathcal{S} = \{S_1, S_2, \dots\}$ を管理するデータ構造.
  - 各集合には代表元を定める.
- $x, y$ を集合の要素として, 以下の三つの操作をサポート.

- $MAKE\text{-}SET(x)$
- $UNION(x, y)$
- $FIND\text{-}SET(x)$



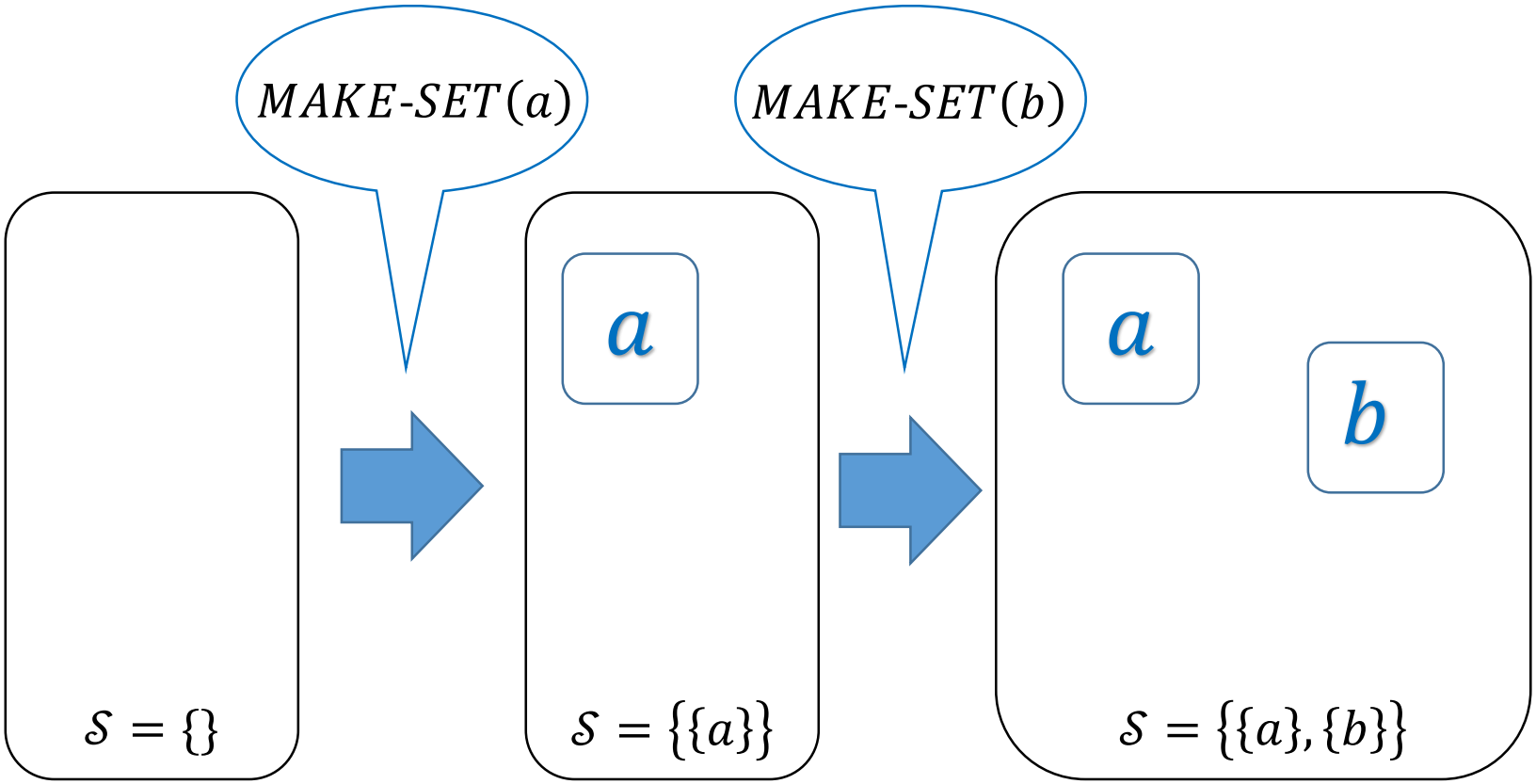

青色は, 各集合の**代表元**を表す.

$$\mathcal{S} = \{S_a, S_d, S_b\}$$
$$S_a = \{a, c, e, g, h\}$$
$$S_d = \{d, f, i, j\}$$
$$S_b = \{b\}$$

# 操作 $MAKE\text{-}SET(x)$



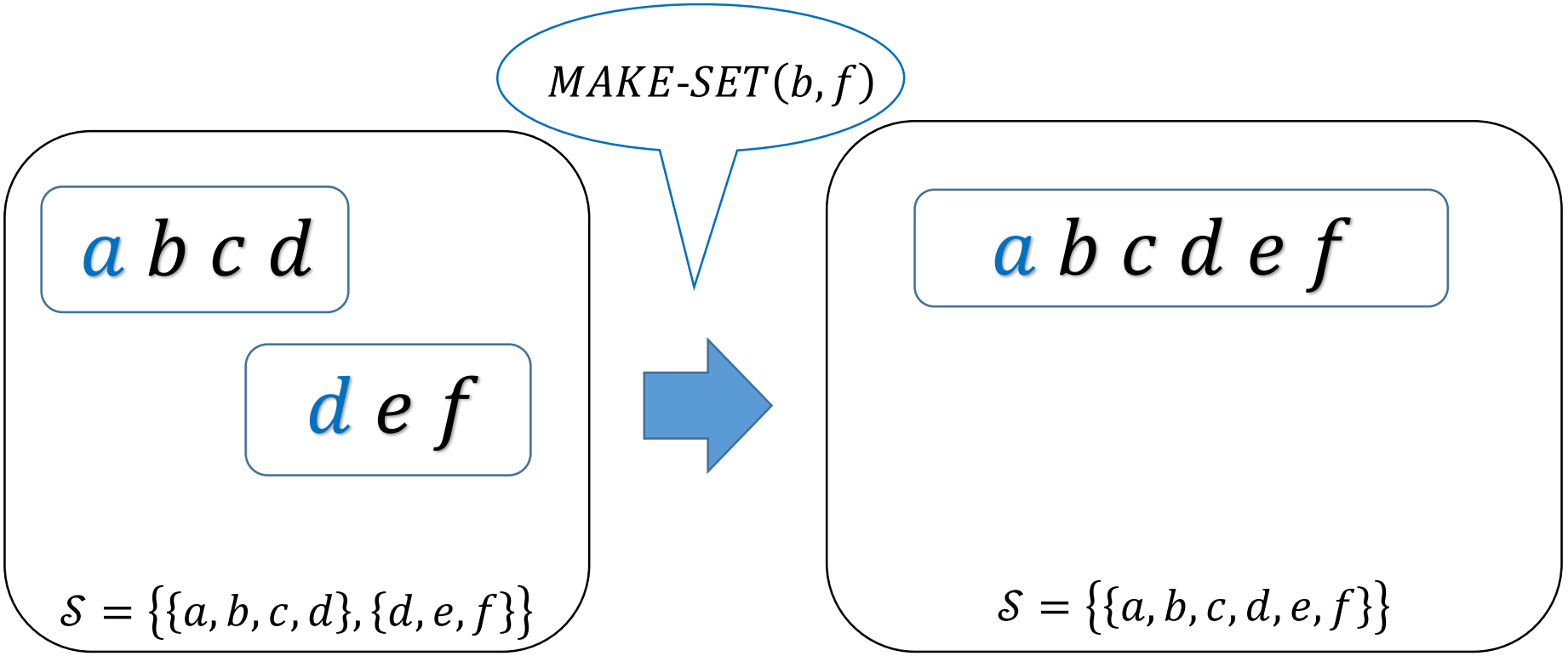
- $S' \leftarrow S \cup \{\{x\}\}$ とする操作.
  - $x$ のみを要素とする新たな集合を $S$ に加える操作.
  - 集合 $\{x\}$ の**代表元**は $x$ と定める.



# 操作 $UNION(x, y)$



- $S' \leftarrow (S \setminus \{S_x, S_y\}) \cup (\{S_x \cup S_y\})$ とする操作.
  - $x, y$ を含む集合をそれぞれ $S_x, S_y$ とする.
  - $S_x, S_y$ を $S$ から取り除き, $S_x \cup S_y$ を $S$ に加える操作.
  - $S_x \cup S_y$ の代表元は, 操作前の $S_x$ における代表元とする.

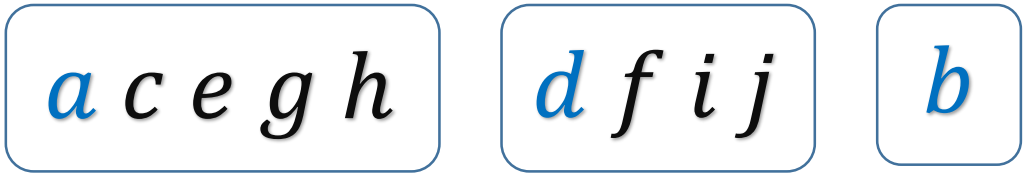


# 操作 $FIND\text{-}SET(x)$



- $S_x$ の**代表元**を取得する操作.
  - $x$ を含む集合を $S_x$ とする.

$FIND\text{-}SET(x)$ と $FIND\text{-}SET(y)$ を比較することで,
$x$ と $y$ が**同じ集合に属するかどうか**を判定できる.




$FIND\text{-}SET(c), FIND\text{-}SET(h)$ から,
同じ代表元 $a$ を得るので,
$c$ と $h$ は**同じ集合に属すること**がわかる

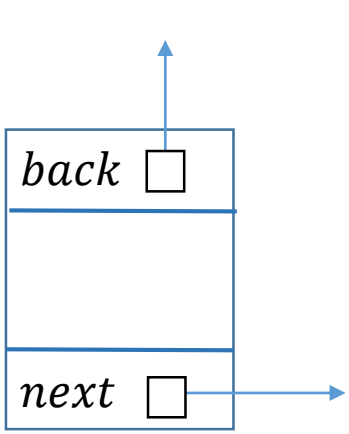
$$\mathcal{S} = \{S_a, S_d, S_b\}$$
$$S_a = \{a, c, e, g, h\}$$
$$S_d = \{d, f, i, j\}$$
$$S_b = \{b\}$$

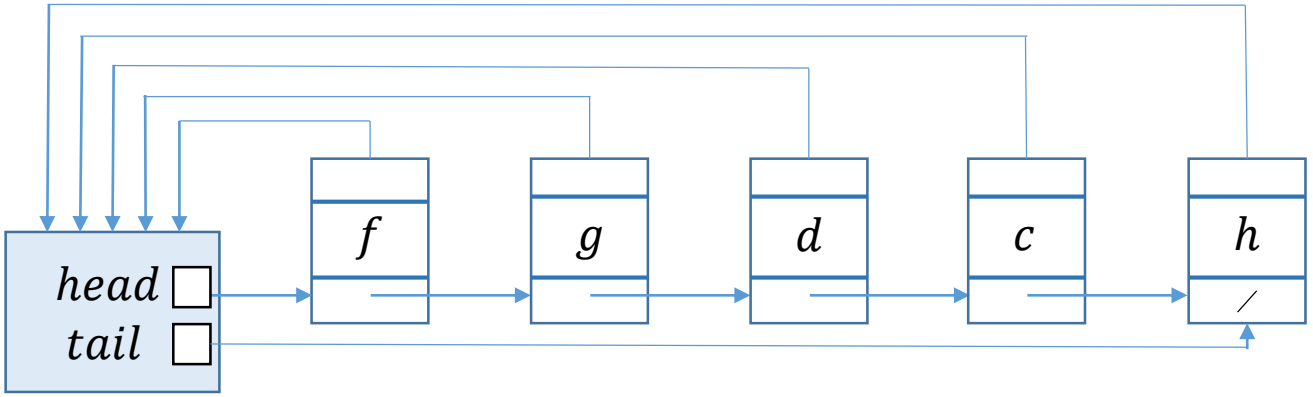
# 目次

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ($back$) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.



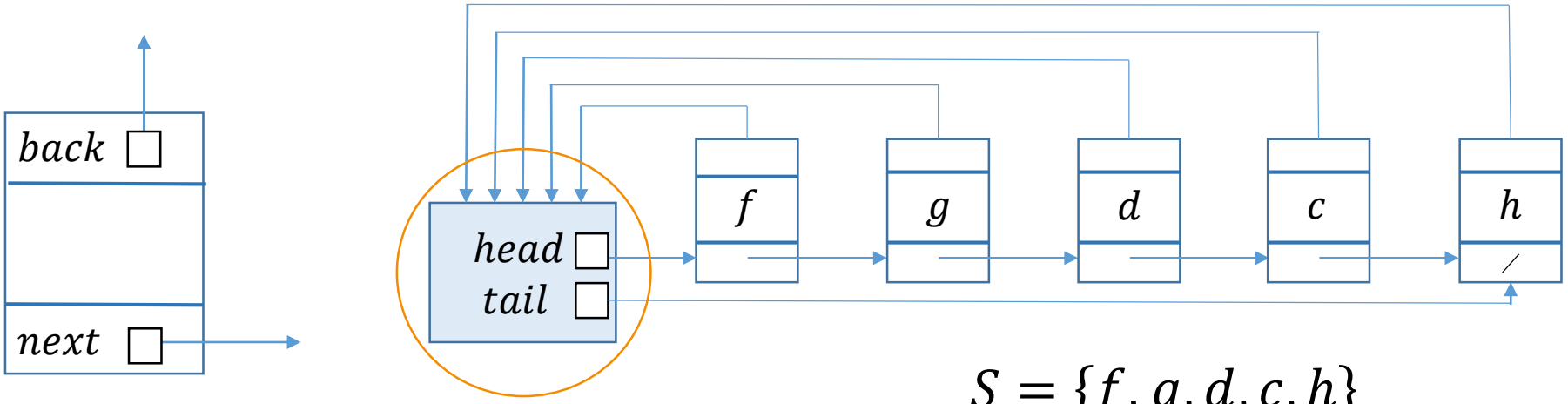




$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つの**ダミー**と複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ ($back$) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.

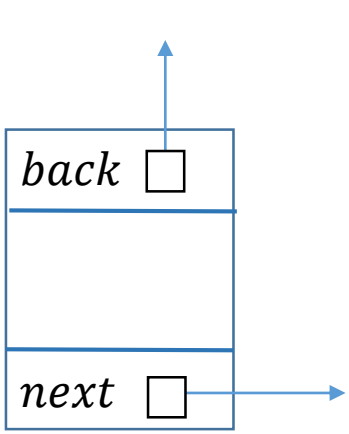


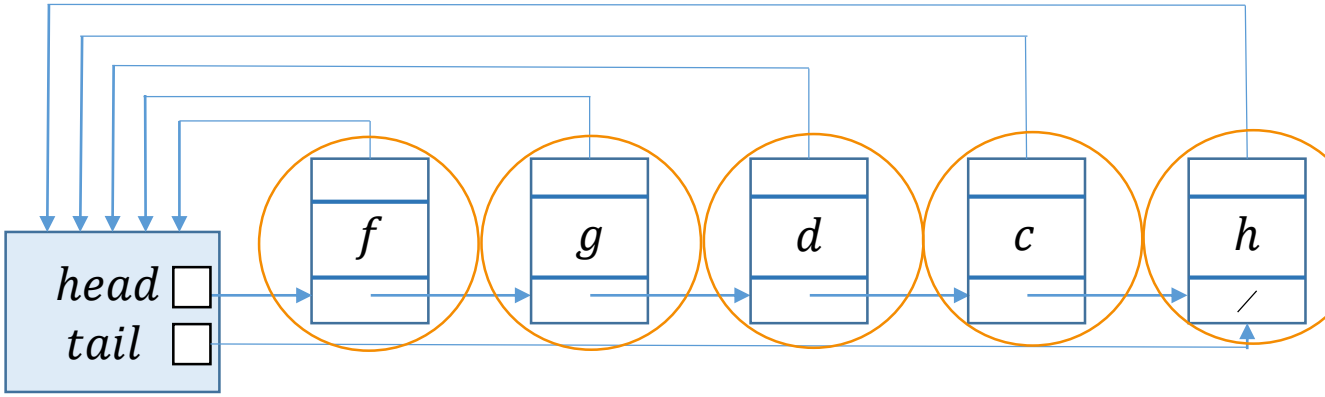

$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数の**ノード**を持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ ($back$) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.



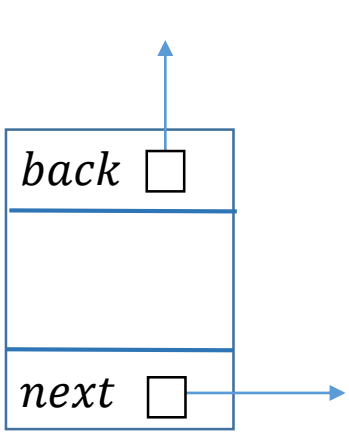



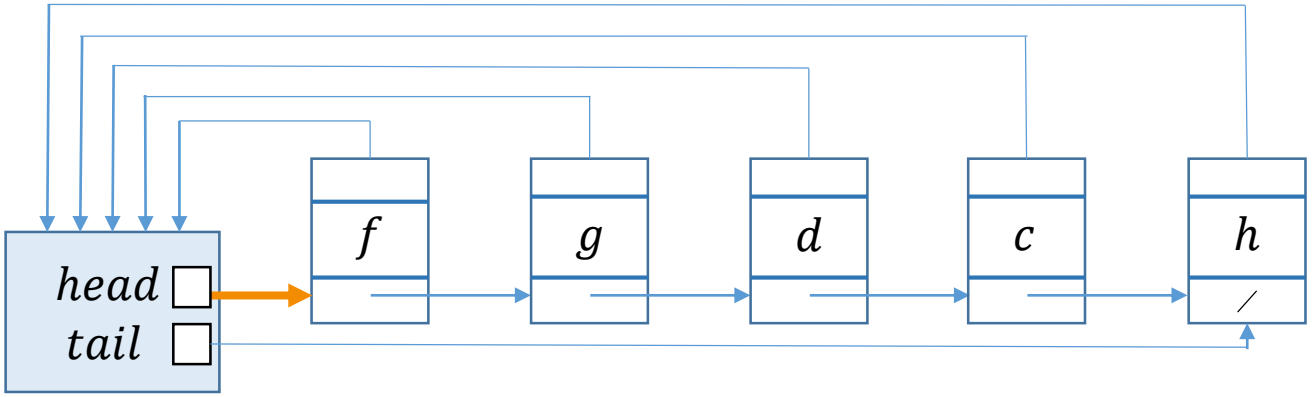

$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ(***head***)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ ($back$) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.



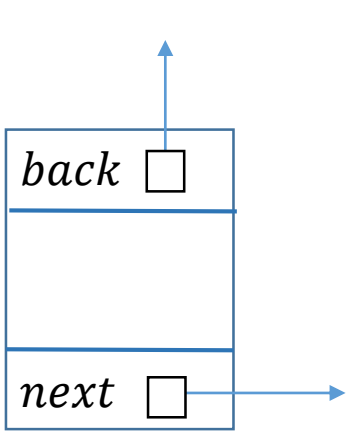



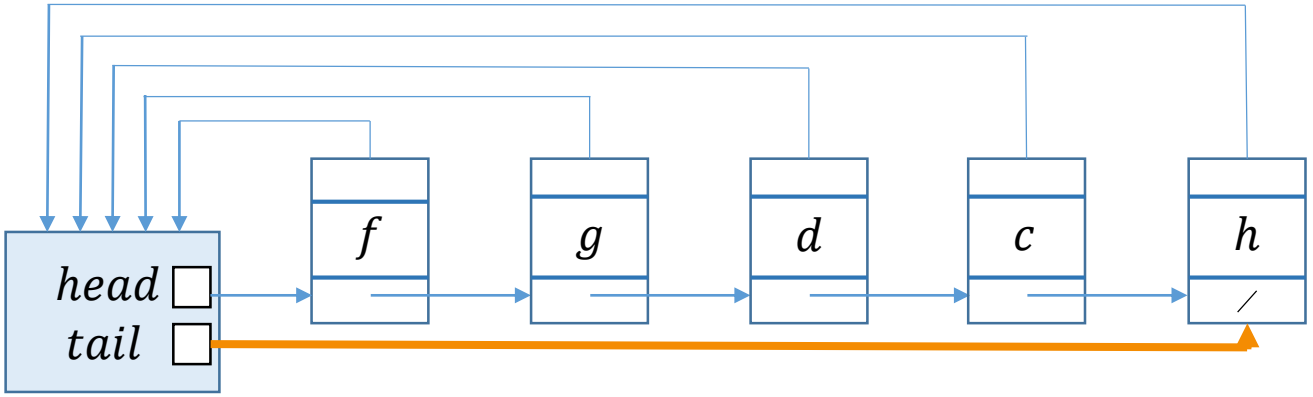

$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ(***tail***) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ($back$) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.



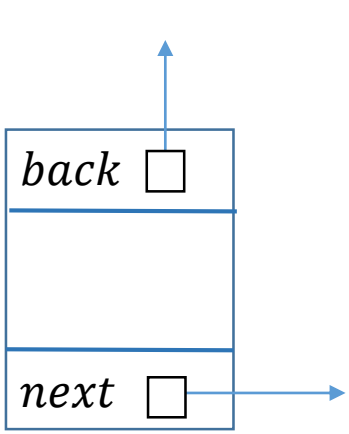



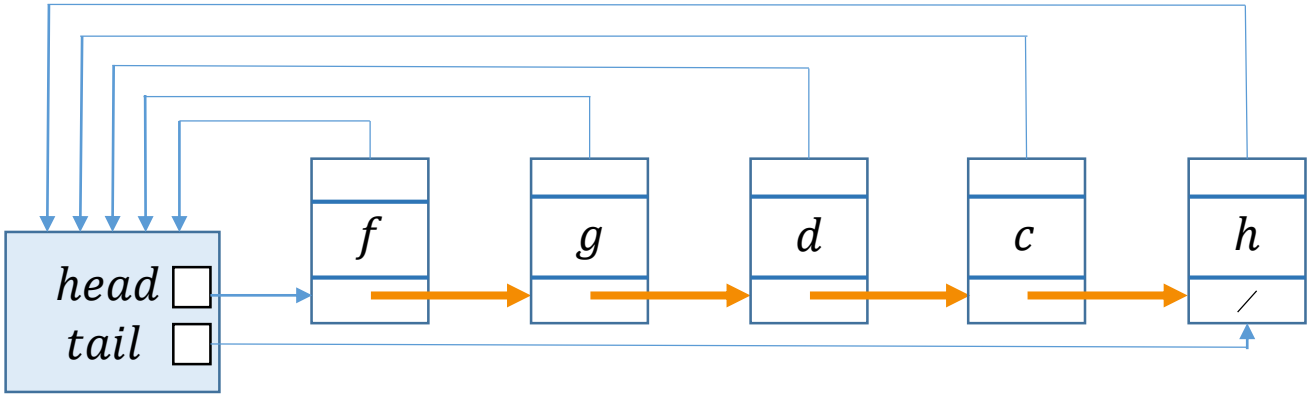

$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ(***next***)と, ダミーへのポインタ($back$) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.



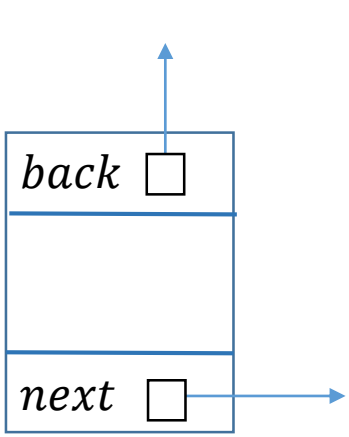



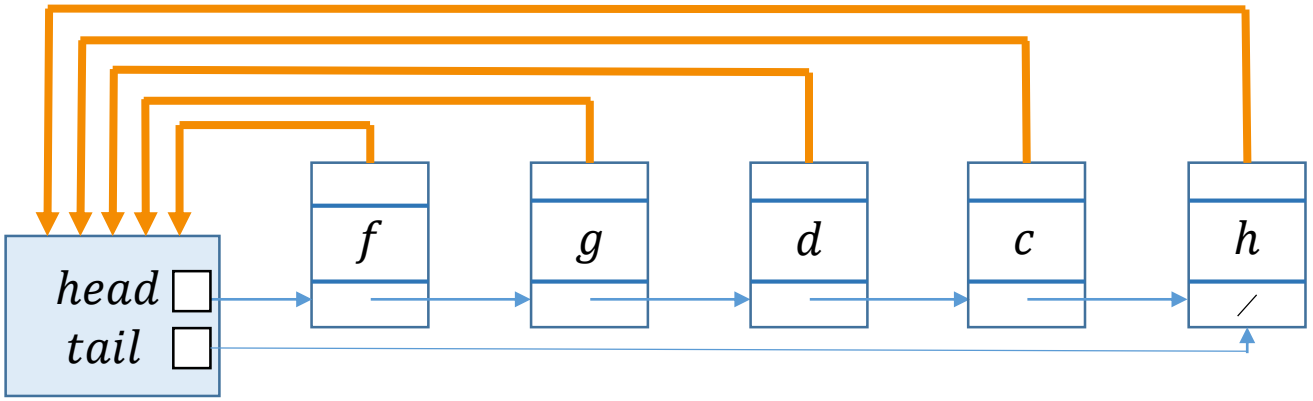

$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ(***back***) と, 要素を持つ.
  - 各集合の代表元は先頭ノードの要素とする.



