Union Find

～ 素集合データ構造 ～

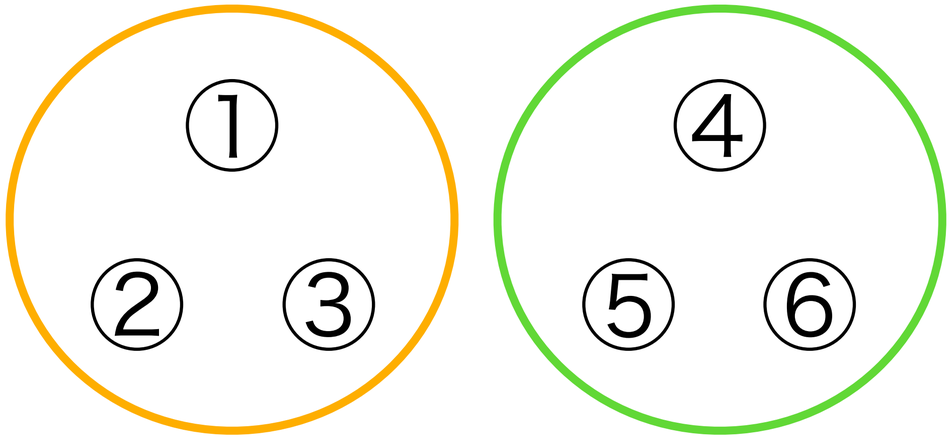
- Union Findとは
- Union Findの説明
- Union Findの計算量

Union Findとは

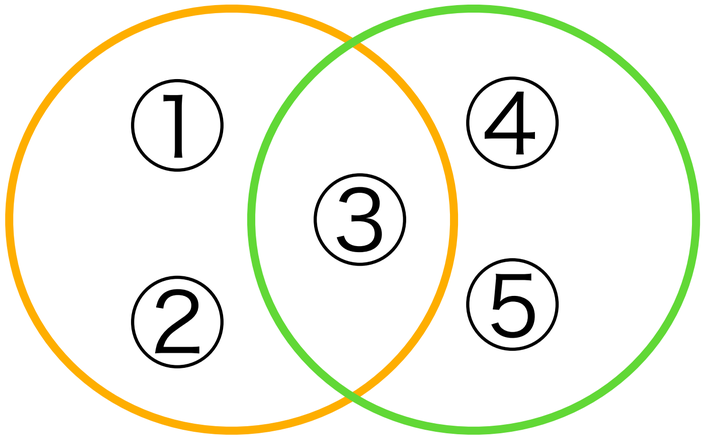
- 要素を素集合に分割して管理する仕組み（データ構造）

  素集合とは...?

  2つのグループが互いに重ならない，つまり共通部分を持たない



互いに重ならない



互いに重なる

Union Findとは

unite: 二つのグループを合併させる

find: ある要素がどのグループに属しているか調べる

上の二つの操作を行えるデータ構造

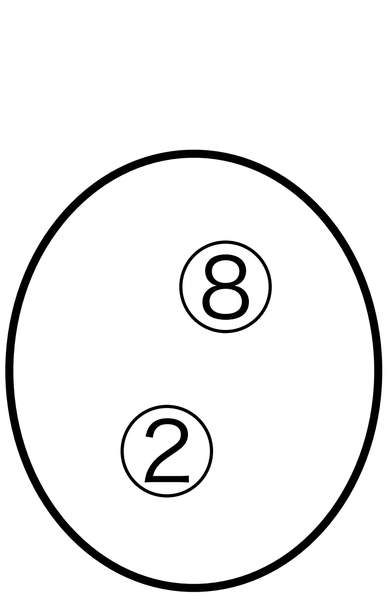
unite, findを高速に行える!