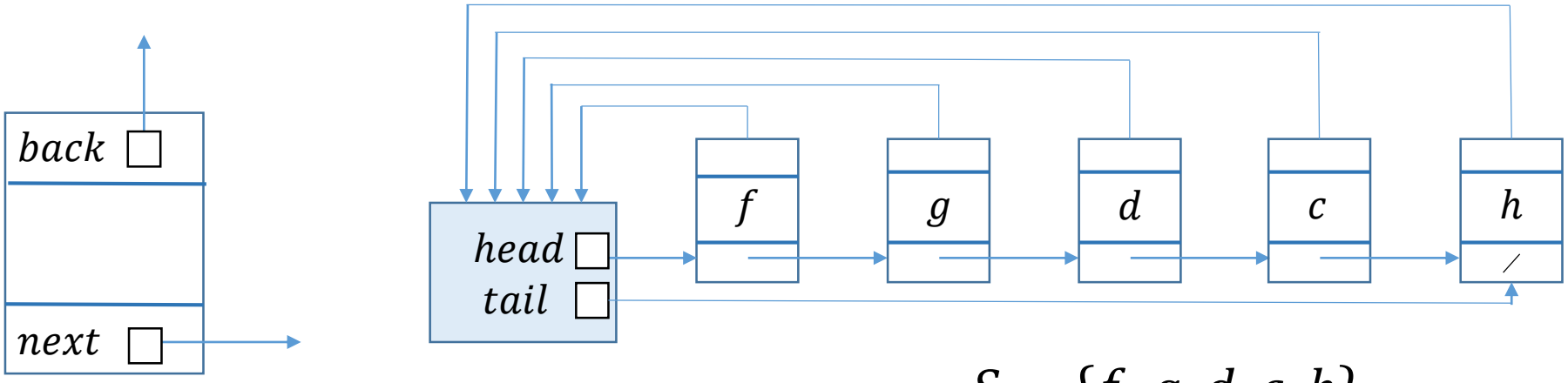




$$S = \{f, g, d, c, h\}$$

# 素集合連結リスト



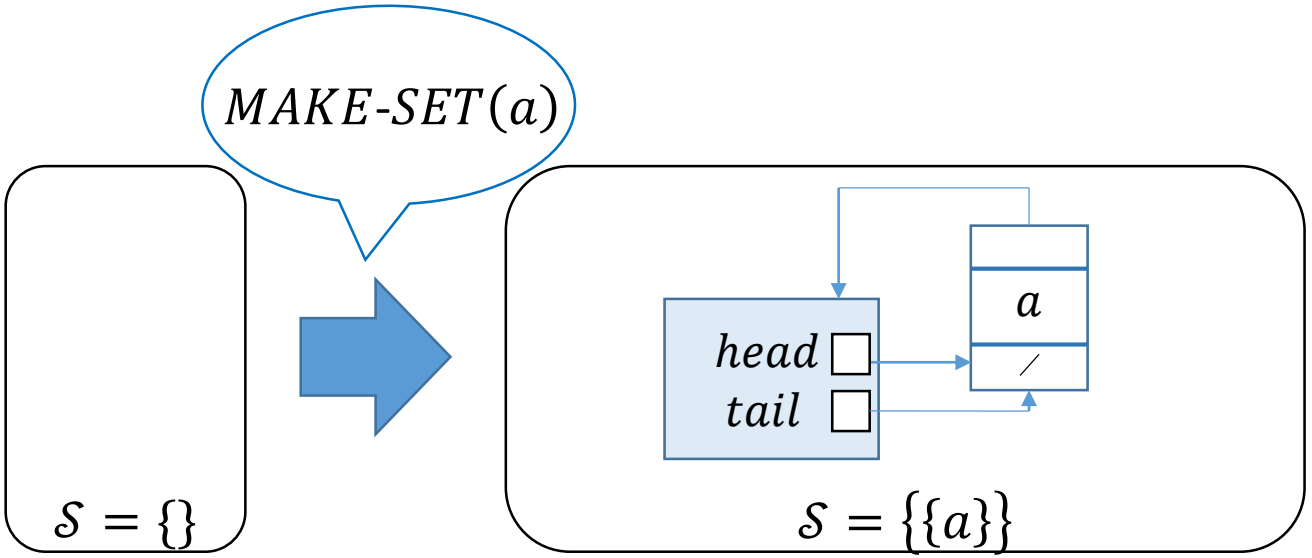
- 素集合連結リスト $\mathcal{L} = \{L_1, L_2, \ldots\}$ では, 各集合を**連結リスト**で表現.
  - 各連結リストは一つのダミーと複数のノードを持つ.
  - ダミーは先頭ノードへのポインタ($head$)と, 末尾ノードへのポインタ($tail$) を持つ.
  - ノードは次のノードへのポインタ($next$)と, ダミーへのポインタ($back$) と, **要素**を持つ.
  - 各集合の代表元は先頭ノードの要素とする.




$$S = \{f, g, d, c, h\}$$

# 操作 $MAKE\text{-}SET(x)$

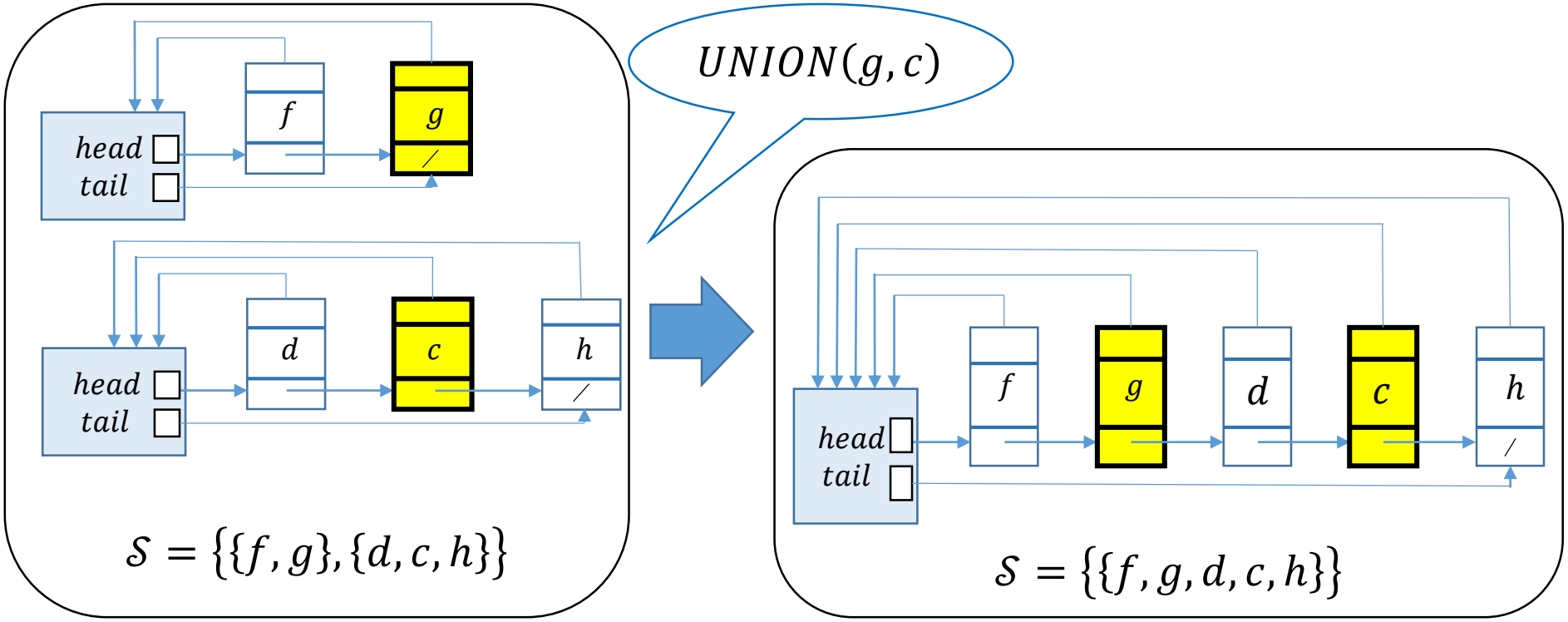
- 新たな連結リストを作る.
  1. 新たなダミーと, 要素が $x$ の新たなノード を作る.
  2. このダミーの $head$ と $tail$ をこのノードに**初期化**.
  3. このノードの $next$ は $NULL$ に, $back$ はこのダミーに**初期化**.



# 操作 $UNION(x, y)$

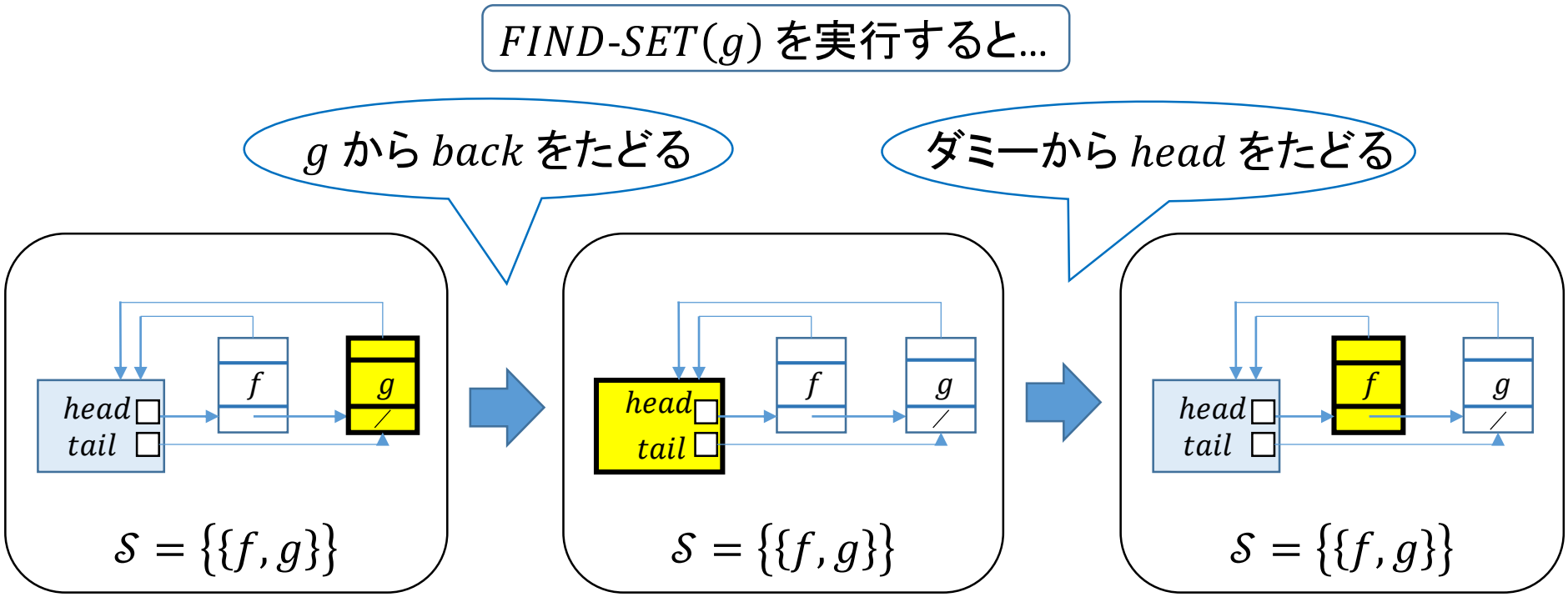


- 二つのリストのポインタを張り替えて, 統合する.
  - $x, y$ を含むリストをそれぞれ $L_x, L_y$ とする.
  1. $L_x$ の末尾ノードの $next$ を $L_y$ の先頭ノードに**更新**.
  2. $L_x$ のダミーの $tail$ を $L_y$ の末尾ノードに**更新**.
  3. $L_y$ のすべてのノードの $back$ を $L_x$ のダミーに**更新**.



# 操作 $FIND\text{-}SET(x)$

- $x$ からポインタをたどり, 最初のノード(**代表元**)を得る.
  - $x$ を含むリストを $L_x$ とする.
  1. $x$ の $back$ をたどり, $L_x$ のダミーを得る.
  2. $L_x$ のダミーから $head$ をたどり $L_x$ の**代表元**を得る.



# ならし計算量



- データ構造における $m$ 回の連続した操作にかかる時間計算量が $O(T(m))$ であるとき,
  $O\left(\frac{T(m)}{m}\right)$ をならし計算量とする.
  - 1 回あたりの操作にかかる時間計算量の効率をはかる指標.

# 計算量解析のための準備



- $n \coloneqq MAKE\text{-}SET(x)$ を実行した回数
- $f \coloneqq FIND\text{-}SET(x)$ を実行した回数
- $m \coloneqq$ 実行した操作回数の合計

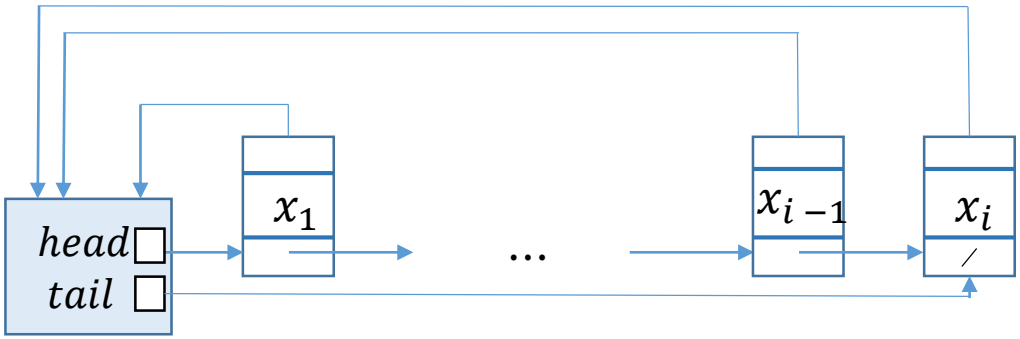
# 計算効率 (工夫なし)



- 右のような $2n-1$ 回の操作列を考える.
- $n$ 回の $MAKE\text{-}SET(x)$ を行うのに $\Theta(n)$.
- $n-1$ 回の $UNION(x,y)$ を行うのに $\Theta(n^2)$.

よって, 操作 1 回の
ならし計算量は $\Theta(n)$

$UNION(i+1,\ i)$ で $i$ 箇所の
$back$ ポインタの更新が起こる.

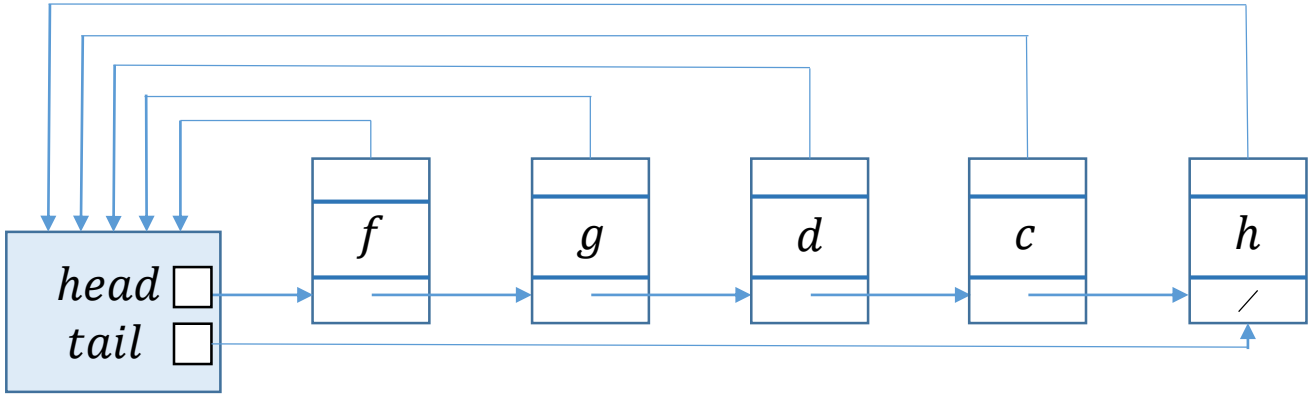



| 操作 | 更新が起こるノードの数 |
|---|---|
| $MAKE\text{-}SET(x_1)$ | 1 |
| ⋮ | ⋮ |
| $MAKE\text{-}SET(x_n)$ | 1 |
| $UNION(x_2, x_1)$ | 1 |
| $UNION(x_3, x_2)$ | 2 |
| $UNION(x_4, x_3)$ | 3 |
| ⋮ | ⋮ |
| $UNION(x_n, x_{n-1})$ | $n-1$ |

# 計算量を改善する工夫



- 各リストが持つノードの個数を 長さ とする.
- $UNION$ 操作の際, 長さの小さいほうのリストを
  大きいほうのリストに付加するようにする.
  - つまり, 長さが小さいほうのリストの $back$ ポインタを張り替える.
  - $length(L)$ ≔ リスト $L$ の長さ.

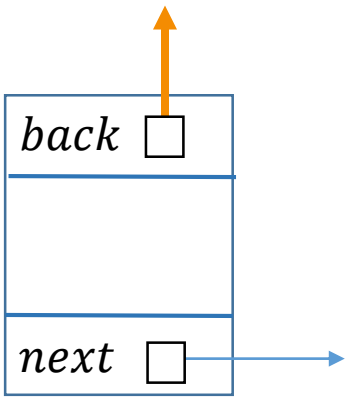


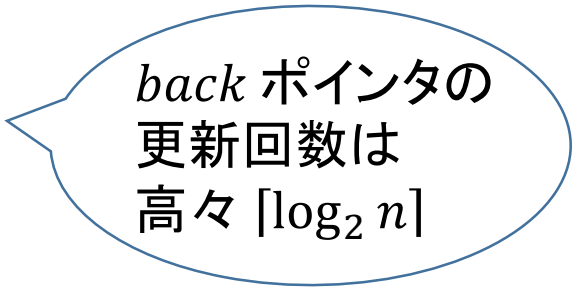

$length(L) = 5$

# 計算効率 (工夫あり)



- あるノード $x$ の $back$ ポインタが, 全体で更新される回数に注目.
- $x$ の $back$ ポインタが張り替わるとき, そのリストの長さは少なくとも 2 倍になる.
- リストの長さは高々 $n - 1$ であるので
  各ノードの $back$ ポインタの更新回数 は高々 $[\log_2 n]$.
- よって $UNION$ 操作全体の時間計算量は $O(n \log n)$.
- $MAKE\text{-}SET$ と $FIND\text{-}SET$ の時間計算量は $O(1)$ であるので
  操作 1 回のならし計算量は $O(\log n)$.



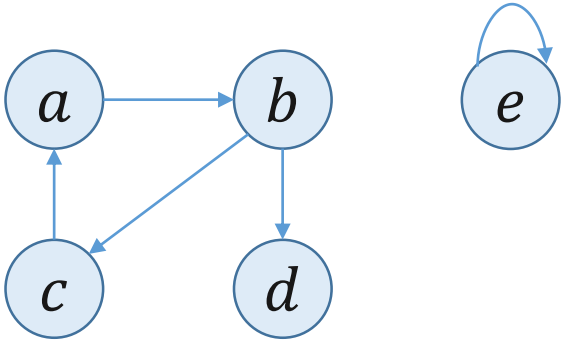

$L_x$ : $x$ を含むリスト

| $x$ の更新回数 | $L_x$ の長さ |
|---|---|
| 1 | 少なくとも 2 |
| 2 | 少なくとも 4 |
| 3 | 少なくとも 8 |
| ⋮ | ⋮ |
| $[\log_2 k]$ | 少なくとも $k$ |

# 目次

# (準備 1/3) 有向グラフ

- **有向グラフ** $G$ とは, **有限集合** $V$ と, $V$ 上の**二項関係** $E$ との組である.
  - $G = (V, E)$ と表す.
  - $V$ を頂点集合と呼び, $V$ の要素を頂点と呼ぶ.
  - $E$ を辺集合と呼び, $E$ の要素を辺と呼ぶ.




頂点は円で表現され,
辺は矢印で表現される

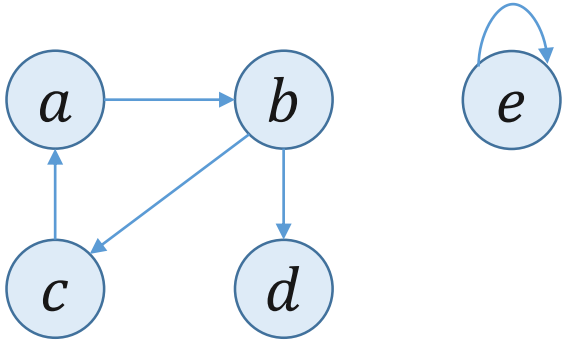
$$G = (V, E)$$
$$V = \{a, b, c, d, e\}$$
$$E = \{(a, b), (c, a), (b, c), (b, d), (e, e)\}$$

# (準備 2/3) パス

- 有向グラフ $G$ における**パス** $p$ とは, 以下の性質を満たす頂点の列 $(v_0, v_1, \ldots, v_k)$ である.
  - (性質 2) $i = 1, 2, \ldots k$ に対して, $(v_{i-1}, v_i) \in E$

- $v_0$ を**始点**, $v_k$ を**終点**, $k$ を**長さ**と呼ぶ.

右の有向グラフにおいて...

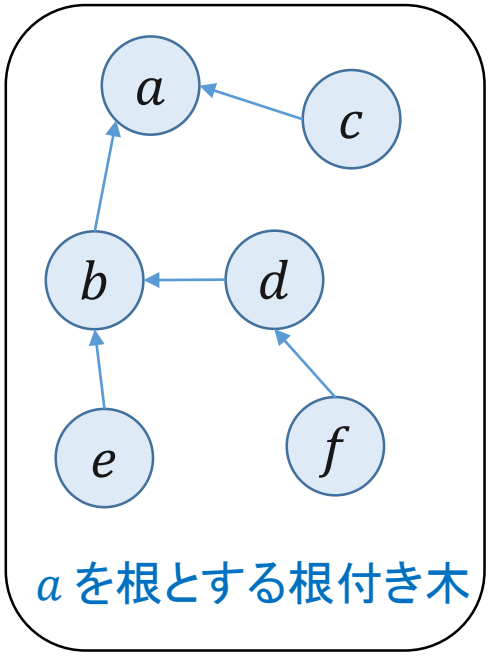


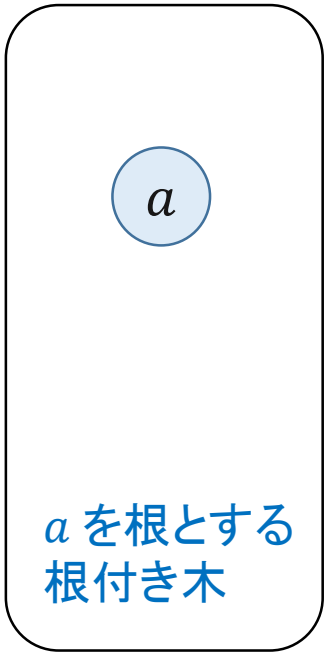

$(a, b, c)$ はパス
$(a, b, c, a, b, d)$ はパス
$(e, e)$ はパス
$(e)$ はパス
$(e, a)$ はパスではない

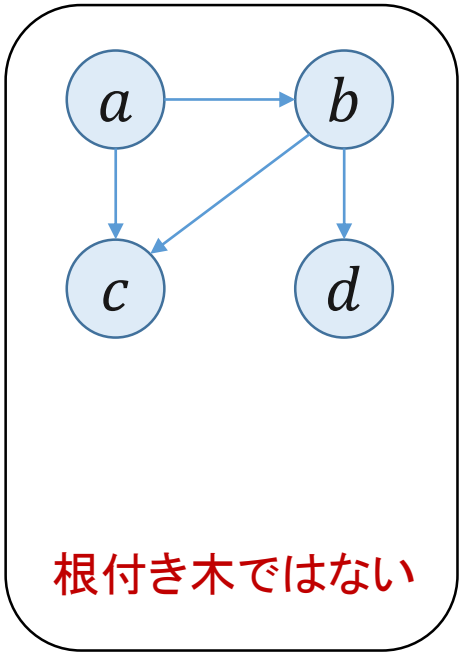
$$G = (V, E)$$
$$V = \{a, b, c, d, e\}$$
$$E = \{(a, b), (c, a), (b, c), (b, d), (e, e)\}$$

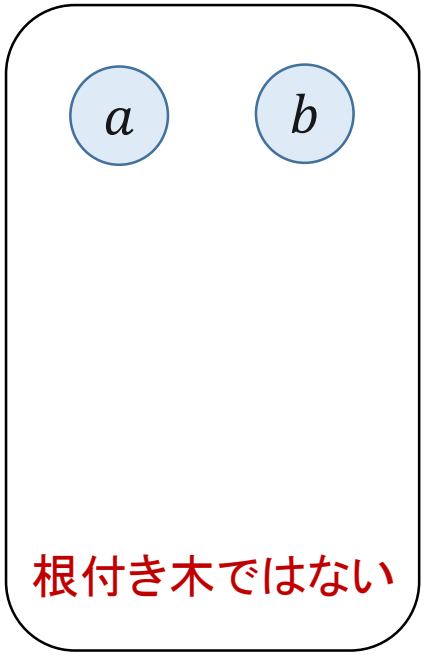
# (準備 3/3) 根付き木

- 有向グラフ $G$ と $G$ の頂点 $r$ が以下の性質を満たすとき, $G$ は $r$ を根とする根付き木であるという.
  - (性質) $x \in V$ に対して, 始点が$x$, 終点が $r$ となるパスがちょうど一つ存在する.



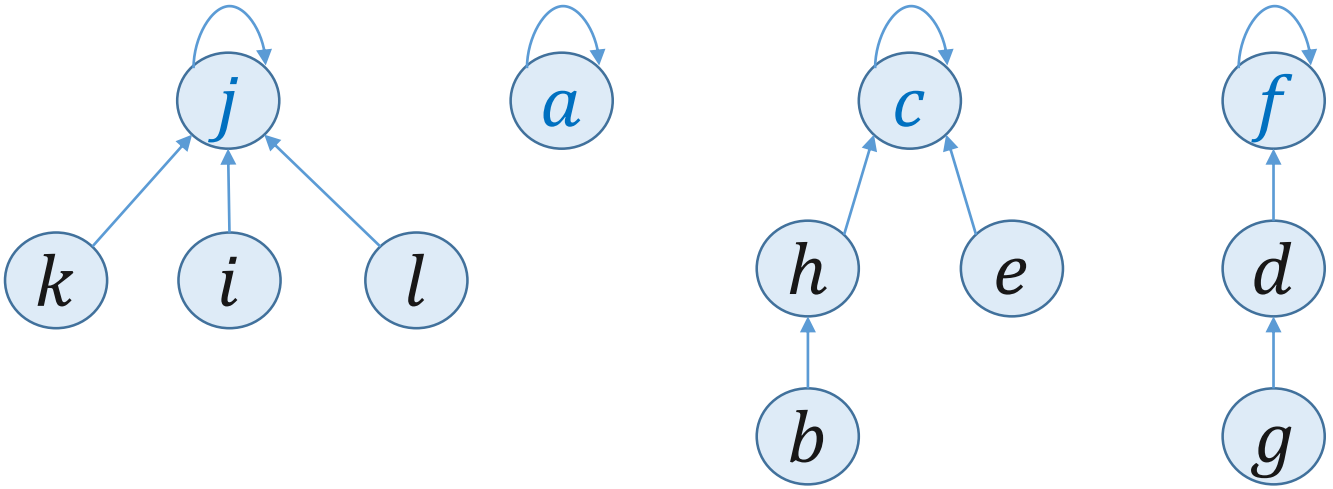



# 素集合森



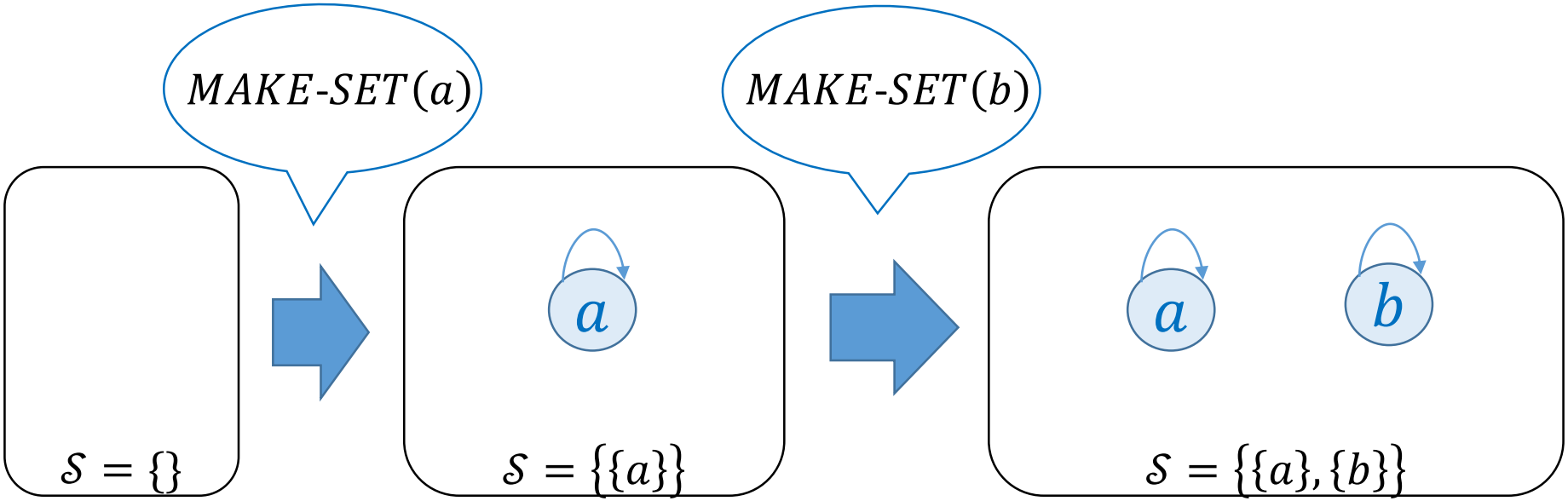
- **素集合森** では, 各集合を根付き木 $\mathcal{G} = \{G_1, G_2, \dots\}$ で表現する.
- 代表元は各根付き木における根とする.
- 各頂点は, 自身の親へのポインタ ($par$) を持つ.
- 根の $par$ は自分自身とする.




$$S = \{\{j, k, i, l\}, \{a\}, \{b, c, e, h\}, \{d, f, g\}\}$$

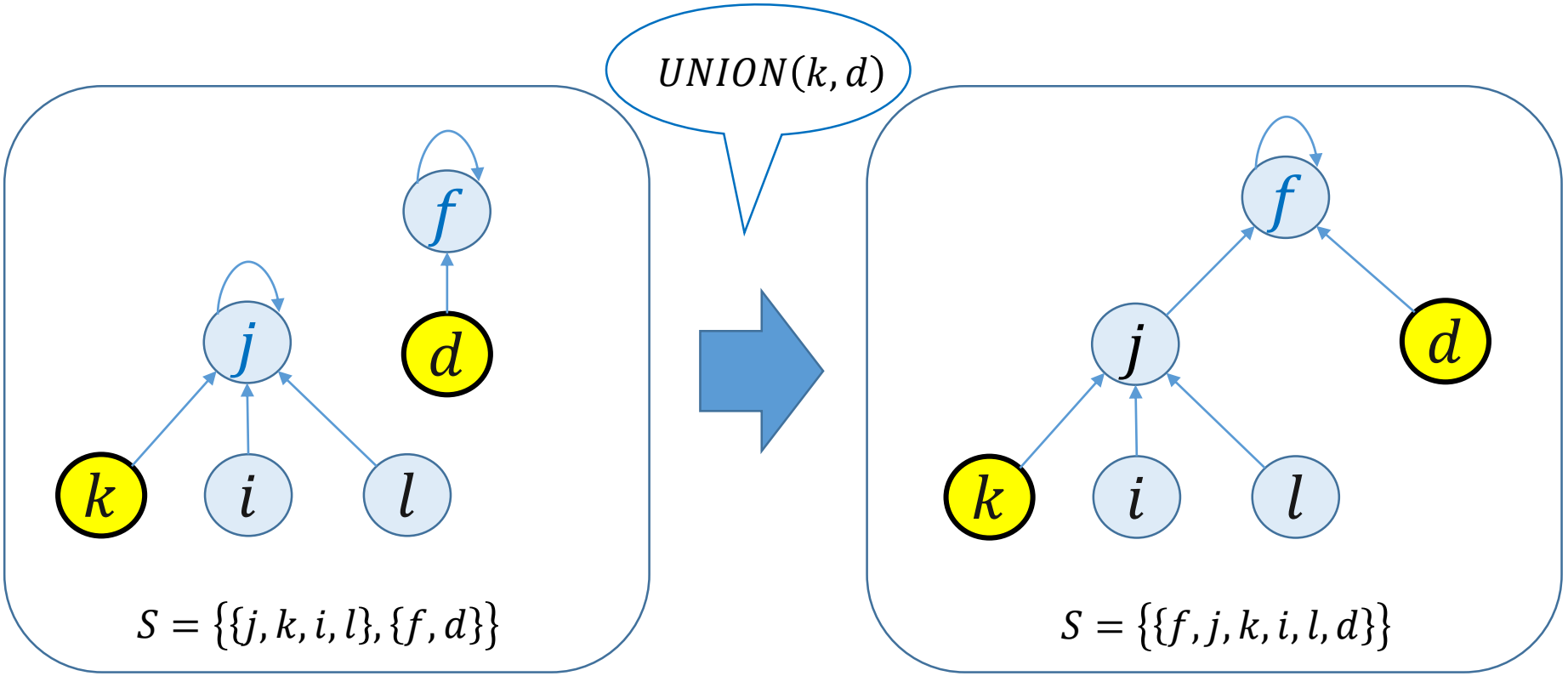
# $MAKE\text{-}SET(x)$

- 新たな根付き木 $G = (\{x\}, \emptyset)$ を作る.
  1. 頂点 $x$ を新たに作る.
  2. $x$ の $par$ を $x$ に**初期化**.



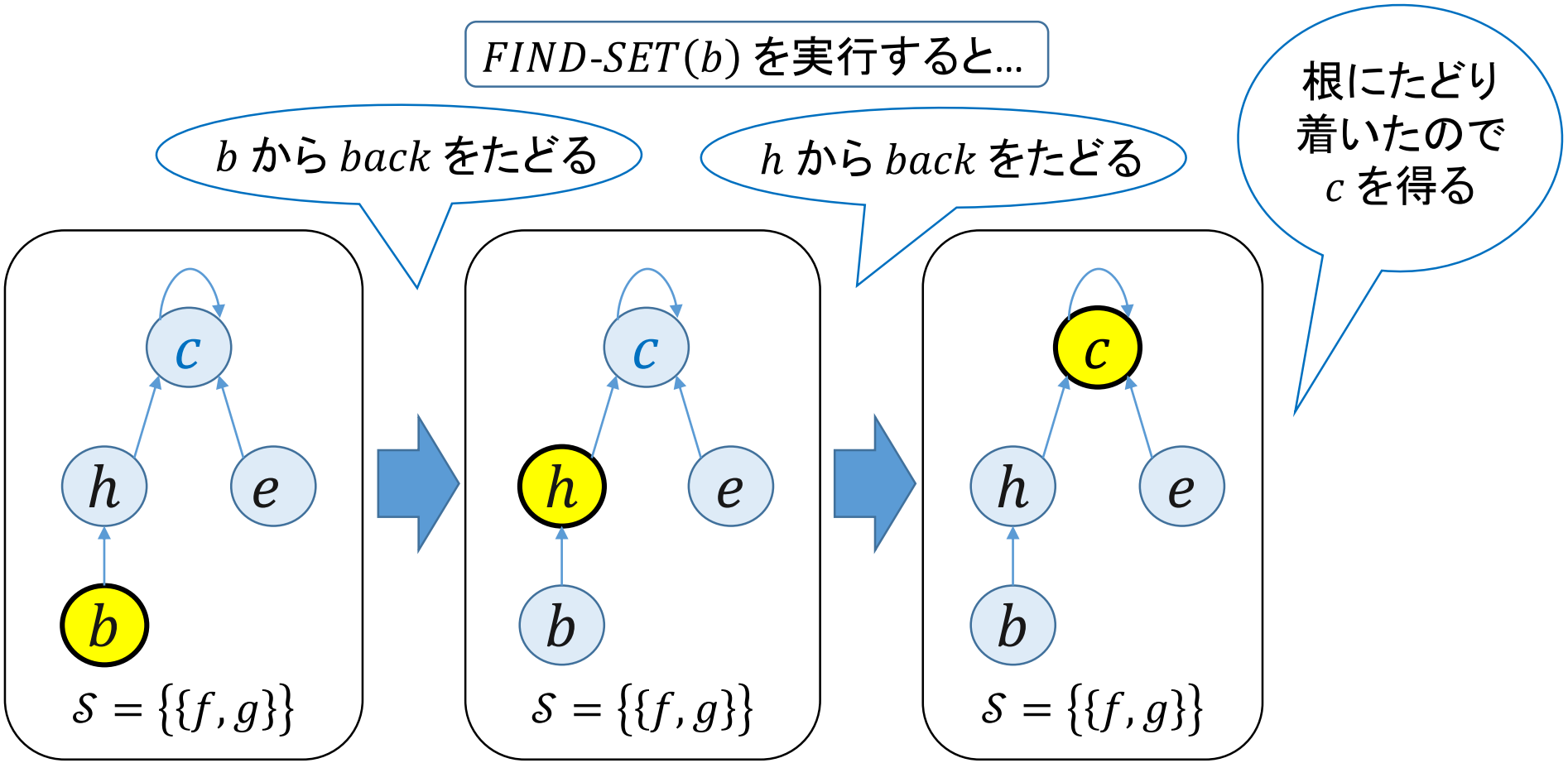
# $UNION(x, y)$

- 二つの根付き木のポインタを張り替えて, 統合する.
  - $x, y$ を含む根付き木をそれぞれ $G_x, G_y$ とする.
  1. $G_x$ の根の $par$ を $G_y$ の根に**更新**.
     - 根を取得するのに $FIND\text{-}SET(x)$ $FIND\text{-}SET(y)$ が必要.



# $FIND\text{-}SET(x)$

- $x$ からポインタをたどり, 根(**代表元**)を得る.
  - $x$ を含む根付き木を $G_x$ とする.
  1. $x$ から根にたどり着くまで $par$ をたどり, $G_x$ の根を得る.



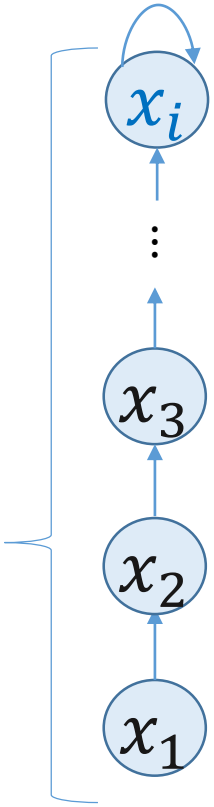
# 計算効率 (工夫なし)



- 右のような $2n-1$ 回の操作列を考える.
- $n$ 回の $MAKE\text{-}SET(x)$ を行うのに $\Theta(n)$.
- $n-1$ 回の $UNION(x,y)$ を行うのに $\Theta(n^2)$.

よって, 操作 1 回のならし計算量は $\Theta(n)$

$i+1$ 回目の操作では,
$par$ ポインタを $x_1$ から $x_i$ まで
$i$ 回たどる必要がある




| 操作 | 更新が起こるノードの数 |
|---|---|
| $MAKE\text{-}SET(x_1)$ | 1 |
| ⋮ | ⋮ |
| $MAKE\text{-}SET(x_n)$ | 1 |
| $UNION(x_1,x_2)$ | 1 |
| $UNION(x_1,x_3)$ | 2 |
| $UNION(x_1,x_4)$ | 3 |
| ⋮ | ⋮ |
| $UNION(x_1,x_n)$ | $n-1$ |

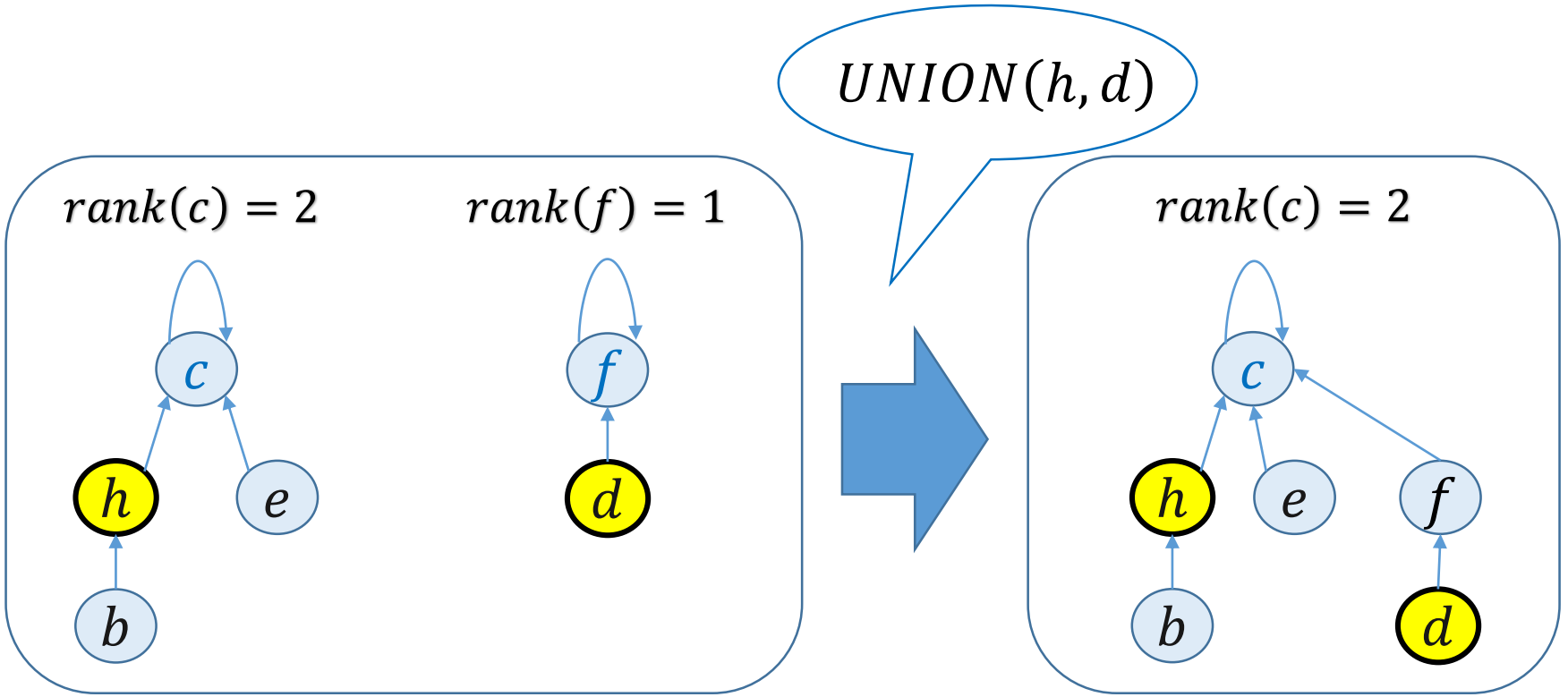
# 計算量を改善する工夫



- 素集合森における, 計算量改善の工夫は二つ.
  1. union by rank
  2. path compression

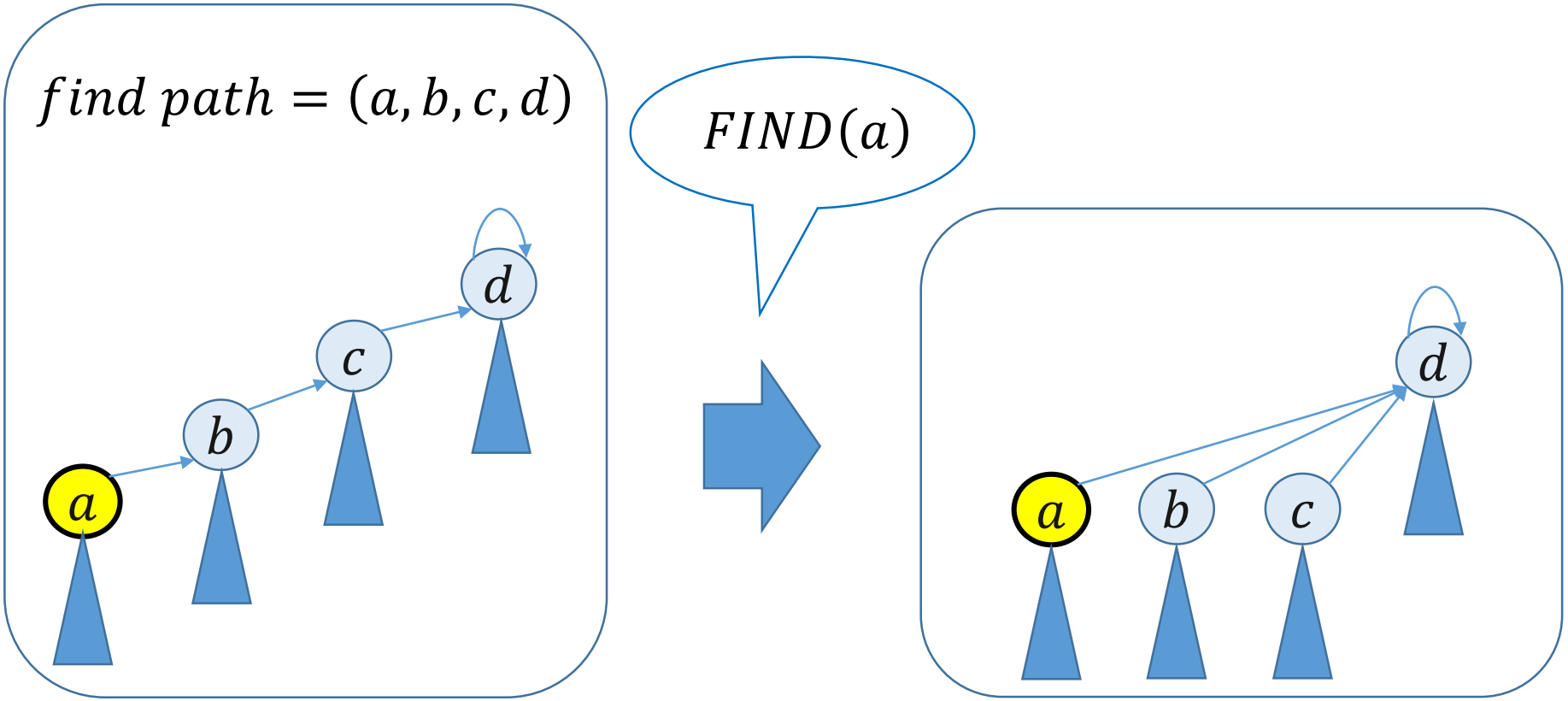
# 工夫1 union by rank

- 各頂点に $rank$ という情報を持たせる.
  - $rank(x)$ := 終点 が $x$ であるようなパスの, 長さの最大値.
- $UNION(x, y)$ を行う際, $rank$ が小さい方の根を, 大きい方の根に繋ぎ変えるような工夫.



# 工夫2 path compression

- $FIND\text{-}SET(x)$ を行なう際, 始点が $x$, 終点が $G_x$ の根であるようなパスを $find\ path$ と呼ぶ.
- $find\ path$ 上のすべての頂点の $par$ を $G_x$ の根にする.
  - ただし, $rank(x)$ は**変えない**ものとする.