- Union Findとは
- Union Findの説明
- Union Findの計算量

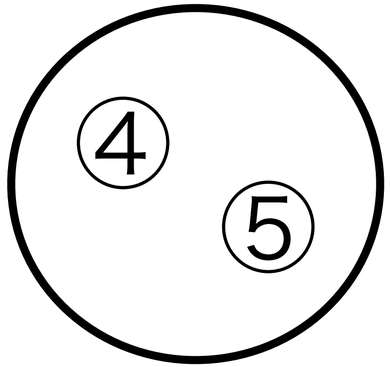
Union Findの説明

・unite, findの例

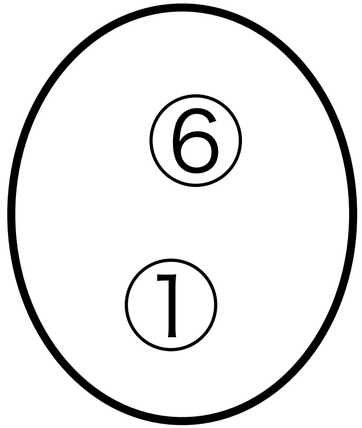
以下のようなグループを考える



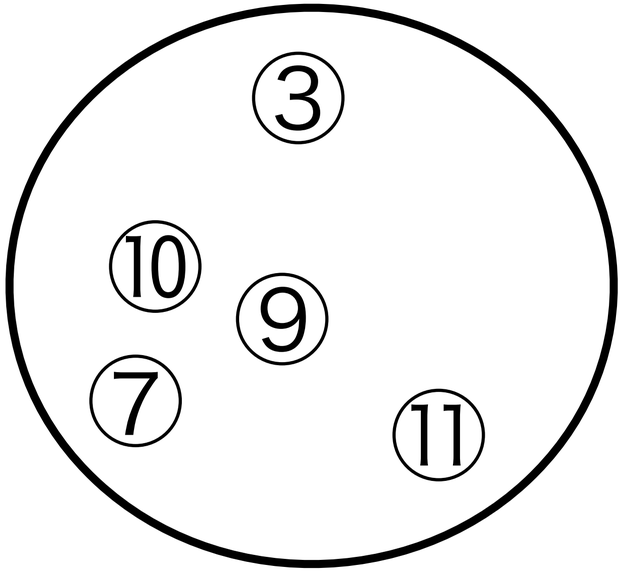
グループ1