# 各操作の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

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

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

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

# $MAKE\text{-}SET(x)$ の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

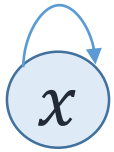
```
MAKE-SET(x)
1  x.p ← x
2  x.rank ← 0
```

# $MAKE\text{-}SET(x)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.p$ を $x$ で初期化

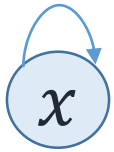


# $MAKE\text{-}SET(x)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank$ を 0 で初期化



# $FIND\text{-}SET(x)$ の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

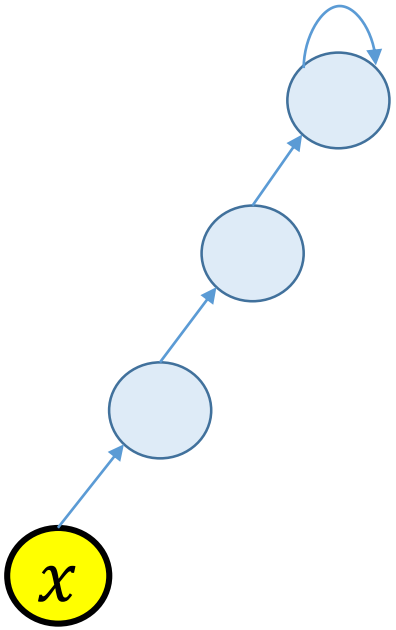
```
FIND-SET(x)
1  if x.rank ≠ x.p
2      x.p ← FIND-SET(x.p)
3  return x
```

# $FIND\text{-}SET(x)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x$ が根ではないなら



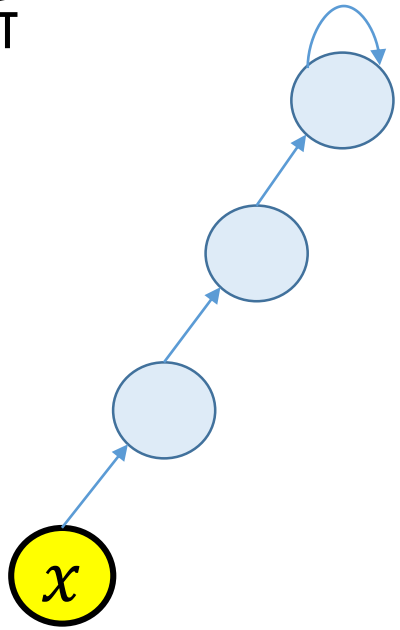
# $FIND\text{-}SET(x)$ の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x$ が根ではないなら
再帰手続きを呼び出して
$x.p$ を根に更新



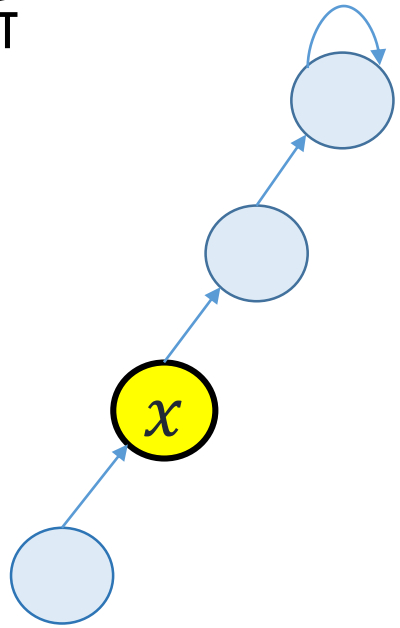
# $FIND\text{-}SET(x)$ の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x$ が根ではないなら
再帰手続きを呼び出して
$x.p$ を根に更新



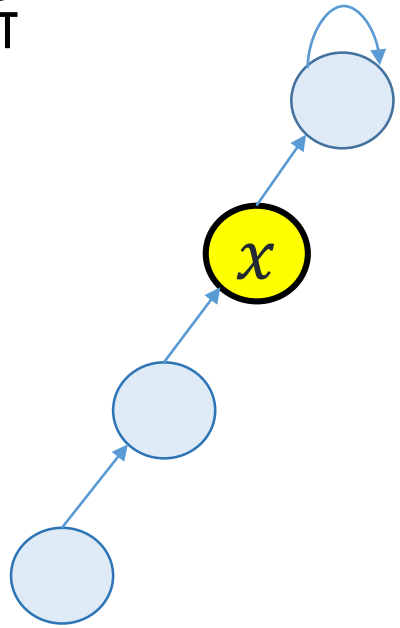
# $FIND\text{-}SET(x)$ の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x$ が根ではないなら
再帰手続きを呼び出して
$x.p$ を根に更新

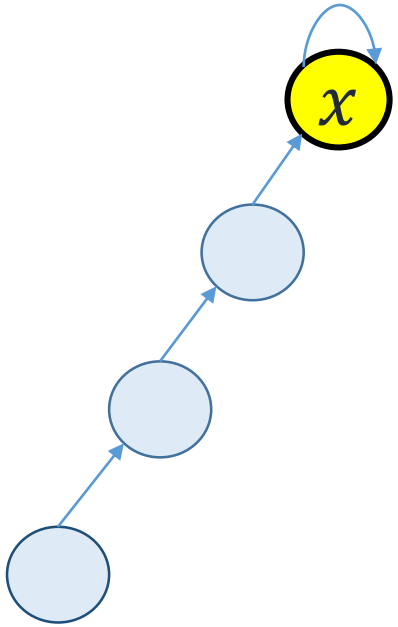


# $FIND\text{-}SET(x)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x$ が根なら
$x$ 自身を返す



# $UNION(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

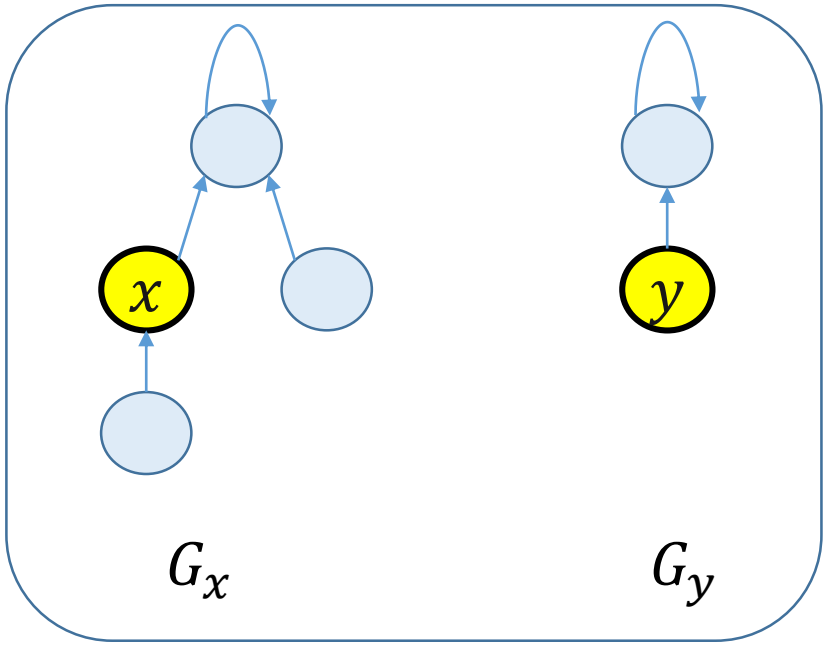
```
UNION(x, y)
1   LINK(FIND-SET(x), FIND-SET(y))
```

# $UNION(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$FIND(x), FIND(\mathrm{y})$ を呼び出して

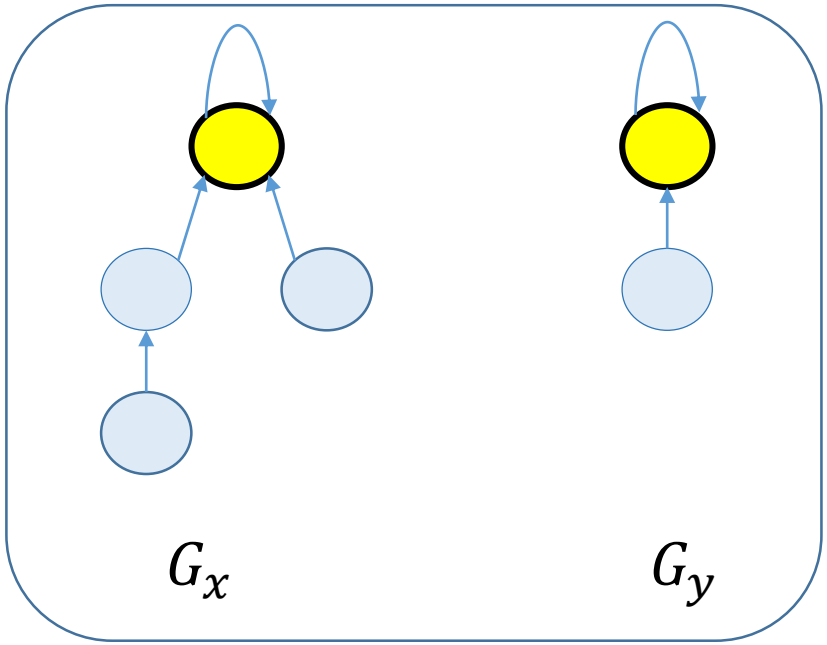


# $UNION(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$G_x, G_y$ の根を
$LINK$ に渡す.



# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$LINK(x, y)$ は $UNION(x, y)$ のサブルーチンで
二つの根を入力として受け取る

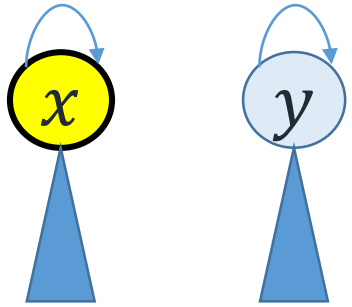
# $LINK(x, y)$ の擬似コード



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank > y.rank$ の時

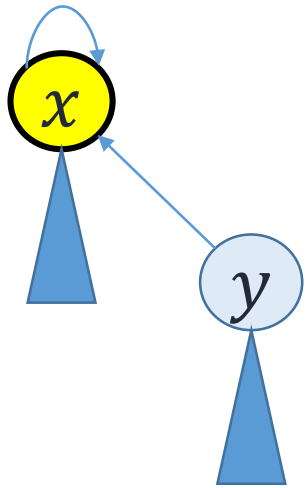


# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank > y.rank$ の時
$y.p$ を $x$ に更新

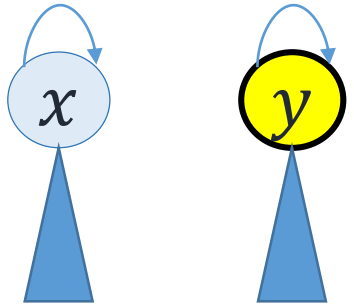


# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank \leq y.rank$ の時

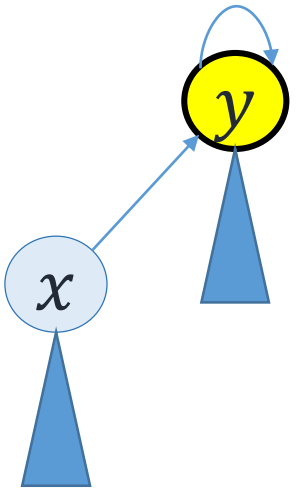


# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank \leq y.rank$ の時
$x.p$ を $y$ に更新

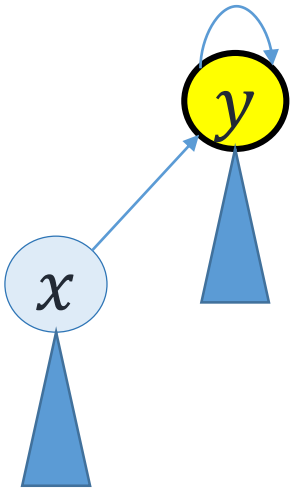


# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank \leq y.rank$ の時
$x.p$ を $y$ に更新
特に $x.rank = y.rank$ の時

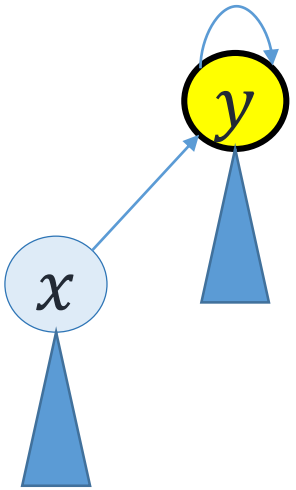


# $LINK(x, y)$ の擬似コード

- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

$x.rank \leq y.rank$ の時
$x.p$ を $y$ に更新
特に $x.rank = y.rank$ の時
$y.rank$ を 1 増やす



# 各操作の擬似コード(再掲)



- $x.rank$: 頂点 $x$ のランク
- $x.p$: 頂点 $x$ の $par$

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

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

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

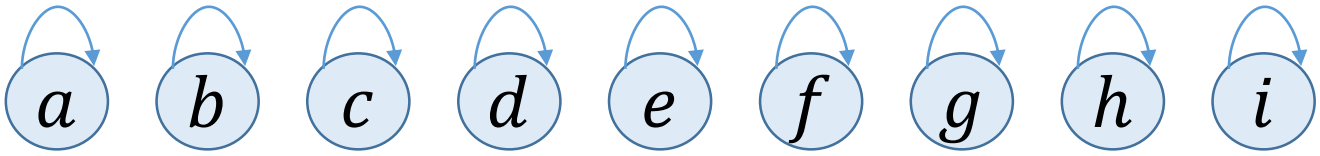
```
UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))
```

# 各操作の実行例



$MAKE\text{-}SET(a) \sim MAKE\text{-}SET(i)$ までは実行済み

| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



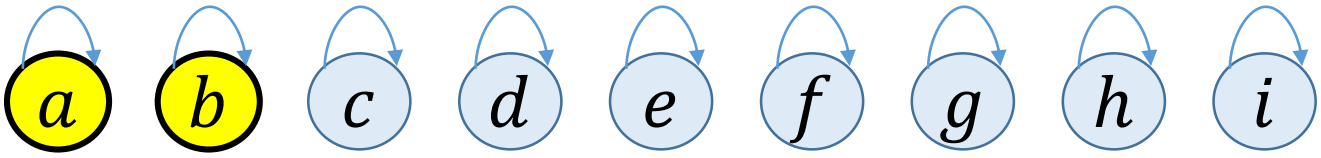

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(a, b)$ を実行

| 操作列 |
| --- |
| $\boldsymbol{\underline{LINK(a, b)}}$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $LINK(b, d)$ |
| $FIND(a)$ |
| $LINK(f, g)$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $LINK(i, d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



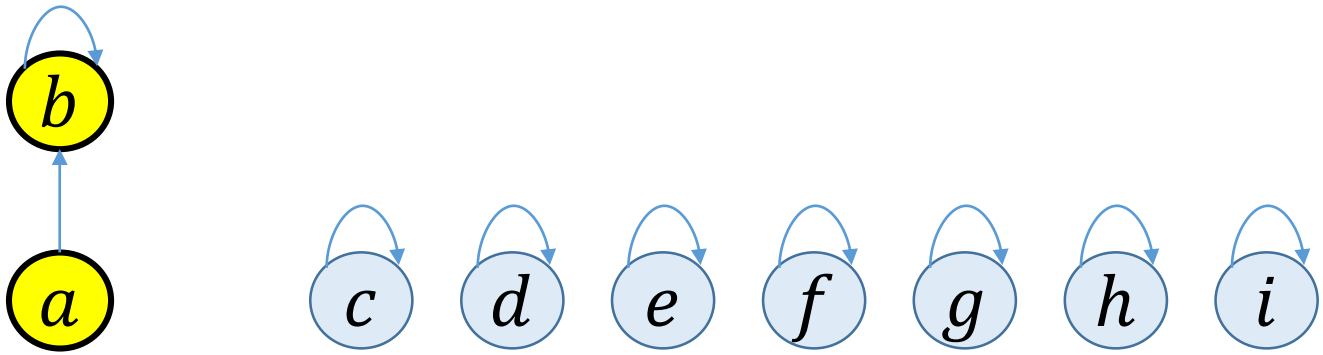

| 頂点 | $\underline{a}$ | $\underline{b}$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(a,b)$ を実行
$b.rank$ が 1 に**増加**

| 操作列 |
|---|
| ***LINK*(*a*, *b*)** |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



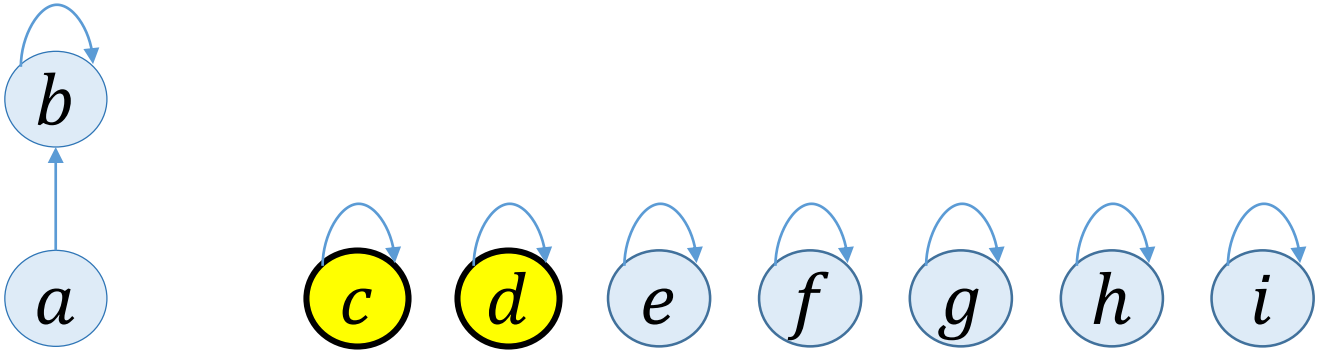

| 頂点 | *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* |
|---|---|---|---|---|---|---|---|---|---|
| *rank* | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(c,d)$ を実行

| 操作列 |
|---|
| $LINK(a,b)$ |
| $\underline{\boldsymbol{LINK(c,d)}}$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



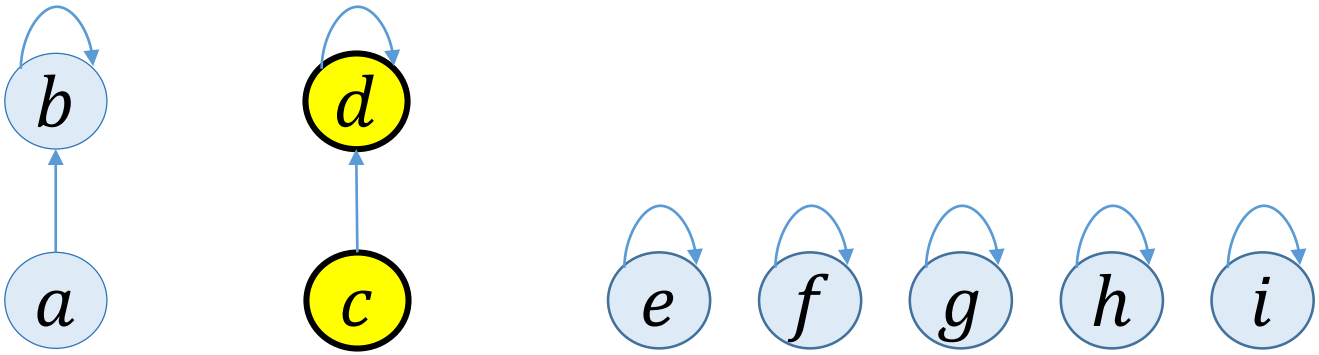

| 頂点 | $a$ | $b$ | $\underline{c}$ | $\underline{d}$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(c,d)$ を実行
$d.rank$ が 1 に**増加**

| 操作列 |
|---|
| $LINK(a,b)$ |
| $\mathbf{LINK(c,d)}$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



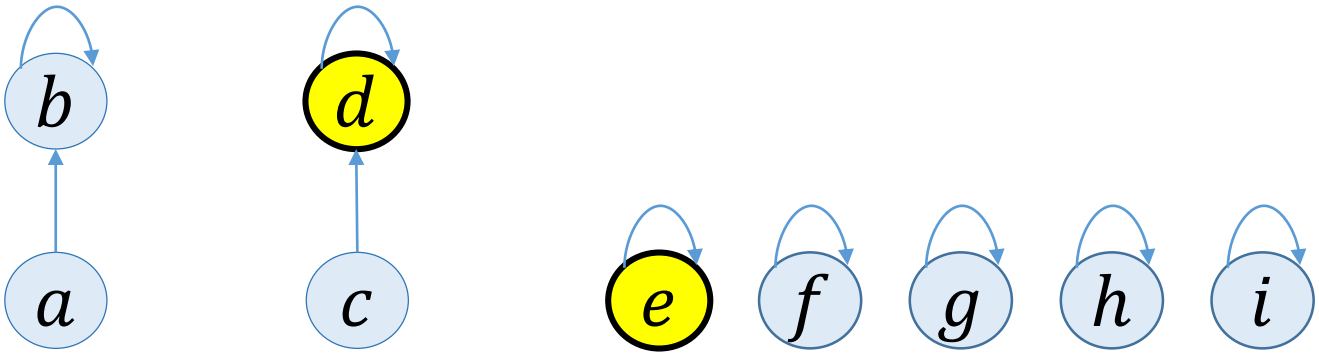

| 頂点 | $a$ | $b$ | $\underline{c}$ | $\underline{d}$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(e,d)$ を実行

| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $\boldsymbol{LINK(e,d)}$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



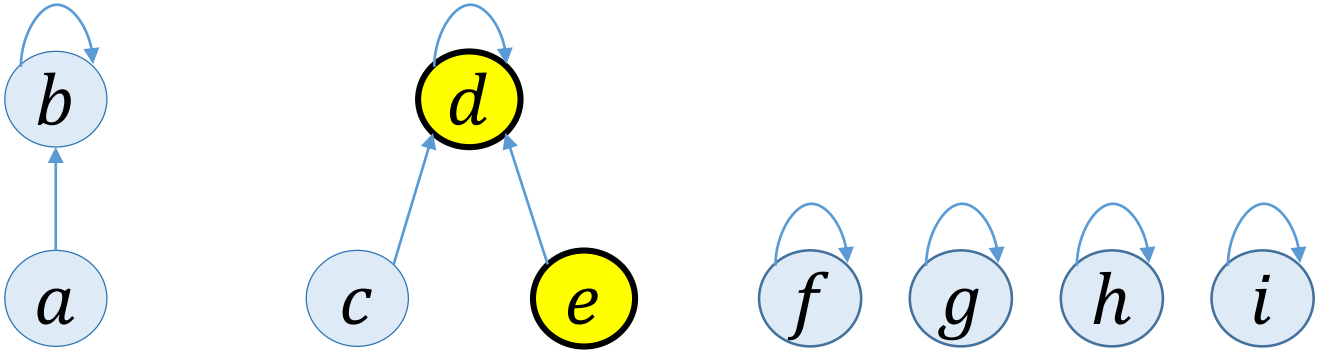

| 頂点 | $a$ | $b$ | $c$ | $\boldsymbol{d}$ | $\boldsymbol{e}$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(e,d)$ を実行
$rank$ は変化しない

| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $\boldsymbol{LINK(e,d)}$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

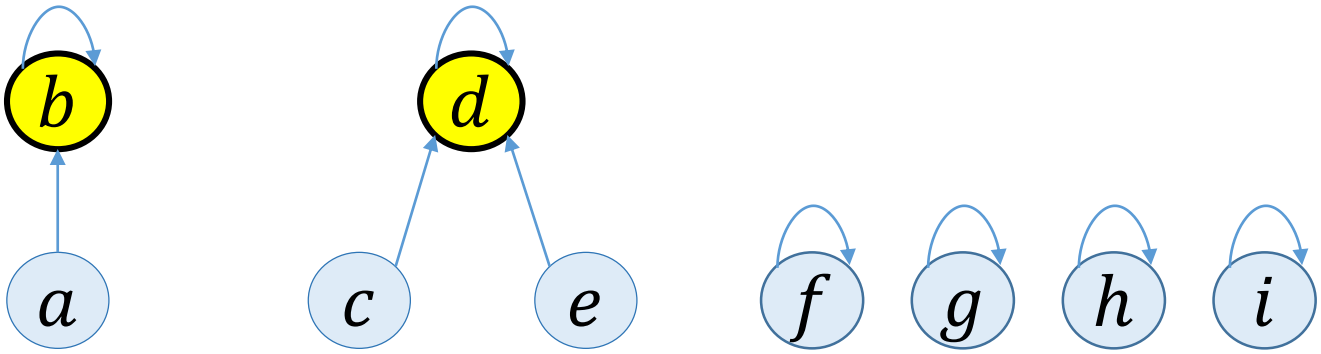



| 頂点 | $a$ | $b$ | $c$ | $\boldsymbol{d}$ | $\boldsymbol{e}$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(b, d)$ を実行



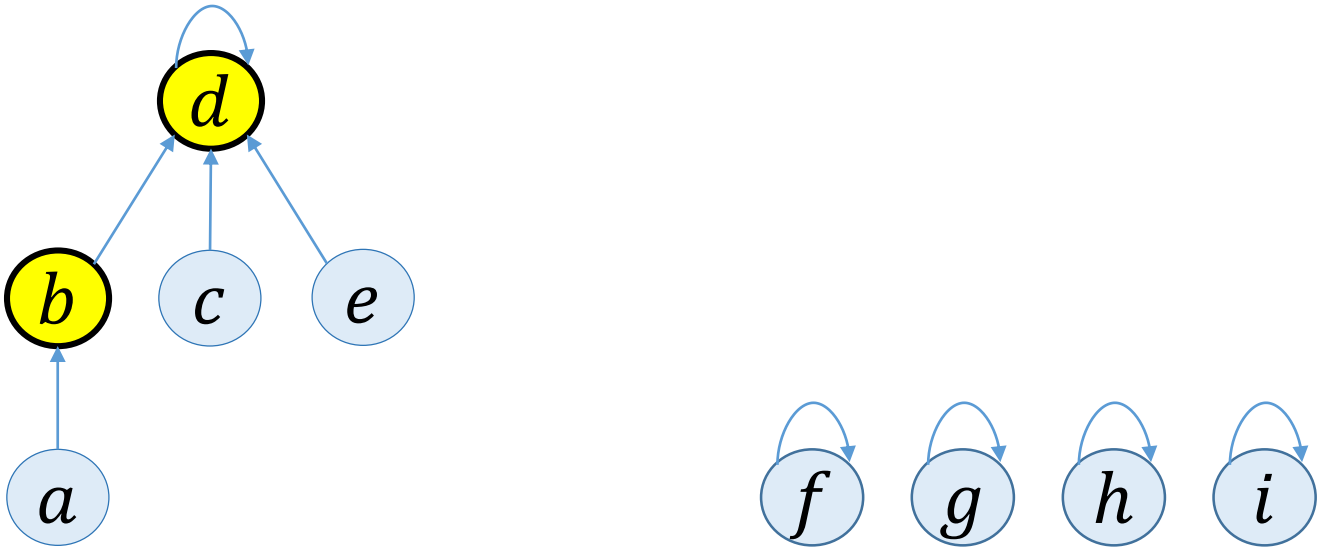

| 操作列 |
| --- |
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $\underline{\boldsymbol{LINK(b, d)}}$ |
| $FIND(a)$ |
| $LINK(f, g)$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $LINK(i, d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $\underline{b}$ | $c$ | $\underline{d}$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(b, d)$ を実行
$d.rank$ が 2 に**増加**



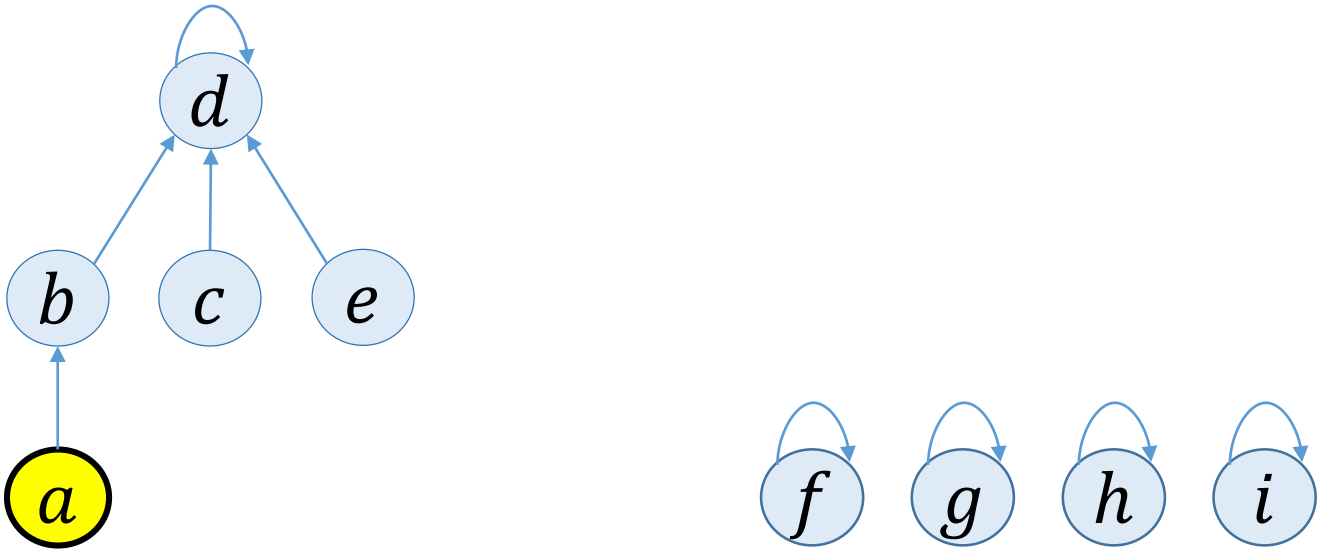

| 操作列 |
|---|
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $\boldsymbol{LINK(b, d)}$ |
| $FIND(a)$ |
| $LINK(f, g)$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $LINK(i, d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $\boldsymbol{b}$ | $c$ | $\boldsymbol{d}$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$FIND(a)$ を実行



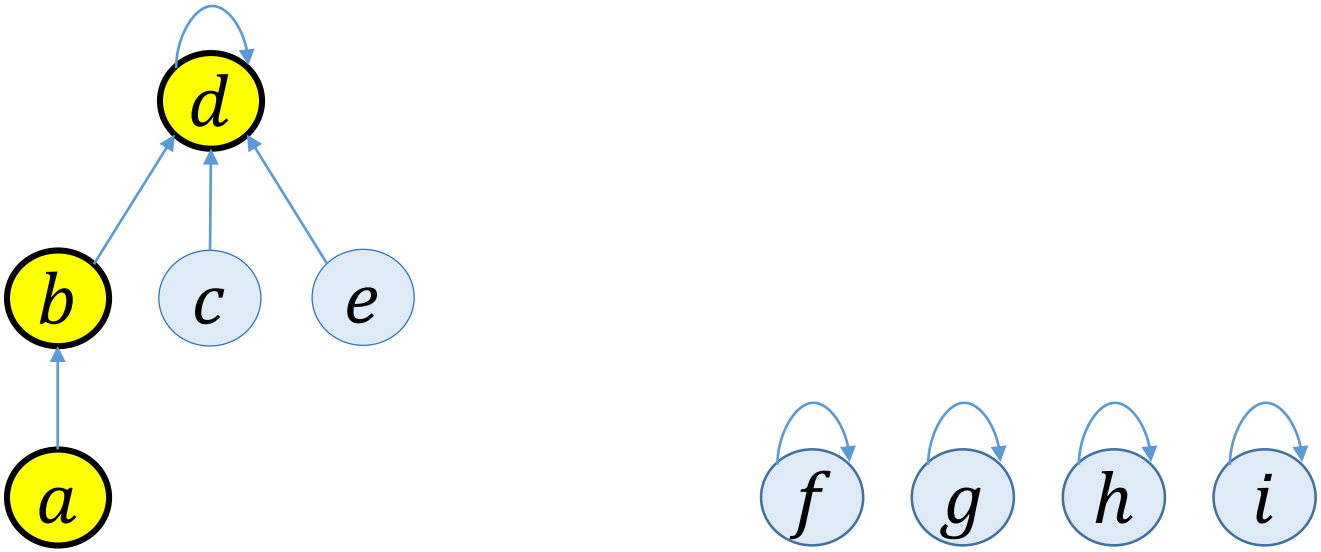

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $\boldsymbol{FIND(a)}$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$FIND(a)$ を実行
$find\ path = (a, b, d)$




| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $\boldsymbol{FIND(a)}$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

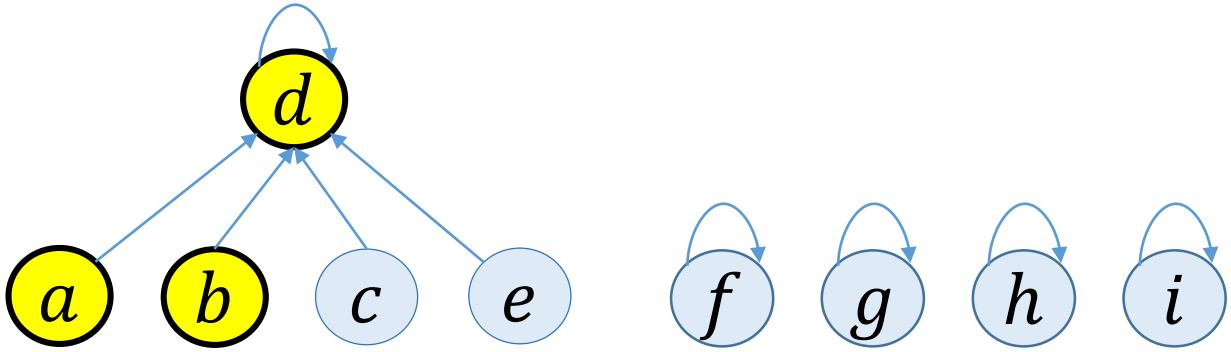
| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$FIND(a)$ を実行
$find\ path = (a, b, d)$
各頂点の$par$ を $d$ に更新

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $\boldsymbol{FIND(a)}$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

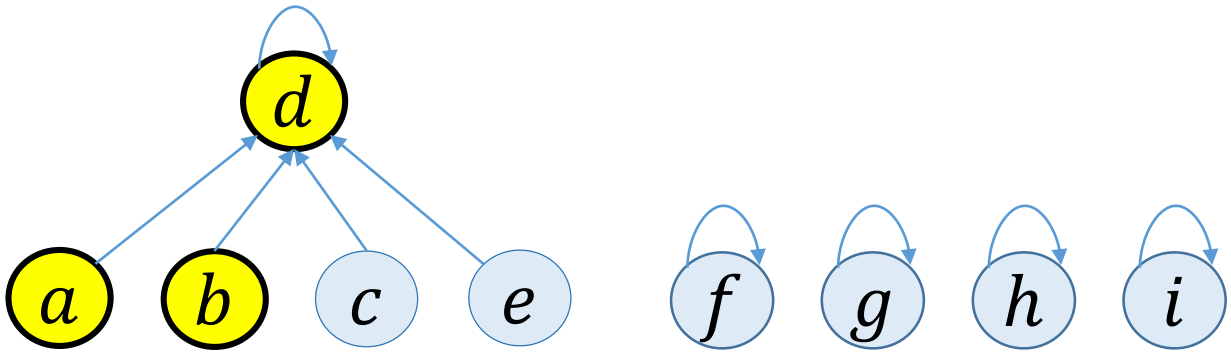



| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

# union by rank and path compression

$FIND(a)$ を実行
$find\ path = (a, b, d)$
各頂点の$par$ を $d$ に更新

**どの頂点も $rank$ は変化しない**




| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

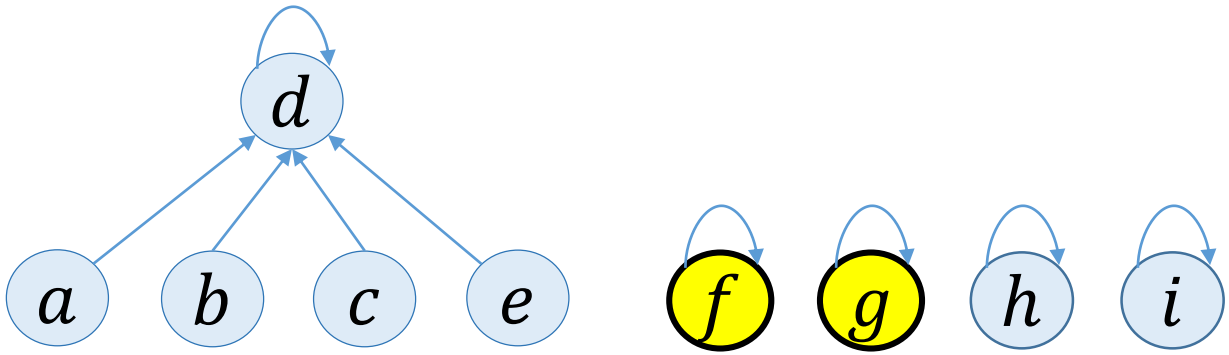
| 操作列 |
|---|
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $LINK(b, d)$ |
| $\boldsymbol{FIND(a)}$ |
| $LINK(f, g)$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $LINK(i, d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

# 各操作の実行例



$LINK(f, g)$ を実行

| 操作列 |
|---|
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $LINK(b, d)$ |
| $FIND(a)$ |
| $\boldsymbol{LINK(f, g)}$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $LINK(i, d)$ |
| $FIND(g)$ |
| $FIND(h)$ |



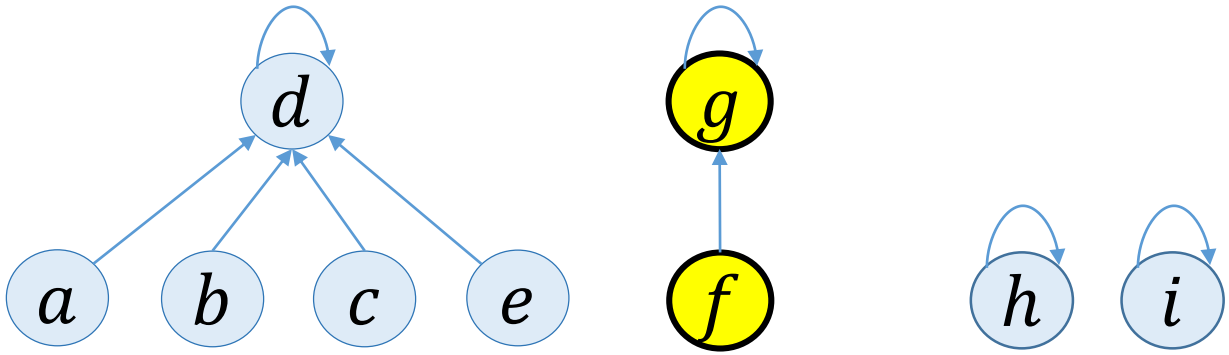

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $\boldsymbol{f}$ | $\boldsymbol{g}$ | $h$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |

# 各操作の実行例



$LINK(f, g)$ を実行
$g.rank$ が 1 に**増加**

| 操作列 |
| --- |
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $LINK(b, d)$ |
| $FIND(a)$ |
| ***LINK*(*f*, *g*)** |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $LINK(i, d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

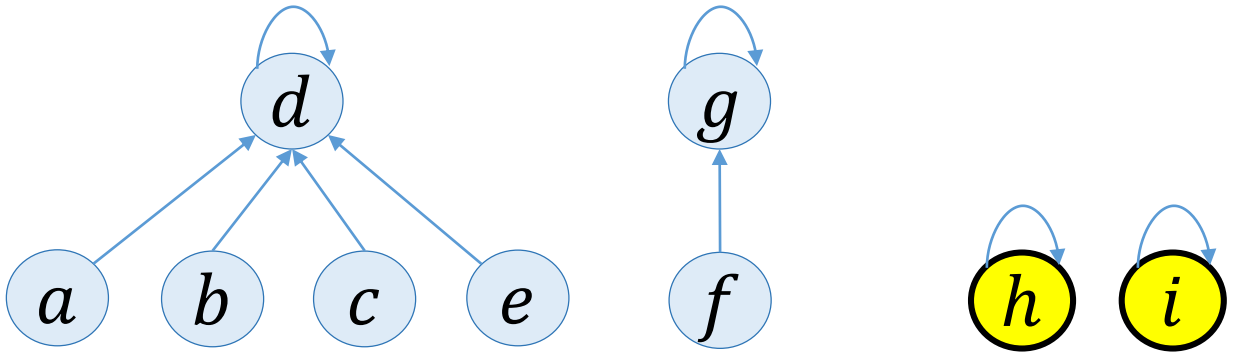



| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |

# 各操作の実行例



$LINK(h,i)$ を実行



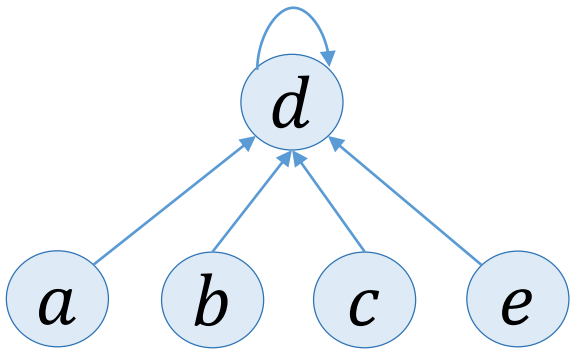

| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $\boldsymbol{LINK(h,i)}$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

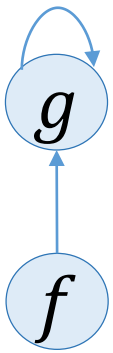
| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $\underline{h}$ | $\underline{i}$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 0 |

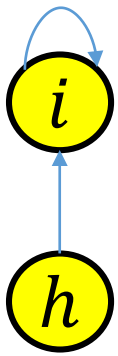
# 各操作の実行例



$LINK(h,i)$ を実行
$h.rank$ が 1 に**増加**



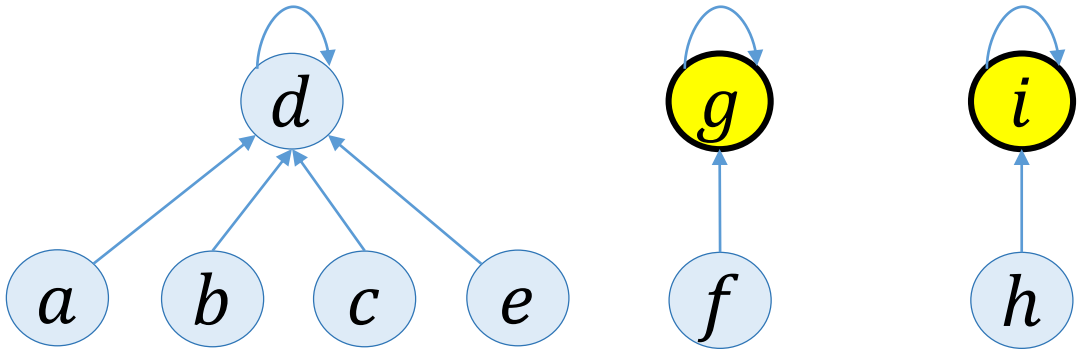

| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $\boldsymbol{LINK(h,i)}$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $\boldsymbol{h}$ | $\boldsymbol{i}$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 1 |

# 各操作の実行例



$LINK(g,i)$ を実行

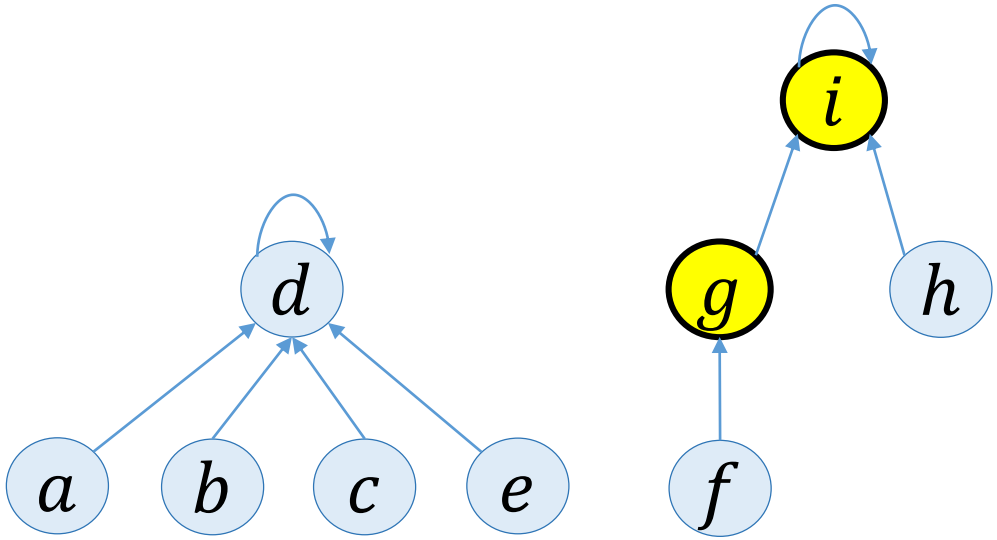



| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| **$\underline{LINK(g,i)}$** |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $\underline{g}$ | $h$ | $\underline{i}$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 1 |

# 各操作の実行例

$LINK(g,i)$ を実行
$i.rank$ が 2 に**増加**



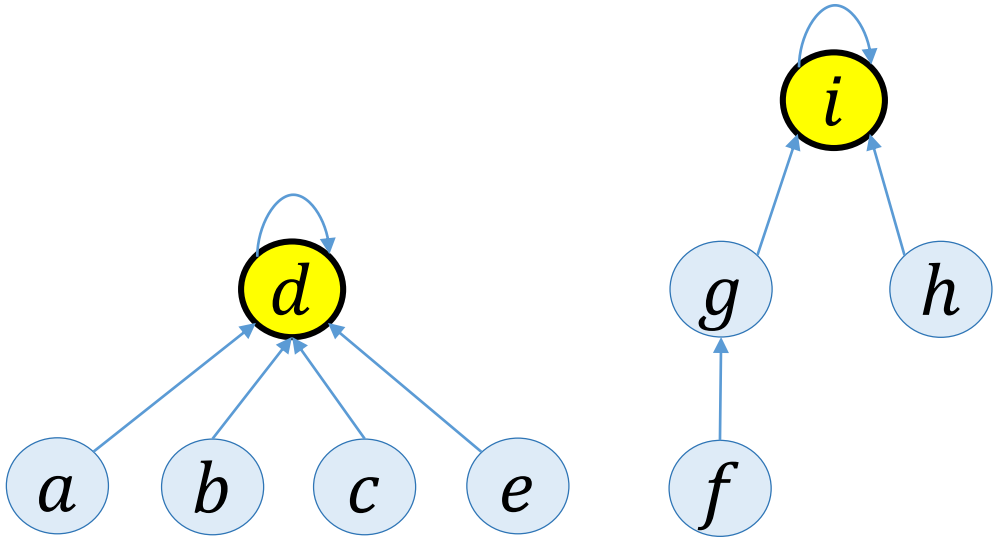

| 操作列 |
|---|
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $\boldsymbol{LINK(g,i)}$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $\boldsymbol{g}$ | $h$ | $\boldsymbol{i}$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$LINK(i, d)$ を実行



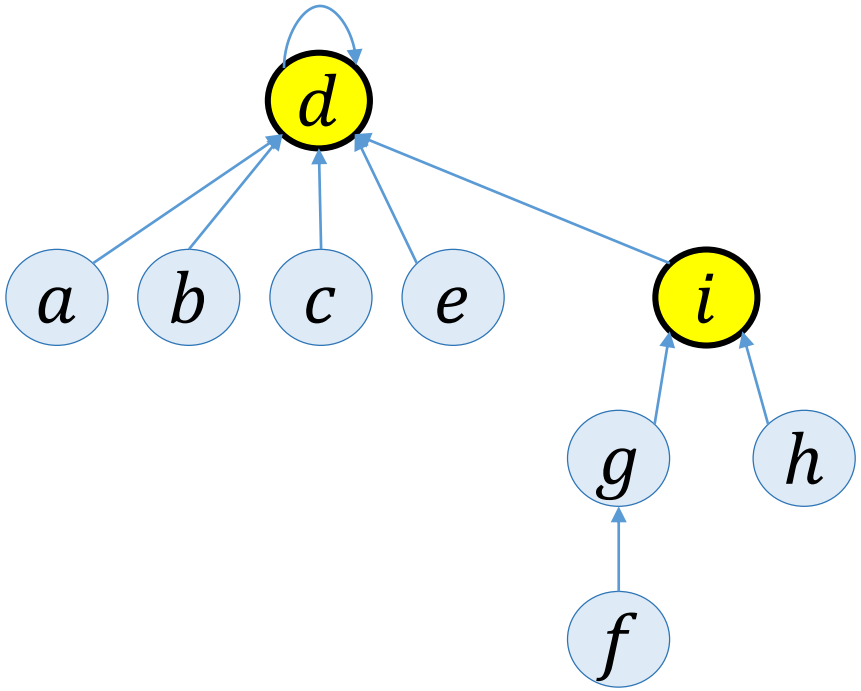

| 操作列 |
| --- |
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $LINK(b, d)$ |
| $FIND(a)$ |
| $LINK(f, g)$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $\boldsymbol{LINK(i, d)}$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $\underline{d}$ | $e$ | $f$ | $g$ | $h$ | $\underline{i}$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$LINK(i, d)$ を実行
$d.rank$ が 2 に**増加**



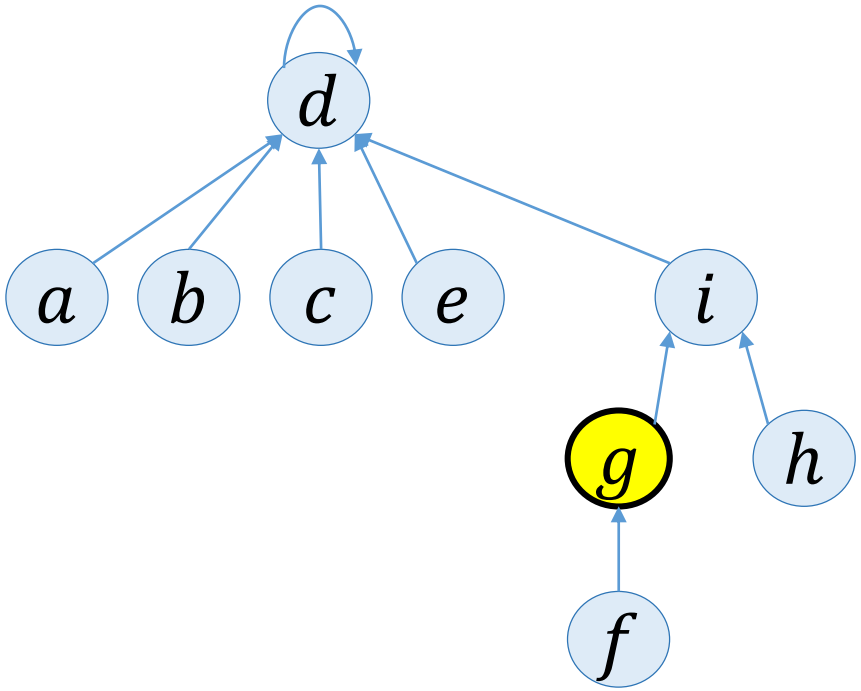

| 操作列 |
|---|
| $LINK(a, b)$ |
| $LINK(c, d)$ |
| $LINK(e, d)$ |
| $LINK(b, d)$ |
| $FIND(a)$ |
| $LINK(f, g)$ |
| $LINK(h, i)$ |
| $LINK(g, i)$ |
| $\boldsymbol{LINK(i, d)}$ |
| $FIND(g)$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $\underline{d}$ | $e$ | $f$ | $g$ | $h$ | $\underline{i}$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$FIND(g)$ を実行



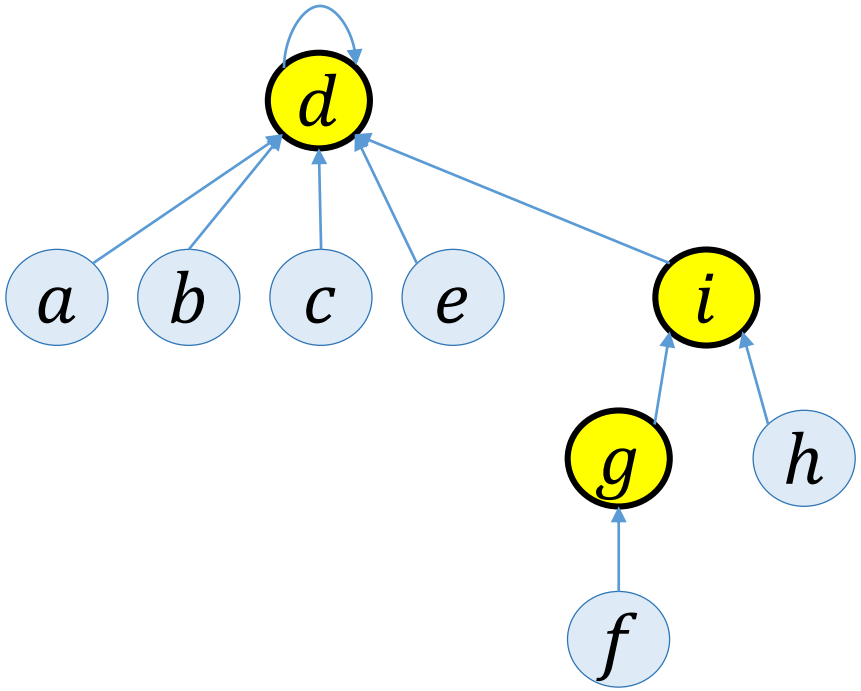

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $\underline{\boldsymbol{FIND(g)}}$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$FIND(g)$ を実行