グループ2



グループ3

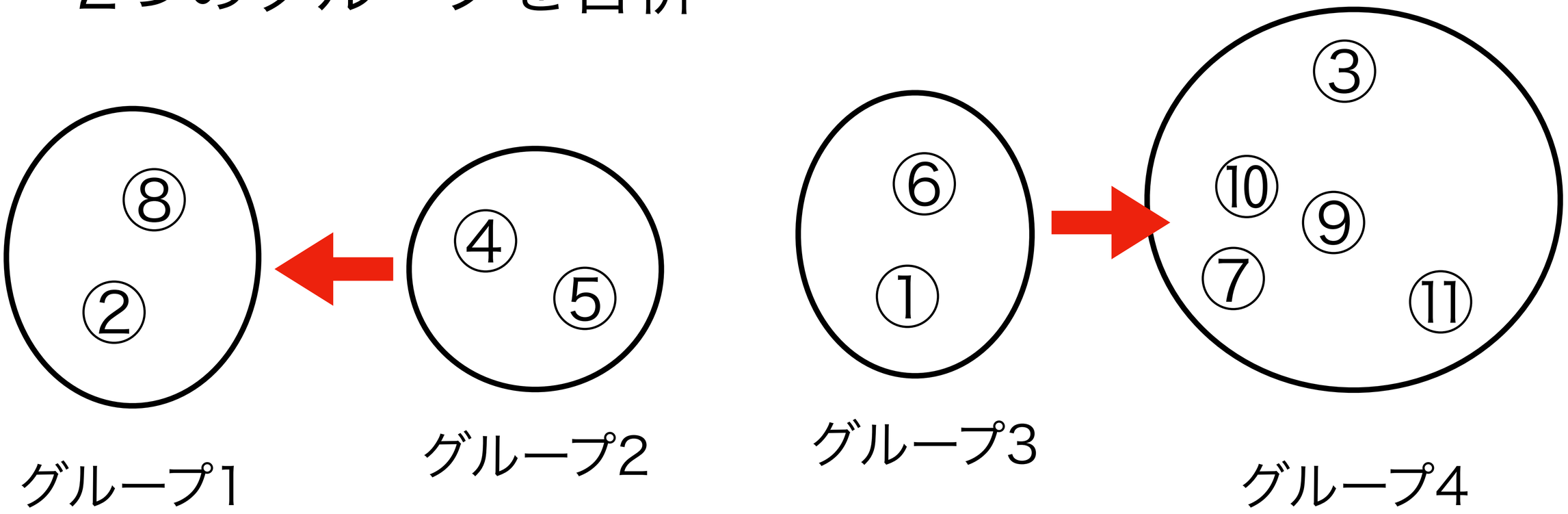


グループ4

Union Findの説明

・unite

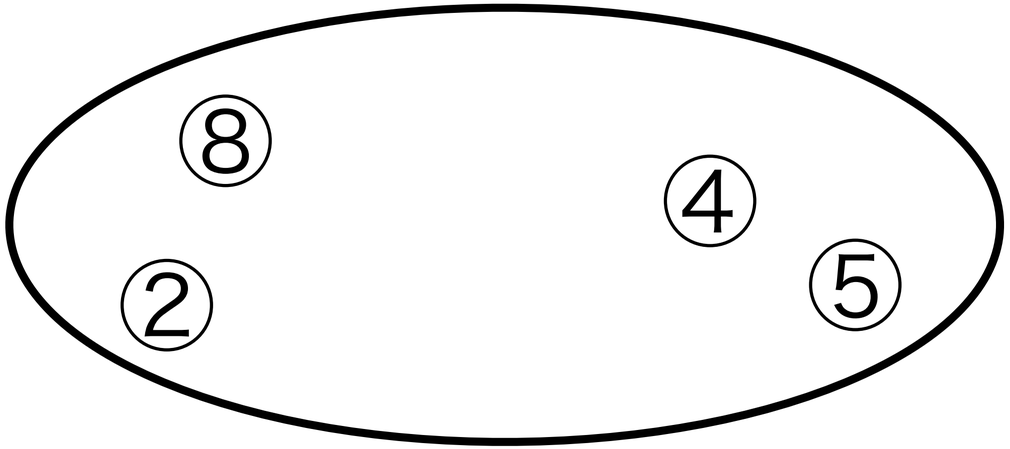
2つのグループを合併



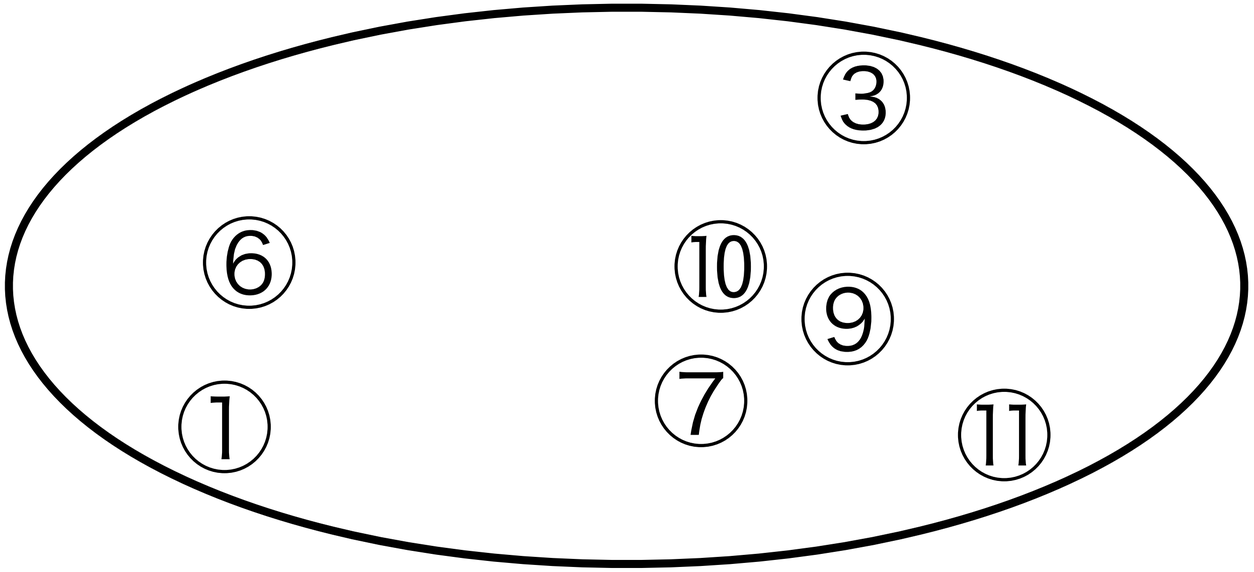
Union Findの説明

・unite

2つのグループを合併



グループ1

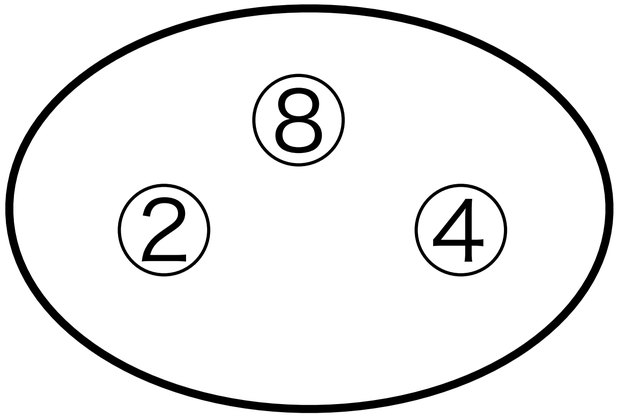


グループ4

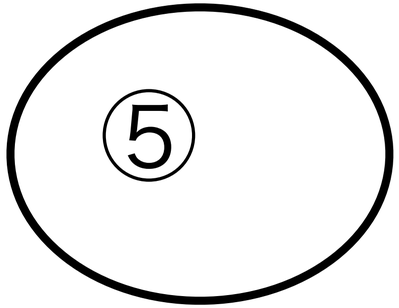
Union Findの説明

・find

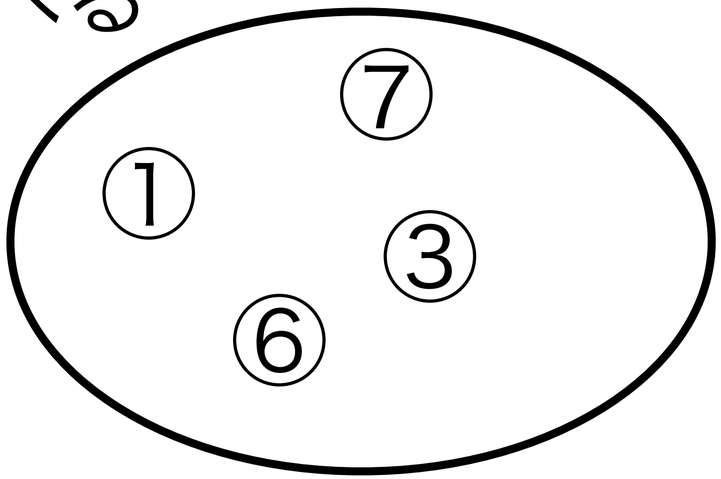
要素がどのグループに属しているか調べる



グループ1



グループ2



グループ3

➡どうやって調べる...???