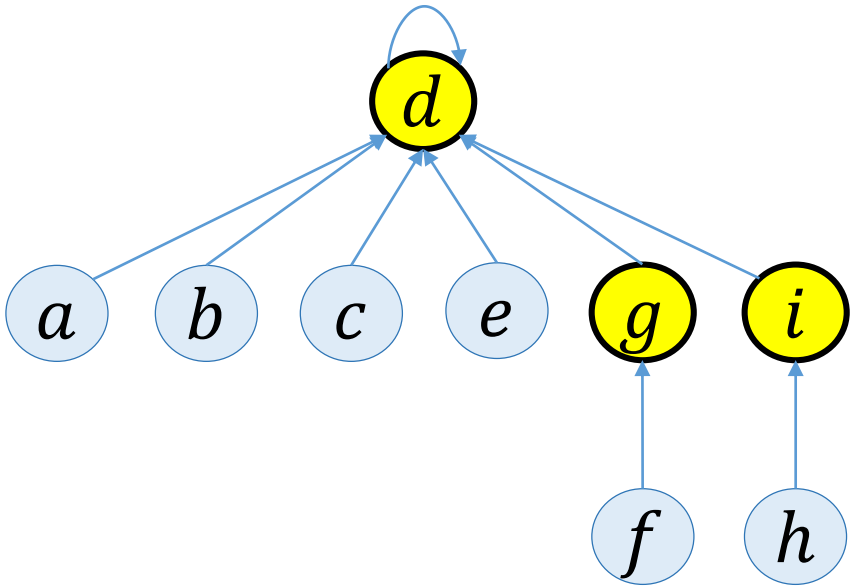

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $\underline{\boldsymbol{FIND(g)}}$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$FIND(g)$ を実行



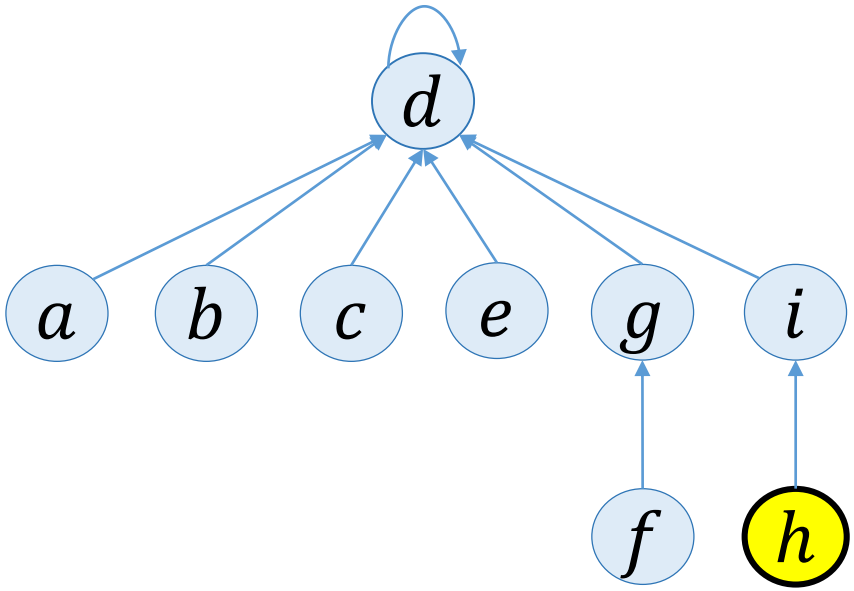

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $\underline{\boldsymbol{FIND(g)}}$ |
| $FIND(h)$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$FIND(h)$ を実行



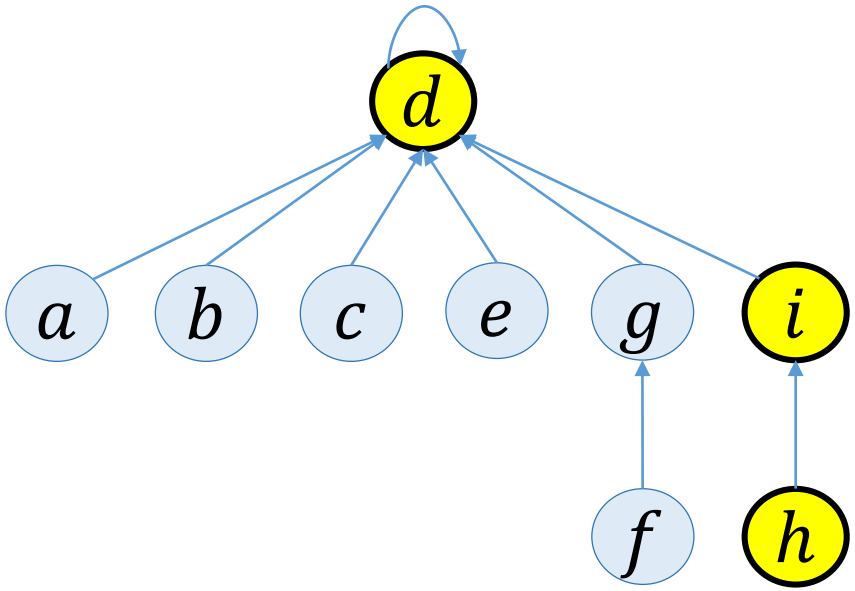

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $\boldsymbol{FIND(h)}$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$FIND(h)$ を実行



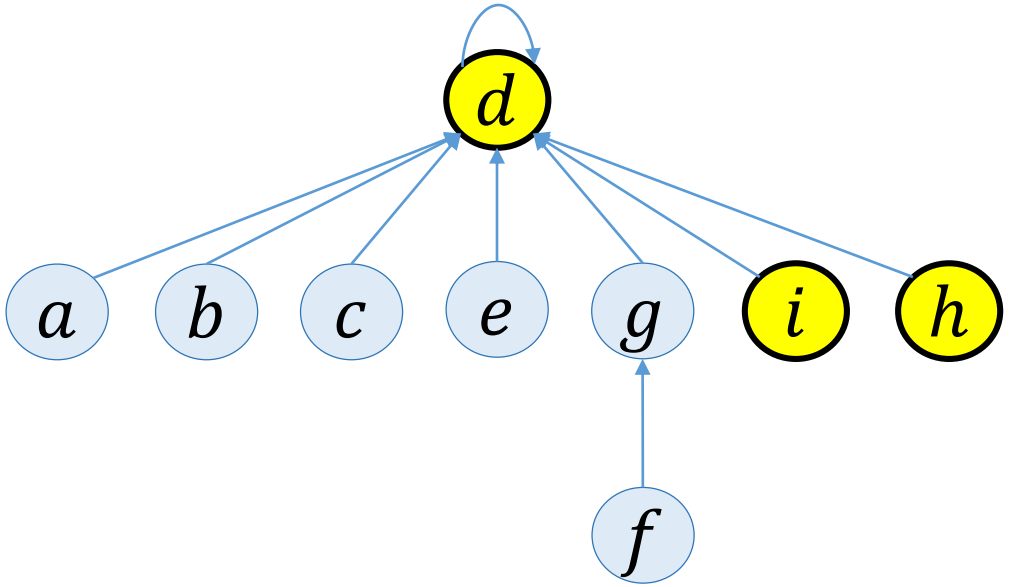

| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $\boldsymbol{FIND(h)}$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 各操作の実行例



$FIND(h)$ を実行




| 操作列 |
| --- |
| $LINK(a,b)$ |
| $LINK(c,d)$ |
| $LINK(e,d)$ |
| $LINK(b,d)$ |
| $FIND(a)$ |
| $LINK(f,g)$ |
| $LINK(h,i)$ |
| $LINK(g,i)$ |
| $LINK(i,d)$ |
| $FIND(g)$ |
| $\underline{\boldsymbol{FIND(h)}}$ |

| 頂点 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $rank$ | 0 | 1 | 0 | 3 | 0 | 0 | 1 | 0 | 2 |

# 計算効率 (工夫あり)



- union by rank のみを施した場合
  操作 1 回あたりのならし計算量は $O(\log n)$.
- path compression のみを施した場合
  操作全体の時間計算量は $O\left(n + f \cdot \left(1 + \log_{2+\frac{f}{n}} n\right)\right)$

- この両方を施した場合
  操作 1 回あたりのならし計算量は $O(\alpha(n))$.

**本スライドで解説**

# 目次

# 今日の目次

- アッカーマン関数について
- $rank$ に関する性質
- ポテンシャル法
- ポテンシャル関数を導入のための準備
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

# 今日の目次

- **アッカーマン関数について**
- $rank$ に関する性質
- ポテンシャル法
- ポテンシャル関数を導入のための準備
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

# アッカーマン関数(1/3)

- アッカーマン関数 $A_k(j)$ を以下のように定義する.

[定義]
整数 $k \geq 0, j \geq 1$ に対して,

$$A_k(j) = \begin{cases} j+1 & \text{if } k = 0 \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1 \end{cases}$$

- 繰り返し関数の記法

$$f^{(i)}(n) = \begin{cases} n & \text{if } i = 0 \\ f\left(f^{(i-1)}(n)\right) & \text{if } i > 0 \end{cases}$$

# アッカーマン関数(2/3)



[補題 1-1]

整数 $j \geq 1$ に対して, $A_1\ (j) = 2j + 1$.

# アッカーマン関数(2/3)

[補題 1-1]

整数 $j \geq 1$ に対して, $A_1\ (j) = 2j + 1$.

[証明]

まず, $A_0^{(i)}(j) = j + i$ を**数学的帰納法**で示す.

1. $A_0^{(0)}(j) = j = j + 0$ より正しい.
2. $A_0^{(i-1)}(j) = j + (i - 1)$ と仮定.

$$A_0^{(i)}(j) = A_0\left(A_0^{(i-1)}(j)\right) = \left(j + (i - 1)\right) + 1 = j + i.$$

よって, $A_1\ (j) = A_0^{(j+1)}(j) = j + (j + 1) = 2j + 1$.

# アッカーマン関数(3/3)



[補題 1-2]

整数 $j \geq 1$ に対して, $A_2\ (j) = 2^{j+1}(j+1) - 1.$

# アッカーマン関数(3/3)

[補題 1-2]

整数 $j \geq 1$ に対して, $A_2\,(j) = 2^{j+1}(j+1) - 1.$

[証明]

まず, $A_1^{(i)}(j) = 2^i(j+1) - 1$ を**数学的帰納法**で示す.

1. $A_1^{(0)}(j) = j = 2^0(j+1) - 1$ より正しい.
2. $A_1^{(i-1)}(j) = 2^{i-1}(j+1) - 1$ と仮定.

$$A_1^{(i)}(j) = A_1\left(A_1^{(i-1)}(j)\right) = A_1\left(2^{i-1}(j+1) - 1\right)$$
$$= 2 \cdot \left(2^{i-1}(j+1) - 1\right) + 1 = 2^i(j+1) - 1.$$

よって, $A_2\,(j) = A_1^{(j+1)}(j) = 2^{j+1}(j+1) - 1.$

# アッカーマン関数の増え方



- $k = 0, 1, 2, 3, 4$ において, $A_k(1)$ がいかに爆発的に増加するかを観察する.

$$A_0(1) = 1 + 1 = 2.$$

$$A_1(1) = 2 \cdot 1 + 1 = 3.$$

$$A_2(1) = 2^{1+1}(1 + 1) - 1 = 7.$$

$$A_3(1) = A_2^{(2)}(1) = A_2\big(A_2(1)\big) = A_2(7) = 2^8 \cdot 8 - 1 = 2047.$$

$$A_4(1) = A_3^{(2)}(1) = A_3\big(A_3(1)\big) = A_3(2047) = A_2^{(2048)}(2047)$$
$$\gg A_2(2047) = 2^{2048} \cdot 2048 - 1 > 2^{2048} = 16^{512}$$

# アッカーマン関数の逆



- アッカーマン関数の逆 $\alpha(n)$ を, 以下のように定める.

[定義]
整数 $n \geq 0$ に対して,
$\alpha(n) = \min\{k : A_k(1) \geq n\}$

$$\alpha(n) = \begin{cases} 0 & \text{for } 0 \leq n \leq 2 \\ 1 & \text{for } n = 3 \\ 2 & \text{for } 4 \leq n \leq 7 \\ 3 & \text{for } 8 \leq n \leq 2047 \\ 4 & \text{for } 2048 \leq n \leq A_4(1) \end{cases}$$

# 今日の目次

- アッカーマン関数について
- **$rank$ に関する性質**
- ポテンシャル法
- ポテンシャル関数導入のための準備
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

# $rank$ に関する性質(1/3)

[補題 2-1]

全てのノード $x$ に対して, $x.rank \leq x.p.rank$ が成立
(等号成立条件は $x = x.p$)

# $rank$ に関する性質(1/3)

[補題 2-1]

全てのノード $x$ に対して, $x.rank \leq x.p.rank$ が成立
(等号成立条件は $x = x.p$)

OH

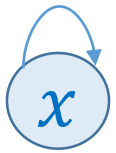
[証明]

**数学的帰納法**で示す.

(1) 基礎ケース

$x = x.p$ であり,

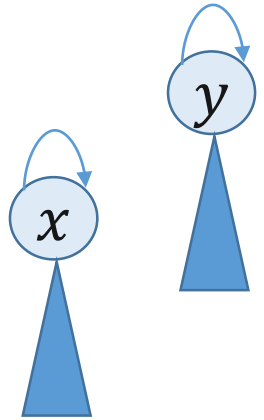
$x.rank = x.p.rank = 0$ なので成立.

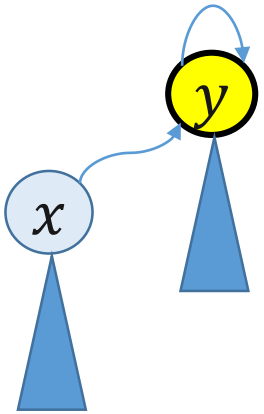


# $rank$ に関する性質(1/3)

[補題 2-1]
全てのノード $x$ に対して, $x.rank \leq x.p.rank$ が成立
(等号成立条件は $x = x.p$)

[証明]
**数学的帰納法**で示す.
(2) 帰納ステップ
$UNION(x, y)$ で変化するのは $y.rank$ と $x.p$ のみ.
操作後, $x.rank < y.rank = x.p.rank$ が成立.

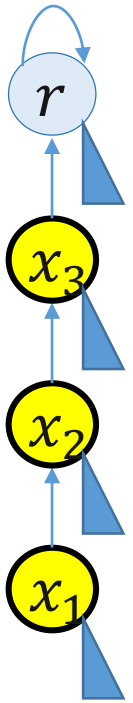


# $rank$ に関する性質(1/3)

[補題 2-1]
全てのノード $x$ に対して, $x.rank \leq x.p.rank$ が成立 (等号成立条件は $x = x.p$)

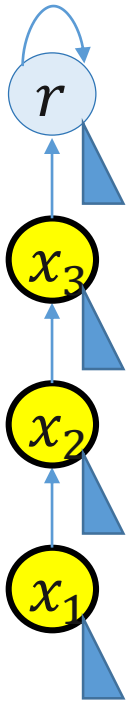
[証明]
**数学的帰納法**で示す.
(2) 帰納ステップ
$FIND(x)$ で変化するのは,
$find\ path = (x_1, \ldots, x_k, r)$ として,
$x_i.p\ (1 \leq i \leq k)$ のみ.
操作後 $x_i.rank < r.rank = x.p.rank$ が成立.



# $rank$ に関する性質(1/3)

[補題 2-1]
全てのノード $x$ に対して, $x.rank \leq x.p.rank$ が成立 (等号成立条件は $x = x.p$)

[証明]
**数学的帰納法**で示す.
(2) 帰納ステップ
$FIND(x)$ で変化するのは,
$find\ path = (x_1, \ldots, x_k, r)$ として,
$x_i.p\ (1 \leq i \leq k)$ のみ.
操作後 $x_i.rank < r.rank = x.p.rank$ が成立.




以上より, 補題 2-1 が示された.

# $rank$ に関する性質(2/3)

[補題 2-2]

$x.rank$ は最初は 0 で, $x = x.p$ の間, 広義単調増加する.

$x \neq x.p$ になった以降, $x.rank$ は変化しない.

# $rank$ に関する性質(2/3)

[補題 2-2]
$x.rank$ は最初は 0 で, $x = x.p$ の間, 広義単調増加する.
$x \neq x.p$ になった以降, $x.rank$ は変化しない.

[証明] 疑似コードより

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

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

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

# $rank$ に関する性質(3/3)

[補題 2-3]
全ての頂点 $x$ に対して, $x.rank \leq n-1$ が成立.

# $rank$ に関する性質(3/3)

[補題 2-3]
全ての頂点 $x$ に対して, $x.rank \leq n-1$ が成立.

[証明]

頂点の数が $n$ なので

# 今日の目次

- アッカーマン関数について
- $rank$ に関する性質
- **ポテンシャル法**
- ポテンシャル関数を導入のための準備
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

# ポテンシャル法(1/3)



- ポテンシャル法とはデータ構造に対して定義されるポテンシャル関数を用いた, ならし計算量解析のテクニック.

- ポテンシャル法の考え方を説明し, 簡単な具体例で実際に計算量を解析する.

# ポテンシャル法(2/3)



- 初期データ構造 $D_0$ に対する $n$ 回の操作の実行を考える.
- $i = 1, 2, \ldots, n$ に対して,

  $c_i$ := $i$ 番目の操作の実コスト

  $D_i$ := $i$ 番目の操作後のデータ構造

  $\Phi$ := 各データ構造 $D_i$ から, ある実数 $\Phi(D_i)$ を得る関数と定義する.

$i$ 番目の操作のならしコスト $\hat{c}_i$ を,

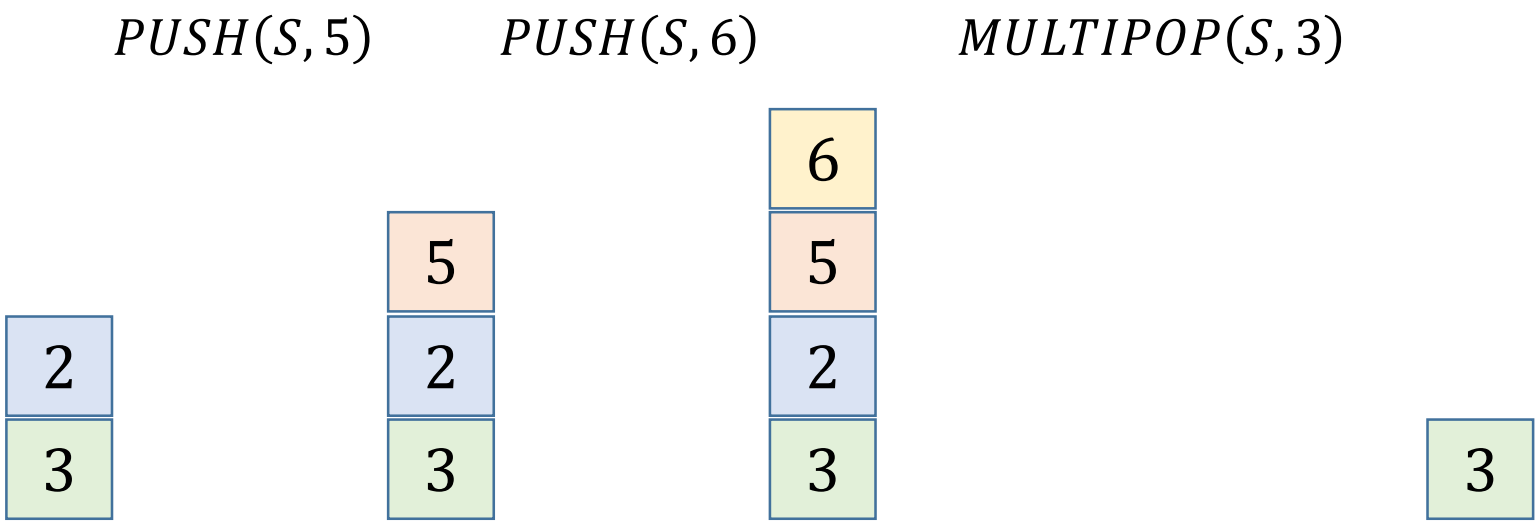
$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ と定義する.

# ポテンシャル法(3/3)

- $\sum_{i=0}^{n} \widehat{c_i} = \sum_{i=0}^{n} \left( c_i + \Phi(D_i) - \Phi(D_{i-1}) \right)$
  $= \sum_{i=0}^{n} c_i + \Phi(D_n) - \Phi(D_0)$
  が成立する.
- 全ての $n$ に対して, $\Phi(D_n) - \Phi(D_0) \geq 0$ が成立するように
  ポテンシャル関数を定めると,
  $\sum_{i=0}^{n} \widehat{c_i} \geq \sum_{i=0}^{n} c_i$ が成立し, 総ならしコストが, 総実コストの
  上界となる.

# スタック

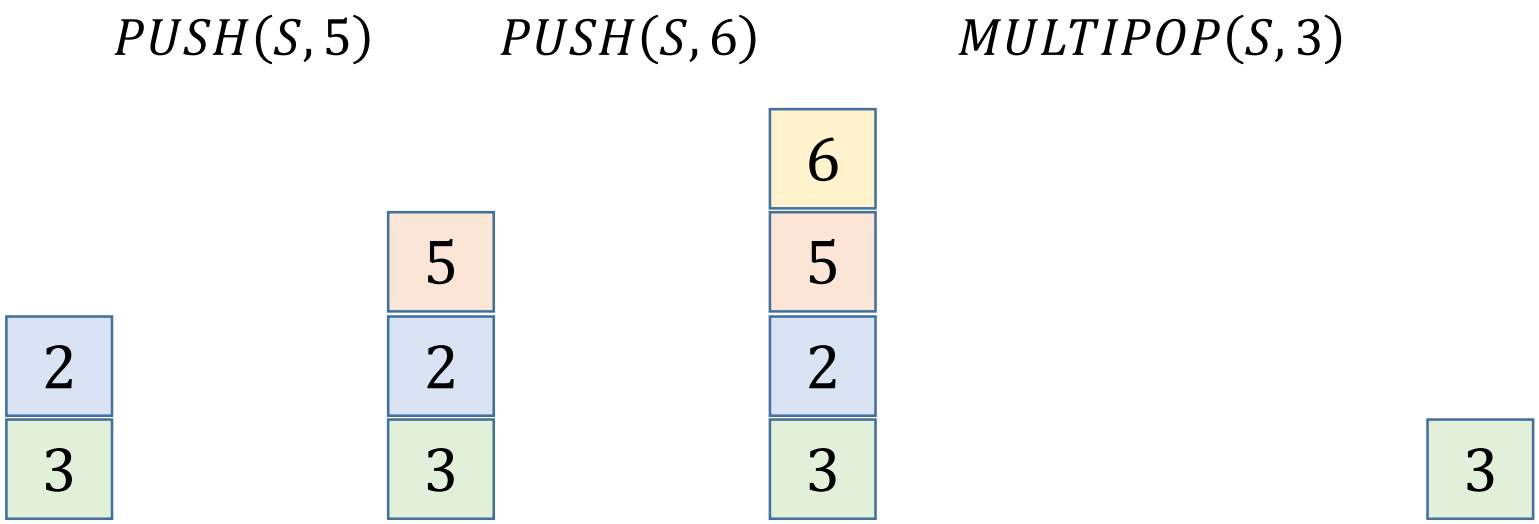
- スタックを例にとり, ポテンシャル法を適用する.
  - スタック $S$ := 以下の二つの操作ができるデータ構造.
  - $PUSH(S, x)$ := 要素 $x$ を $S$ の先頭に追加する操作.
  - $POP(S)$ := $S$ の上から 1 個の要素を取り除く操作.
  - $MULTIPOP(S, k)$ := $S$ の上から $\min(S.\text{size}, k)$ 個の要素を取り除く操作.
    - $S.size$ = スタックに積まれている要素数.