Union Findの説明

❶: 代表値

①: それ以外の要素

・find

要素がどのグループに属しているか調べる

グループ1　　グループ2　　グループ3

それぞれのグループの代表値を決める

→ 代表値はグループ名のような働きをする

Union Findの説明

代表値

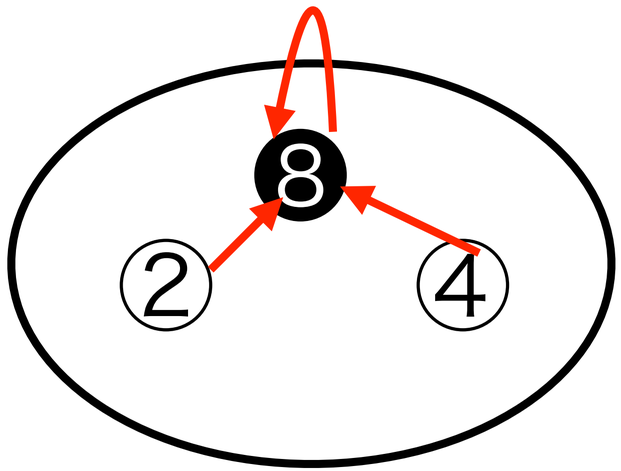
グループ1: ❽

グループ2: ❺

グループ3: ❸

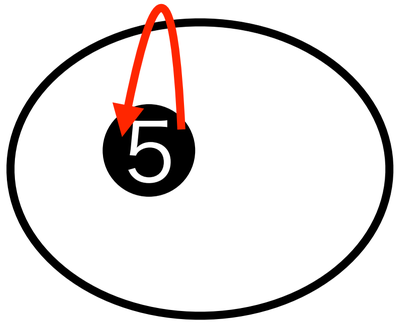
・find

要素がどのグループに属しているか調べる



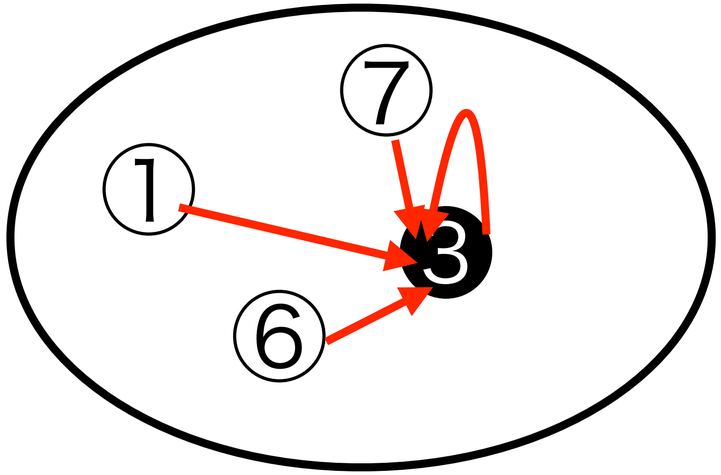
グループ1

(❽)



グループ2

(❺)



グループ3

(❸)

Union Findの説明

このようなUnion Findを，プログラミング言語で実装

→ どうやってunite, findを実装する?
　そもそも, どうやってグループや代表値を表現する??

Union Find木という木構造を使って実装する!!

Union Findの説明

・木構造の説明

木構造（きこうぞう）とは，グラフ理論の木の構造をしたデータ構造のこと。（Wikipediaより）

・データ構造の一種
・要素（ノード，節点）と辺（エッジ，枝）からなる
・要素と要素は辺でつながっている
・「親ノード」，「子ノード」などの親子関係で要素を表現
・ 根ノード(root): 親を持たないノードで，木構造の最上位

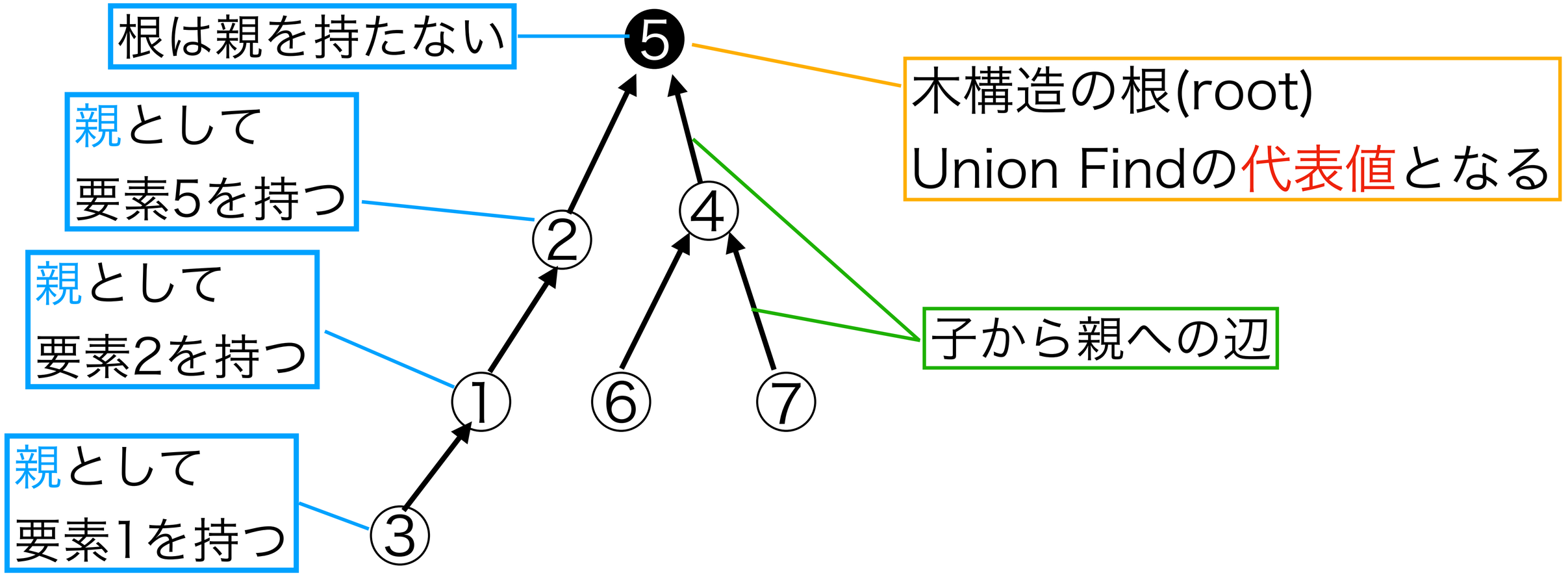
etc...