# 計算量解析(工夫なし)



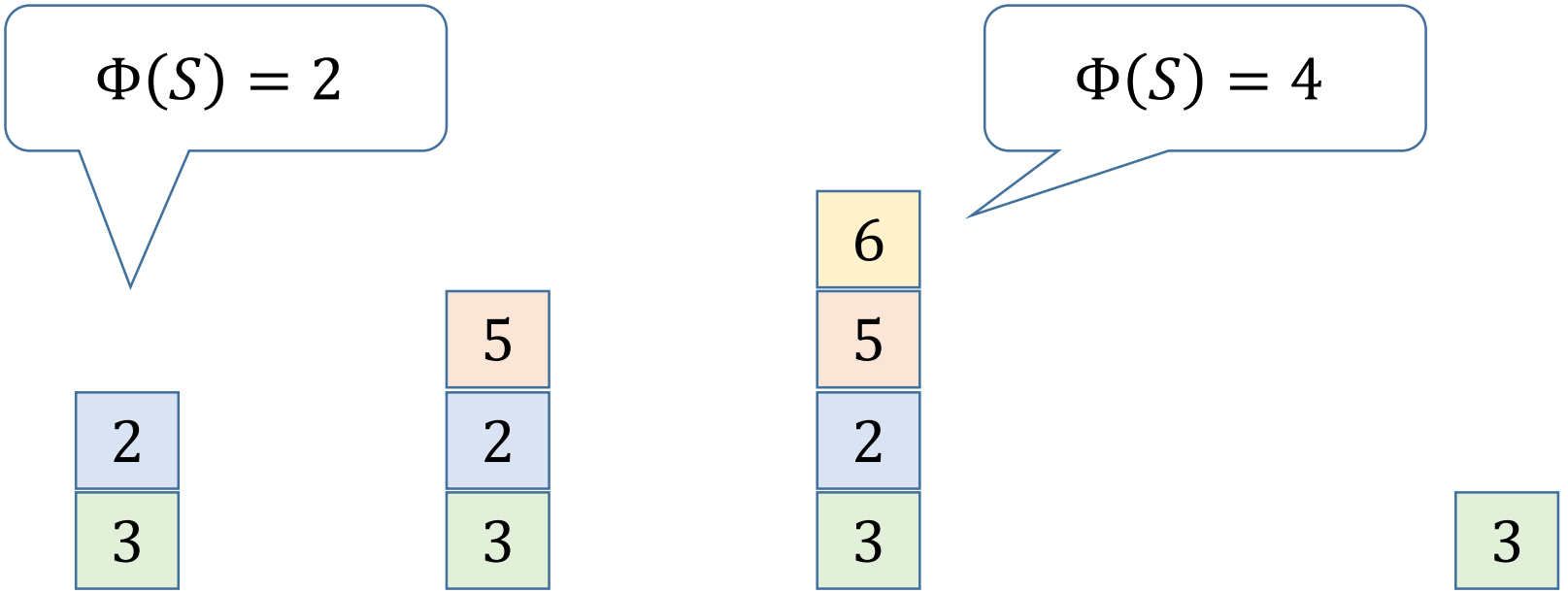
- 操作回数を $n$ とする.
- $S.size$ は $n$ まで大きくなりうるから,
  $MULTIPOP$ 操作一回の最悪計算量は $O(n)$ である.
- $MULTIPOP$ が $n$ 回実行される可能性があるから,
  操作全体の計算量は $O(n^2)$ である.



# 計算量解析(ポテンシャル法)



- ポテンシャル関数を $\Phi(S) := S.size$ と定める.
  - 全ての $n$ に対して, $\Phi(D_n) - \Phi(D_0) \geq 0$ が成立.



# $PUSH(S, x)$

$i$ 番目の操作が $s$ 個の要素を含むスタックへの $PUSH$ 操作と仮定.

ポテンシャル差は,
$\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$ である.

この操作の実コストは $c_i = 1$ であるので,
ならしコストは

$$\begin{aligned}\widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= O(1) + 1 = O(1)\end{aligned}$$

となる.

# $MULTIPOP(S, k)$

$i$ 番目の操作が $s$ 個の要素を含むスタックへの $MULTIPOP(S, k)$ であり, $k' = \min(s, k)$ 個の要素がポップされるものと仮定.

ポテンシャル差は,
$\Phi(D_i) - \Phi(D_{i-1}) = (s + 1) - s = -k'$ である.

この操作の実コストは $c_i = O(k')$ であるので,
ならしコストは

$$\begin{aligned}\widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= O(k') - k'\end{aligned}$$

となる.

# $POP(S, k)$

$MULTIPOP(S, k)$ の特殊なケースとみなせばよい.

# 計算量解析(ポテンシャル法)



各操作のならしコストは以下のようになる.

- $PUSH: O(1)$
- $MULTIPOP: O(k') - k'$

ここでポテンシャル関数の 1 単位を**十分大きくとると**,
$O(k') - k' = O(1)$ となる.

よって, 長さ $n$ の操作列を実行したときの総ならしコストは $O(n)$ となる.
総ならしコストは総実コストの上界であるので,
長さ $n$ の操作列を実行した時の時間計算量は
$O(n)$ となり, ならし計算量は $O(1)$ となる.

# 今日の目次

- アッカーマン関数について
- $rank$ に関する性質
- ポテンシャル法
- **ポテンシャル関数を導入のための準備**
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

# $level(x)$ 関数



- 第一の補助関数 $level(x)$ を以下のように定める.

[定義]
根ではなく, $x.rank \geq 1$ であるようなノード $x$ に対して
$level(x) = \max\{k : x.p.rank \geq A_k(x.rank)\}$

$x.rank$ は不変であり, $x.p.rank$ は時間とともに増加するので, $level(x)$ も時間とともに増加する.

# $level(x)$ 関数に関する性質



- $level(x)$ に対して, 以下の不等式が成立.

[補題 3-1]

$0 \leq level(x) < \alpha(n)$

# $level(x)$ 関数に関する性質



- $level(x)$ に対して, 以下の不等式が成立.

[補題 3-1]
$0 \leq level(x) < \alpha(n)$

[証明(左)]

$x.p.rank \geq x.rank + 1$ ($\because$ 補題 2–1)

$= A_0(x.rank)$ ($\because A_0(j)$ の定義)

よって, $0 \leq level(x)$ が成立.

# $level(x)$ 関数に関する性質



- $level(x)$ に対して, 以下の不等式が成立.

[補題 3-1]
$0 \leq level(x) < \alpha(n)$

[証明(右)]

$$A_{\alpha(n)}(x.rank) \geq A_{\alpha(n)}(1) \quad (\because A_k(j) \text{ は単調増加する})$$
$$\geq n \quad (\because \alpha(n) \text{ の定義})$$
$$> x.p.rank \quad (\because \text{補題 2–3})$$

よって, $level(x) < \alpha(n)$ が成立.

以上より, 補題 3-1 が成立.

# $iter(x)$ 関数

- 第二の補助関数 $iter(x)$ を以下のように定める.

[定義]
根ではなく, $x.rank \geq 1$ であるような頂点 $x$ に対して,

$$iter(x) = \max\left\{i : x.p.rank \geq A^{(i)}_{level(x)}(x.rank)\right\}$$

# $iter(x)$ 関数に関する性質



- $iter(x)$ に対して, 以下の不等式が成立.

[補題 3-2]

$1 \leq iter(x) \leq x.rank$

# $iter(x)$ 関数に関する性質



- $iter(x)$ に対して, 以下の不等式が成立.

[補題 3-2]

$1 \leq iter(x) \leq x.rank$

[証明(左)]

$x.p.rank \geq A_{level(x)}(x.rank)$ ($\because level(x)$ の定義)

$= A^{(1)}_{level(x)}(x.rank)$

よって, $1 \leq iter(x)$ が成立.

# $iter(x)$ 関数に関する性質



- $iter(x)$ に対して, 以下の不等式が成立.

[補題 3-2]

$1 \leq iter(x) \leq x.rank$

[証明(右)]

$$A_{level(x)}^{(x.rank+1)}(x.rank) = A_{level(x)+1}(x.rank) \quad (\because A_k\ (j) \text{ の定義})$$

$$> x.p.rank \quad (\because level(x) \text{ の定義})$$

よって, $iter(x) \leq x.rank$ が成立.

# $iter(x)$ 関数に関する性質



- $iter(x)$ に対して, 以下の不等式が成立.

[補題 3-2]
$1 \leq iter(x) \leq x.rank$

[証明(右)]

$$A_{level(x)}^{(x.rank+1)}(x.rank) = A_{level(x)+1}(x.rank) \quad (\because A_k\ (j) \text{ の定義})$$
$$> x.p.rank \quad (\because level(x) \text{ の定義})$$

説明）

よって, $iter(x) \leq x.rank$ が成立.

以上より, 補題 3-2 が成立.

# 今日の目次

- アッカーマン関数について
- rank に関する性質
- ポテンシャル法
- ポテンシャル関数導入のための準備
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

# ポテンシャル関数の導入



- ポテンシャル関数 $\Phi_q(D), \phi_q(x)$ を以下のように定める.

[定義]

$q$ 回目の操作後の素集合森 $\mathcal{G}$ に対して,

$\Phi_q(D) = \sum_x \phi_q(x)$.

ただし, 各ノード $x$ に対して,

$$\phi_q(x) = \begin{cases} \alpha(n) \cdot x.rank & \text{if } (\mathbf{1}) \\ \big(\alpha(n) - level(x)\big) \cdot x.rank - iter(x) & \text{otherwise} \end{cases}$$

$(\mathbf{1})$ $x$ が根, または $x.rank = 0$ の時

# $\phi_q(x)$ に関する性質(1/2)

[補題 4-1]

全ての頂点 $x$ 対して,

$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$

また,根ではなく, $x.rank \geq 1$ であるような頂点 $x$ に対して,

$0 \leq \phi_q(x) < \alpha(n) \cdot x.rank$

# $\phi_q(x)$ に関する性質(1/2)

[補題 4-1]
全ての頂点 $x$ 対して,
$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$
また,根ではなく, $x.rank \geq 1$ であるような頂点 $x$ に対して,
$0 \leq \phi_q(x) < \alpha(n) \cdot x.rank$

[証明]
$x$ が根, または $x.rank = 0$ のとき
定義より, $\phi_q(x) = \alpha(n) \cdot x.rank$
であるので, 上記の不等式は成立.

# $\phi_q(x)$ に関する性質(1/2)

[補題 4-1]
全ての頂点 $x$ 対して,
$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$
また, 根ではなく, $x.rank \geq 1$ であるような頂点 $x$ に対して,
$0 \leq \phi_q(x) < \alpha(n) \cdot x.rank$

[証明(左)]
$x$ が根でなく, $x.rank \geq 1$ の時

$$\begin{aligned}\phi_q(x) &= \big(\alpha(n) - level(x)\big) \cdot x.rank - iter(x) \\ &\geq \big(a(n) - (\alpha(n) - 1)\big) \cdot x.rank - x.rank \\ &= x.rank - x.rank = 0\end{aligned}$$

よって, $0 \leq \phi_q(x)$ が成立.

# $\phi_q(x)$ に関する性質(1/2)

[補題 4-1]
全ての頂点 $x$ 対して,
$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$
また, 根ではなく, $x.rank \geq 1$ であるような頂点 $x$ に対して,
$0 \leq \phi_q(x) < \alpha(n) \cdot x.rank$

[証明(右)]
$x$ が根でなく, $x.rank \geq 1$ の時

$$\begin{aligned}\phi_q(x) &= \big(\alpha(n) - level(x)\big) \cdot x.rank - iter(x) \\ &\leq (a(n) - 0) \cdot x.rank - 1 \\ &= \alpha(n) \cdot x.rank - 1 < \alpha(n) \cdot x.rank\end{aligned}$$

よって, $\phi_q(x) < \alpha(n) \cdot x.rank$ が成立.

# $\phi_q(x)$ に関する性質(1/2)

[補題 4-1]
全ての頂点 $x$ 対して,
$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$
また, 根ではなく, $x.rank \geq 1$ であるような頂点 $x$ に対して,
$0 \leq \phi_q(x) < \alpha(n) \cdot x.rank$

[証明(右)]
$x$ が根でなく, $x.rank \geq 1$ の時

$$\begin{aligned}\phi_q(x) &= \big(\alpha(n) - level(x)\big) \cdot x.rank - iter(x) \\ &\leq (a(n) - 0) \cdot x.rank - 1 \\ &= \alpha(n) \cdot x.rank - 1 < \alpha(n) \cdot x.rank\end{aligned}$$

よって, $\phi_q(x) < \alpha(n) \cdot x.rank$ が成立.
以上より, 補題 4-1 が成立.

# $\phi_q(x)$ に関する性質(2/2)

[補題 4-2]

$q$ 回目の操作が $LINK$ または $FIND\text{-}SET$ だと仮定する.

根でない頂点 $x$ に対して, $\phi_q(x) \leq \phi_{q-1}(x)$ が成立.

さらに, $x.rank \geq 1$ を満たし, $level(x)$ か $iter(x)$ が

$q$ 回目の操作で変化するなら $\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

# $\phi_q(x)$ に関する性質(2/2)

[補題 4-2]
$q$ 回目の操作が $LINK$ または $FIND\text{-}SET$ だと仮定する.
根でない頂点 $x$ に対して, $\phi_q(x) \leq \phi_{q-1}(x)$ が成立.
さらに, $x.rank \geq 1$ を満たし, $level(x)$ か $iter(x)$ が
$q$ 回目の操作で変化するなら $\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

[証明]
$x$ は根ではないので, $x.rank$ は**不変**である.
$(1)$ $x.rank = 0$ の時, $\phi_q(x) = \phi_{q-1}(x) = 0$.
以降, $x.rank \geq 1$ とする.

# $\phi_q(x)$ の性質(2/2)

[証明]

$x.rank \geq 1$ と仮定する.

$q$ 回目の操作で $level(x)$ が**変化しない**時, $iter(x)$ は**変化しないか増加するか**のどちらか.

(1) $level(x), iter(x)$ ともに不変のとき,

$\phi_q(x) = \phi_{q-1}(x)$ が成立.

(2) $level(x)$ が不変で, $iter(x)$ が増加するとき,

ポテンシャルは少なくとも 1 減り

$\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

# $\phi_q(x)$ に関する性質(2/2)

[証明]

$x.rank \geq 1$ と仮定する.

$q$ 回目の操作で $level(x)$ が**増加する**時, 少なくとも 1 増えるので,

$\left(\alpha(n) - level(x)\right) \cdot x.rank$ の部分は少なくとも $x.rank$ 減る.

$-iter(x)$ の部分は高々 $x.rank - 1$ 増える.

よって, ポテンシャルは少なくとも 1 減るので

$\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

# $\phi_q(x)$ に関する性質(2/2)

[補題 4-2]
$q$ 回目の操作が $LINK$ または $FIND\text{-}SET$ だと仮定する.
根でないノード $x$ に対して $\phi_q(x) \leq \phi_{q-1}(x)$ が成立.
さらに $x.rank \geq 1$ かつ $level(x)$ または $iter(x)$ が
$q$ 回目の操作で変化するなら $\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

[証明]

以上より, 補題 4-2 が成立.

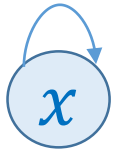
# 今日の目次

- アッカーマン関数について
- $rank$ に関する性質
- ポテンシャル法
- ポテンシャル関数導入のための準備
- ポテンシャル関数に関する性質
- **各操作に対してのならし解析**

# ！！各操作のならしコスト(1/3)

[補題 5-1]
操作 $MAKE\text{-}SET(x)$ のならしコストは $O(1)$ である.

[証明]
$q$ 回目の操作が $MAKE\text{-}SET(x)$ だと仮定する.
$x.rank = 0$ であるので, $\phi_q(x) = 0$.
$x$ 以外のランクは変化しないので, $\Phi_q(\mathcal{G}) = \Phi_{q-1}(\mathcal{G})$ である.
$\mathcal{G}$ なに
$MAKE\text{-}SET(x)$ の実コストは $O(1)$ なので,
$MAKE\text{-}SET(x)$ のならしコストは $O(1)$.

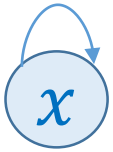


# 各操作のならしコスト(1/3)

[補題 5-1]
操作 $MAKE\text{-}SET(x)$ のならしコストは $O(1)$ である.

[証明]
$q$ 回目の操作が $MAKE\text{-}SET(x)$ だと仮定する.
$x.rank = 0$ であるので, $\phi_q(x) = 0$.
$x$ 以外のランクは変化しないので, $\Phi_q(\mathcal{G}) = \Phi_{q-1}(\mathcal{G})$ である.

$MAKE\text{-}SET(x)$ の実コストは $O(1)$ なので,
$MAKE\text{-}SET(x)$ のならしコストは $O(1)$.




以上より, 補題 5-2 が成立.

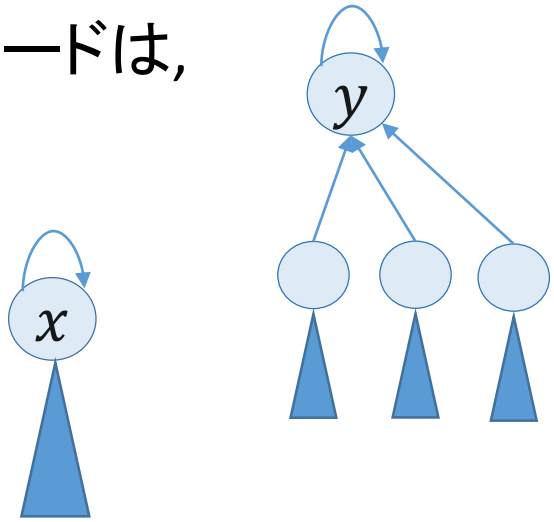
# 各操作のならしコスト(2/3)

[補題 5-2]
操作 $LINK(x, y)$ のならしコストは $O(\alpha(n))$ である.

[証明]
$q$ 回目の操作が $LINK(x, y)$ であると仮定する.
$LINK(x, y)$ の実コストは $O(1)$ である.
一般性を失わずに, $x.par$ を $y$ に更新することとする.
この時, ポテンシャルが変化する可能性があるノードは,

1. $y$
2. $x$
3. $y$ の(直接の)子であったノード

のいずれかである.



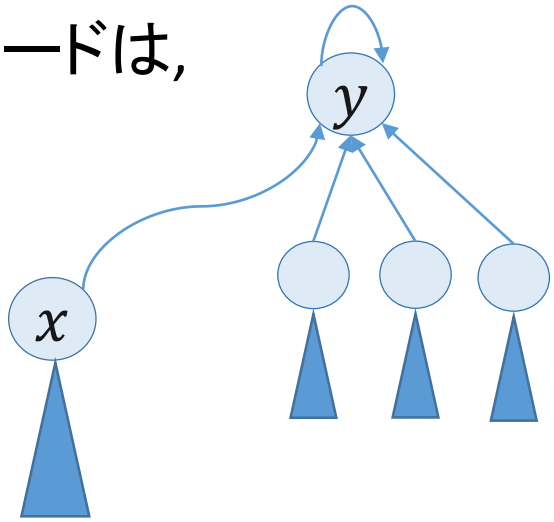
# 各操作のならしコスト(2/3)

[補題 5-2]
操作 $LINK(x, y)$ のならし計算量は $O(\alpha(n))$ である.

[証明]
$q$ 回目の操作が $LINK(x, y)$ であると仮定する.
$LINK(x, y)$ の実コストは $O(1)$ である.
一般性を失わずに, **$x.par$ を $y$ に更新**することとする.
この時, ポテンシャルが変化する可能性があるノードは,

1. $y$
2. $x$
3. $y$ の(直接の)子であったノード

のいずれかである.



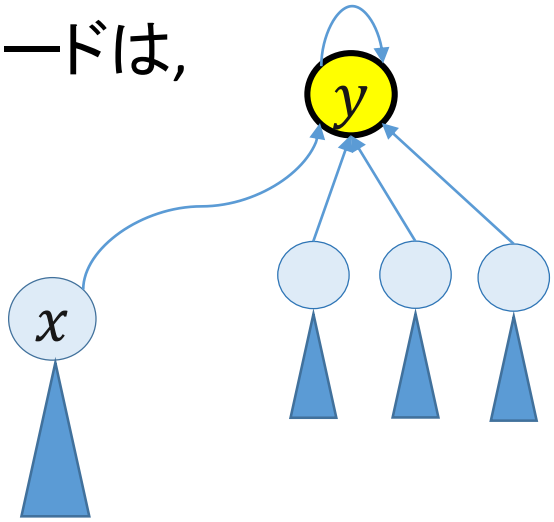
# 各操作のならしコスト(2/3)

[補題 5-2]
操作 $LINK(x, y)$ のならし計算量は $O(\alpha(n))$ である.

[証明]
$q$ 回目の操作が $LINK(x, y)$ であると仮定する.
$LINK(x, y)$ の実コストは $O(1)$ である.
一般性を失わずに, $x.par$ を $y$ に更新することとする.
この時, ポテンシャルが変化する可能性があるノードは,

1. $\boldsymbol{y}$
2. $x$
3. $y$ の(直接の)子であったノード

のいずれかである.



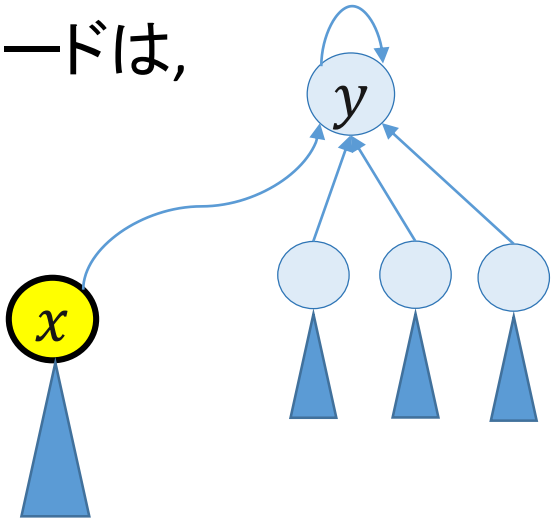
# 各操作のならしコスト(2/3)

[補題 5-2]
操作 $LINK(x, y)$ のならし計算量は $O(\alpha(n))$ である.

[証明]
$q$ 回目の操作が $LINK(x, y)$ であると仮定する.
$LINK(x, y)$ の実コストは $O(1)$ である.
一般性を失わずに, $x.par$ を $y$ に更新すると仮定.
この時, ポテンシャルが変化する可能性があるノードは,

1. $y$
2. $\boldsymbol{x}$
3. $y$ の(直接の)子であったノード

のいずれかである.



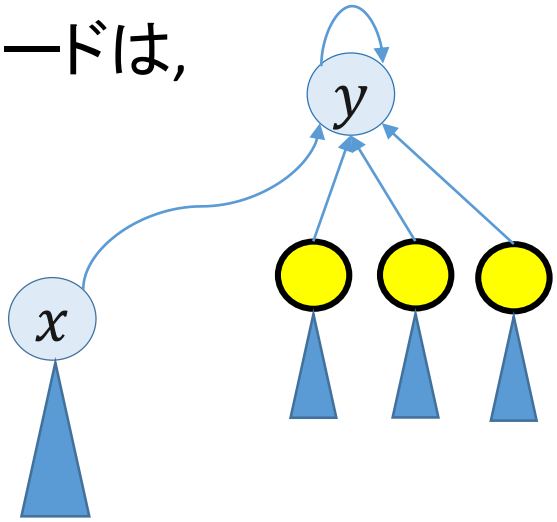
# 各操作のならしコスト(2/3)

[補題 5-2]
操作 $LINK(x, y)$ のならし計算量は $O(\alpha(n))$ である.

[証明]
$q$ 回目の操作が $LINK(x, y)$ であると仮定する.
$LINK(x, y)$ の実コストは $O(1)$ である.
一般性を失わずに, $x.par$ を $y$ に更新することとする.
この時, ポテンシャルが変化する可能性があるノードは,

1. $y$
2. $x$
3. **$y$ の(直接の)子であったノード**

のいずれかである.

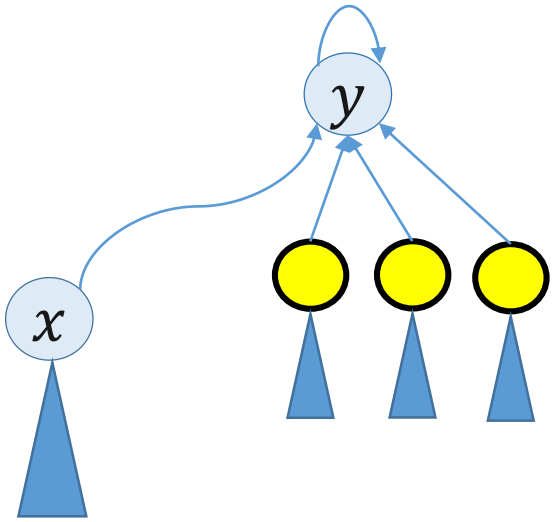


# 各操作のならしコスト(2/3)

[証明]

$y$ **の子**のポテンシャルの変化を観察.

補題 4-2 より, $y$ の子のポテンシャルは増加しない.



# 各操作のならしコスト(2/3)

[証明]

$x$ のポテンシャルの変化を観察.

$x$ は $q$ 番目の操作の直前は根なので, $\phi_{q-1}(x) = \alpha(n) \cdot x.rank$.

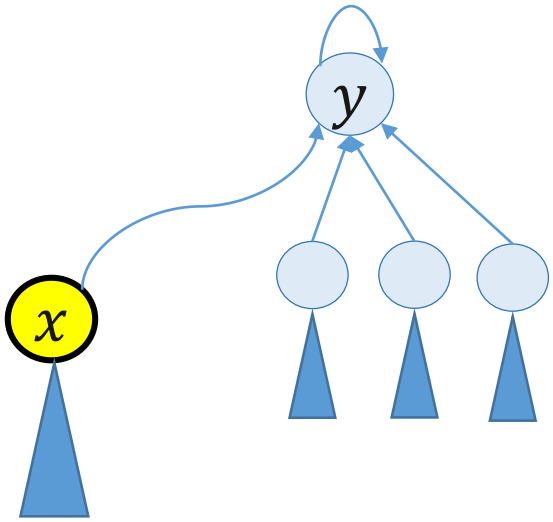
(1) $x.rank = 0$ のとき

$$\phi_q(x) = \phi_{q-1}(x) = 0.$$

(2) $x.rank \geq 0$ のとき

$$\phi_q(x) < \alpha(n) \cdot x.rank \quad (\because \text{ 補題 4–1})$$
$$= \phi_{q-1}(x)$$

よって, $x$ のポテンシャルは**増加しない**.