Union Findの説明

・Union Find木の説明

❶: 根

①: それ以外の要素



Union Findの説明

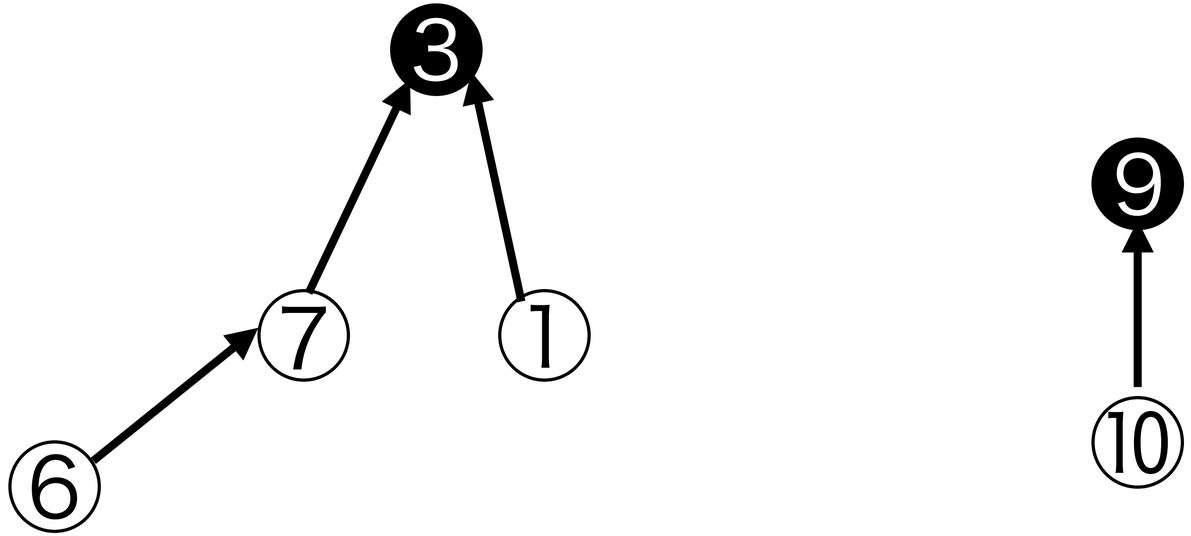
・unite

2つのグループを合併

❶: 根

①: それ以外の要素

例: unite(3, 9)