# 各操作のならしコスト(2/3)



[証明]

$y$ のポテンシャルの変化を観察.

$y$ は $q$ 番目の操作の直前は根なので, $\phi_{q-1}(x) = \alpha(n) \cdot x.rank$.

$y$ は操作後も根であるので $y.rank$ は変化しないか 1 だけ増加するかのどちらか.

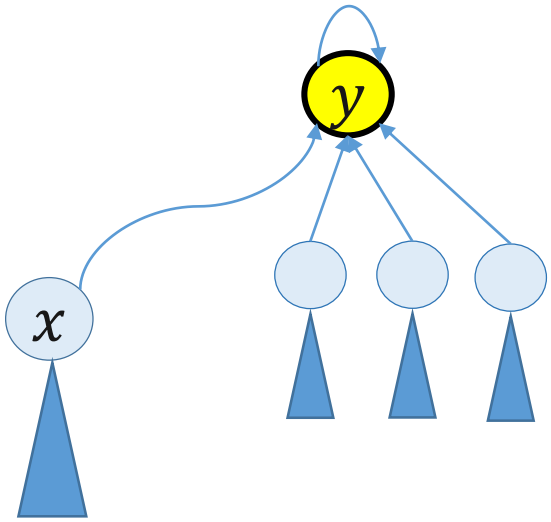
(1) $y.rank$ が変化しないとき

$$\phi_q(y) = \phi_{q-1}(y).$$

(2) $y.rank$ が 1 増加するとき

$$\phi_q(y) = \phi_{q-1}(y) + \alpha(n).$$

よって, $y$ のポテンシャル増加分は高々 $\alpha(n)$.



# 各操作のならしコスト(2/3)



[補題 5-2]

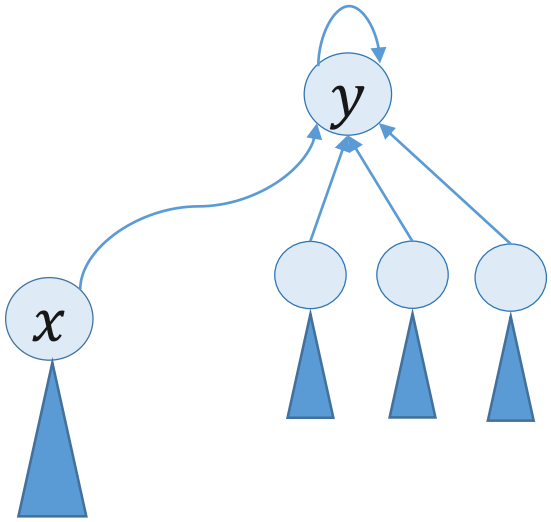
操作 $LINK(x, y)$ のならしコストは $O\big(\alpha(n)\big)$ である.

[証明]

以上より, $LINK(x, y)$ によって増加する可能性のある頂点は $y$ のみであり, 増加分は高々 $\alpha(n)$ であるので,

$LINK(x, y)$ のならしコストは

$O(1) + \alpha(n) = O\big(\alpha(n)\big)$.



# 各操作のならしコスト(2/3)

[補題 5-2]

操作 $LINK(x, y)$ のならしコストは $O\big(\alpha(n)\big)$ である.

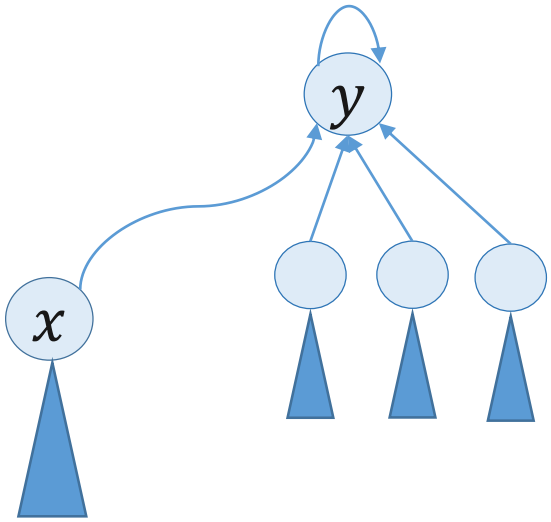
[証明]

以上より, $LINK(x, y)$ によって増加する可能性のある頂点は $y$ のみであり, 増加分は高々 $\alpha(n)$ であるので,

$LINK(x, y)$ のならしコストは

$O(1) + \alpha(n) = O\big(\alpha(n)\big)$.

以上より, 補題 5-2 が成立.



# 各操作のならしコスト(3/3)



[補題 5-3]

操作 $FIND\text{-}SET(x)$ のならしコストは $O(\alpha(n))$ である.

[証明]

$q$ 回目の操作が $FIND\text{-}SET(x)$ であり, $x$ から根までのパスに含まれる頂点数が $s$ であると仮定する.

$FIND\text{-}SET(x)$ の実コストは $O(s)$ である.

**以下の 2 つの事実を示す.**

(1) どの頂点もポテンシャルが増加しない.

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少する.

# 各操作のならしコスト(3/3)



[証明]

(1) どの頂点もポテンシャルが増加しないことを示す.

補題 4-2 より, 根でない頂点 $x$ に対して

$\phi_q(x) \leq \phi_{q-1}(x)$ となる.

$x$ が根のとき,

$\phi_q(x) = \phi_{q-1}(x) = \alpha(n) \cdot x.rank$ となる.

以上より, (1) が示せた.

# 各操作のならしコスト(3/3)

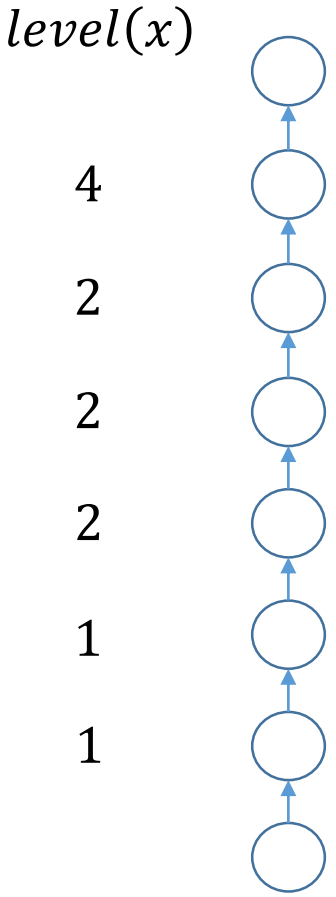


[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する



# 各操作のならしコスト(3/3)



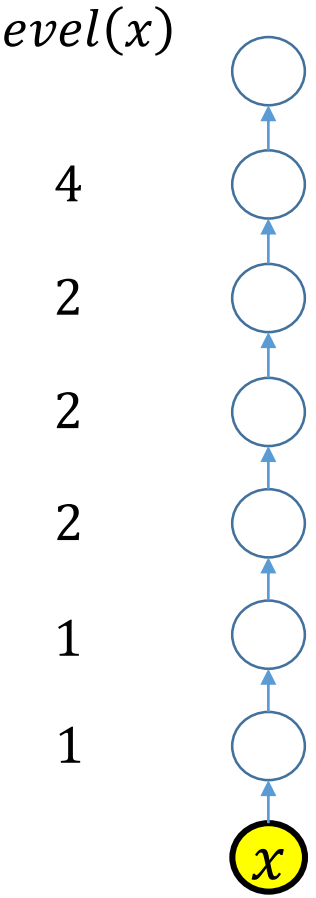
[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

$x.rank = 0$ であるので条件に反する



# 各操作のならしコスト(3/3)



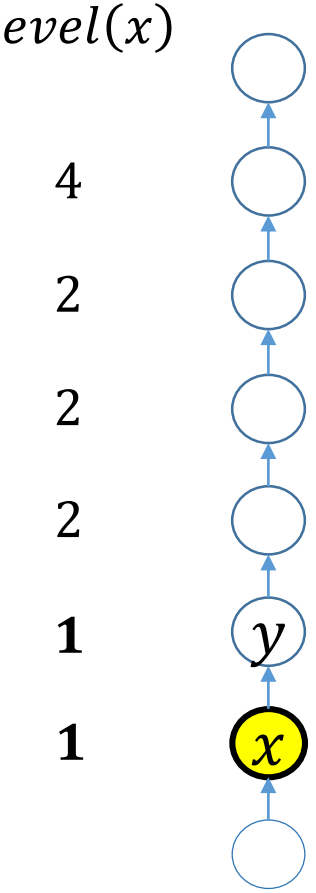
[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

$y$ が存在し, 条件を満たす



# 各操作のならしコスト(3/3)



[証明]

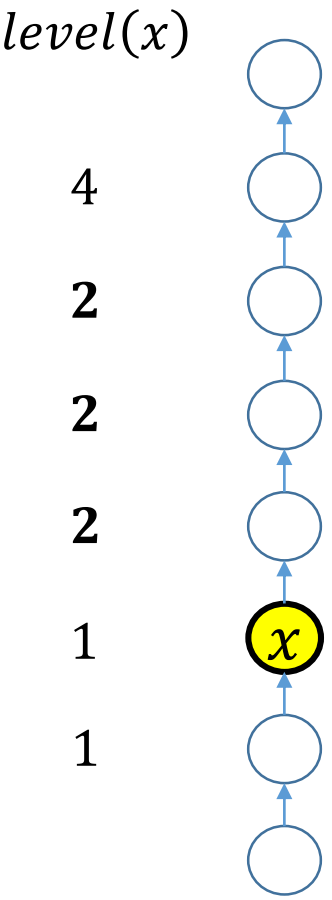
(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

$level(y) = 1$ を満たす $y$ が存在しないので

条件に反する



# 各操作のならしコスト(3/3)



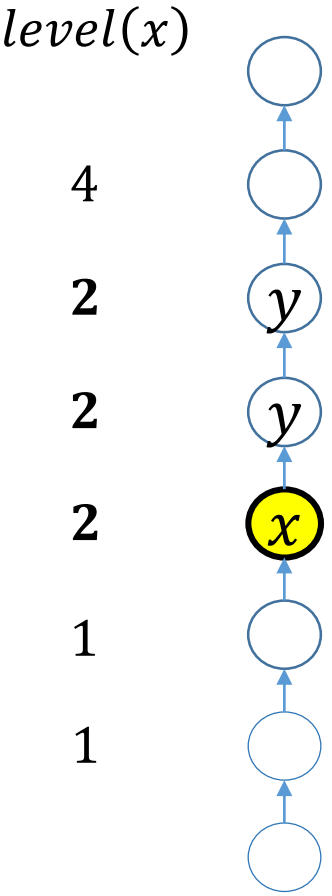
[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

$y$ が存在し, 条件を満たす



# 各操作のならしコスト(3/3)



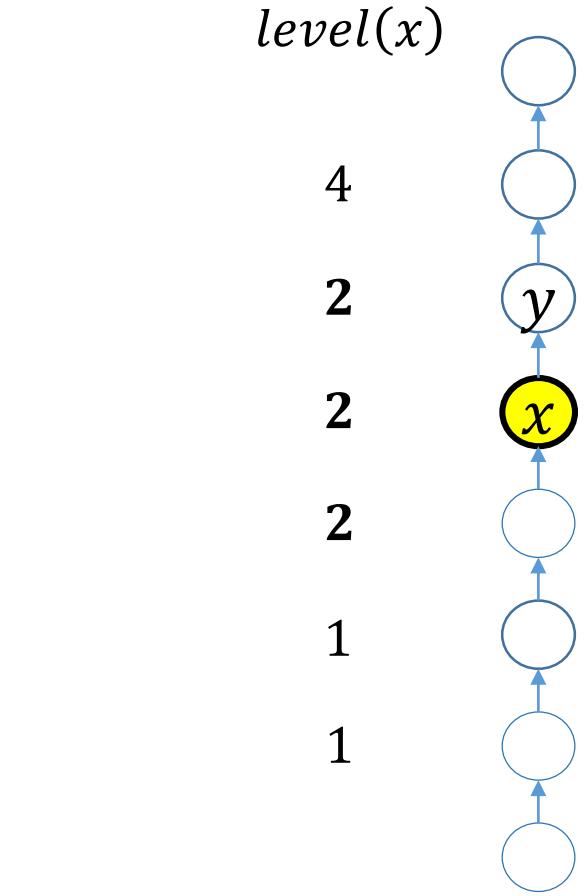
[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

$y$ が存在し, 条件を満たす



# 各操作のならしコスト(3/3)



[証明]

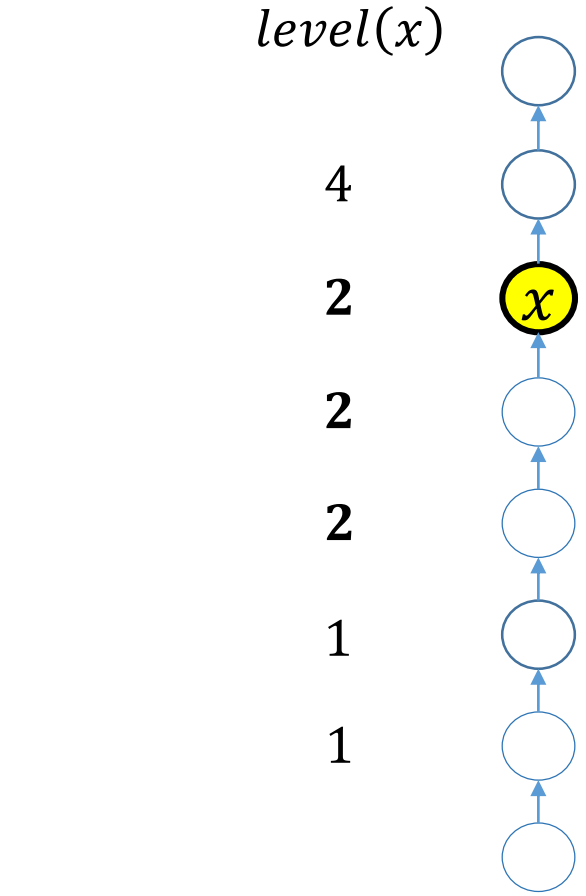
(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

$level(y) = 2$ を満たす $y$ が存在しないので

条件に反する



# 各操作のならしコスト(3/3)



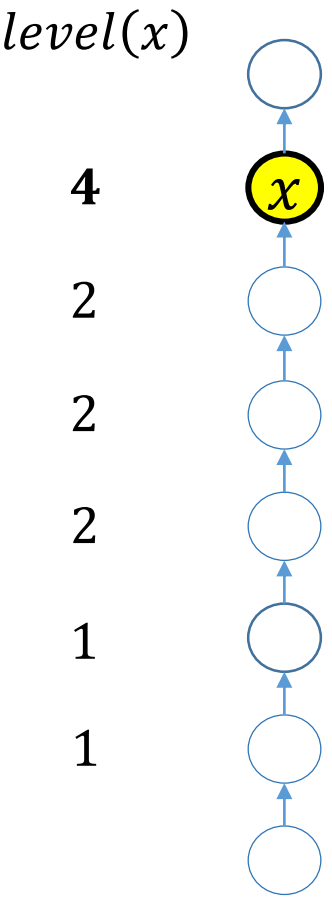
[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

そもそも根でない頂点が $x$ 以降存在しないので
条件に反する



# 各操作のならしコスト(3/3)



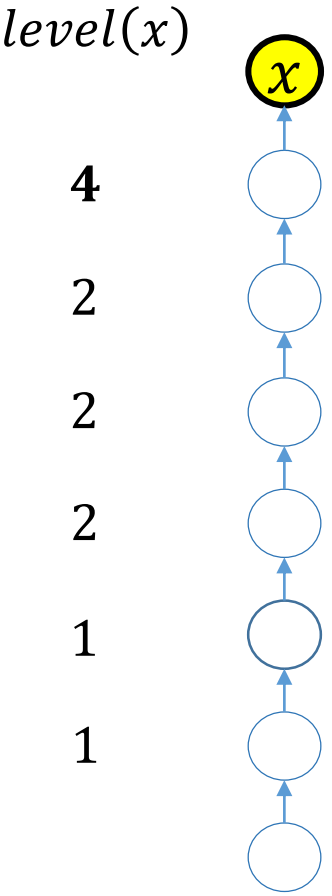
[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を以下の条件を満たす頂点だと仮定する.

1. $find\ path$ 上の頂点である
2. $x.rank > 0$ である
3. $level(x) = level(y)$ を満たす
   根でない頂点 $y$ が, $x$ 以降に存在する

そもそも頂点が $x$ 以降存在しないので
条件に反する



# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

根と葉の 2 頂点は条件を満たさない.

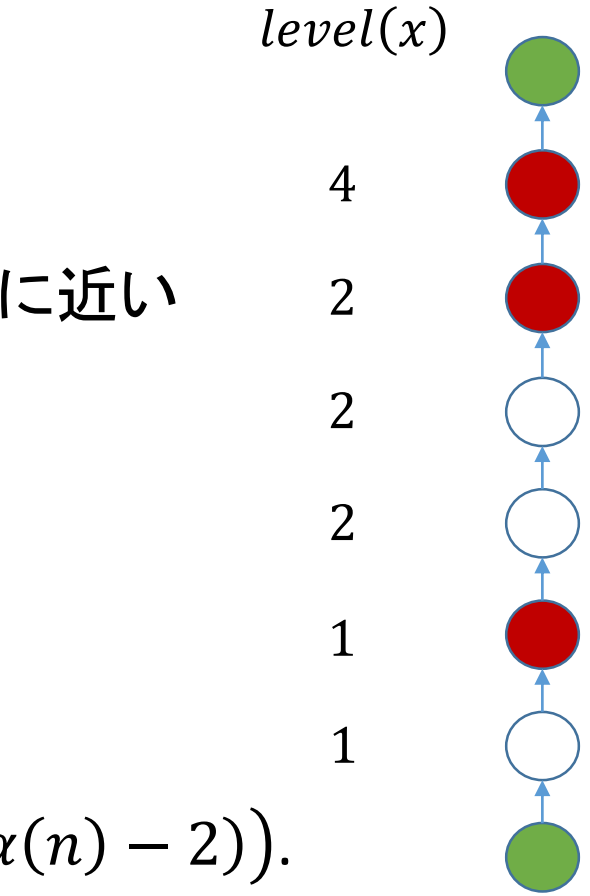
$level(x) = k$ となるような頂点のうち, 最も根に近い頂点は条件を満たさない

補題 3-1 より, $0 \leq k < \alpha(n)$ なので,

高々 $\alpha(n)$ 頂点は条件を満たさない.

条件を**満たさない**頂点は高々 $\alpha(n)+2$.

条件を**満たす**頂点は少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$.



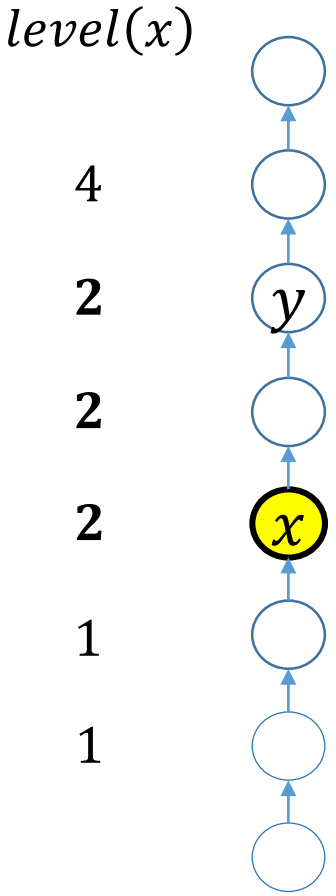
# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

$x$ を先述の条件を満たす頂点だと仮定し,
$x$ のポテンシャルが少なくとも 1 減少することを示す.



# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

操作前の頂点 $x$, 頂点 $y$ に対して以下の不等式が成立.

- ただし, $k = level(x) = level(y),\ i = iter(x)$ とする.

$$
\begin{aligned}
y.p.rank &\geq A_k(y.rank) && (\because level(x) \text{ の定義}) \\
&\geq A_k(x.p.rank) && (\because y.rank \geq x.p.rank) \\
&\geq A_k\left(A_k^i(x.rank)\right) && (\because iter(x) \text{ の定義}) \\
&= A_k^{i+1}(x.rank)
\end{aligned}
$$

# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

操作前の頂点 $x$, 頂点 $y$ に対して以下の不等式が成立.

- ただし, $k = level(x) = level(y),\ i = iter(x)$ とする.

$$y.p.rank \geq A_k^{i+1}(x.rank)$$

操作後, $x.p.rank = y.p.rank$ が成立し, $y.p.rank$ は減少しない.
また, $x.rank$ は変化しないから, 操作後の頂点 $x$, 頂点 $y$ に対して,

$$x.p.rank = y.p.rank \geq y.p.rank \geq A_k^{i+1}(x.rank)$$

が成立.

# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

操作前の頂点 $x$, 頂点 $y$ に対して以下の不等式が成立.

- ただし, $k = level(x) = level(y),\ i = iter(x)$ とする.

$$y.p.rank \geq A_k^{i+1}(x.rank)$$

操作後, $x.p.rank = y.p.rank$ が成立し, $y.p.rank$ は減少しない.
また, $x.rank$ は変化しないから, 操作後の頂点 $x$, 頂点 $y$ に対して,

$$x.p.rank = y.p.rank \geq y.p.rank \geq A_k^{i+1}(x.rank)$$

が成立.

# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

**操作前**

- $k = level(x) = level(y),\ i = iter(x)$

**操作後**

$x.p.rank \geq A_k^{i+1}(x.rank)$

操作後, $level(x)$ が変化しないとき, $iter(x)$ は少なくとも 1 増加.

操作後, $level(x)$ が変化するときと合わせて, 補題 4-2 より,

$\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max\left(0, s-(\alpha(n)-2)\right)$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

**操作前**

- $k = level(x) = level(y),\ i = iter(x)$

**操作後**

$x.p.rank \geq A_k^{i+1}(x.rank)$

操作後, $level(x)$ が変化しないとき, $iter(x)$ は少なくとも 1 増加.

操作後, $level(x)$ が変化するときと合わせて, 補題 4-2 より,

$\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

# 各操作のならしコスト(3/3)



[証明]

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少することを示す.

**操作前**

- $k = level(x) = level(y),\ i = iter(x)$

**操作後**

$x.p.rank \geq A_k^{i+1}(x.rank)$

操作後, $level(x)$ が変化しないとき, $iter(x)$ は少なくとも 1 増加.

操作後, $level(x)$ が変化するときと合わせて, 補題 4-2 より,

$\phi_q(x) \leq \phi_{q-1}(x) - 1$ が成立.

以上より, (2) が示せた.

# 各操作のならしコスト(3/3)



[補題 5-3]

操作 $FIND\text{-}SET(x)$ のならしコストは $O(\alpha(n))$ である.

[証明]

$q$ 回目の操作が $FIND\text{-}SET(x)$ であり, $x$ から根までのパスに含まれる頂点数が $s$ であると仮定する.

$FIND\text{-}SET(x)$ の実コストは $O(s)$ である.

**以上より, 以下の 2 つの事実を示した.**

(1) どの頂点もポテンシャルが増加しない.

(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少する.

# 各操作のならしコスト(3/3)

[補題 5-3]
操作 $FIND\text{-}SET(x)$ のならしコストは $O(\alpha(n))$ である.

[証明]
**以上より, 以下の 2 つの事実を示した.**
(1) どの頂点もポテンシャルが増加しない.
(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少する.
ここで, **ポテンシャル関数の 1 単位を十分大きくとると**,
ならしコストは $O(s)-(s-(\alpha(n)-2))=O(s)-s+O(\alpha(n))$
$=O(\alpha(n))$ となる.

# 各操作のならしコスト(3/3)

[補題 5-3]
操作 $FIND\text{-}SET(x)$ のならしコストは $O(\alpha(n))$ である.

[証明]
**以上より, 以下の 2 つの事実を示した.**
(1) どの頂点もポテンシャルが増加しない.
(2) $find\ pass$ 上の頂点のうち, 少なくとも $\max(0, s-(\alpha(n)-2))$ の頂点のポテンシャルが少なくとも 1 減少する.
ここで, **ポテンシャル関数の 1 単位を十分大きくとると**,
ならしコストは $O(s)-(s-(\alpha(n)-2))=O(s)-s+O(\alpha(n))$
$=O(\alpha(n))$ となる.

以上より, 補題 5-3 が成立.

# 計算量解析



- 補題 5-1, 5-2, 5-3 より, 操作 1 回あたりならし計算量は $O(\alpha(n))$ であることが示された.

# まとめ

- アッカーマン関数について
- $rank$ に関する性質
- ポテンシャル法
- ポテンシャル関数を導入のための準備
- ポテンシャル関数に関する性質
- 各操作に対してのならし解析

- 素集合データ構造の表現方法を 2 つ紹介した.
  - 素集合連結リスト
    - 操作 1 回あたりならしが計算量 $O(\log n)$であることを示した.
  - 素集合森
    - 操作 1 回あたりならし計算量が $O\left(\alpha(n)\right)$ であることを示した.