Union Findの説明

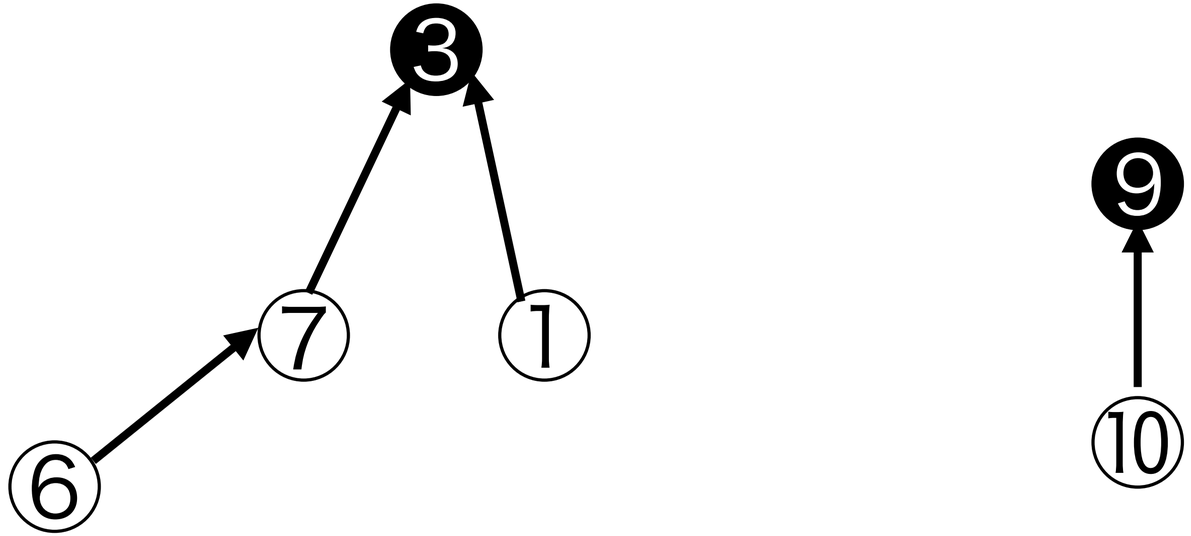
❶: 根

①: それ以外の要素

・unite

2つのグループを合併

例: unite(3, 9)

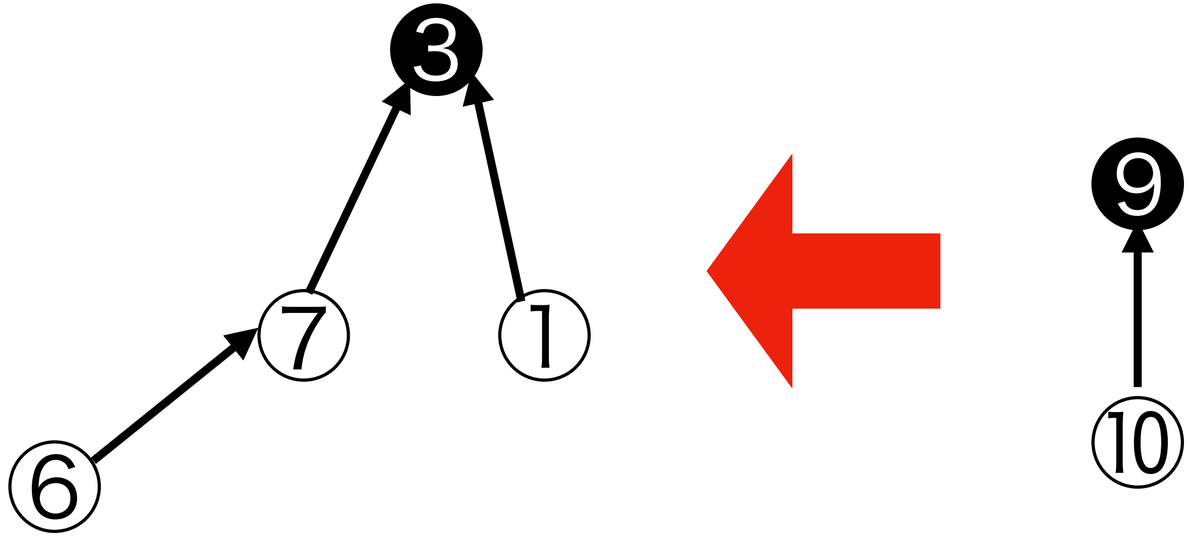


合併する側　　合併される側

Union Findの説明

・unite

2つのグループを合併



❶: 根

①: それ以外の要素

例: unite(3, 9)

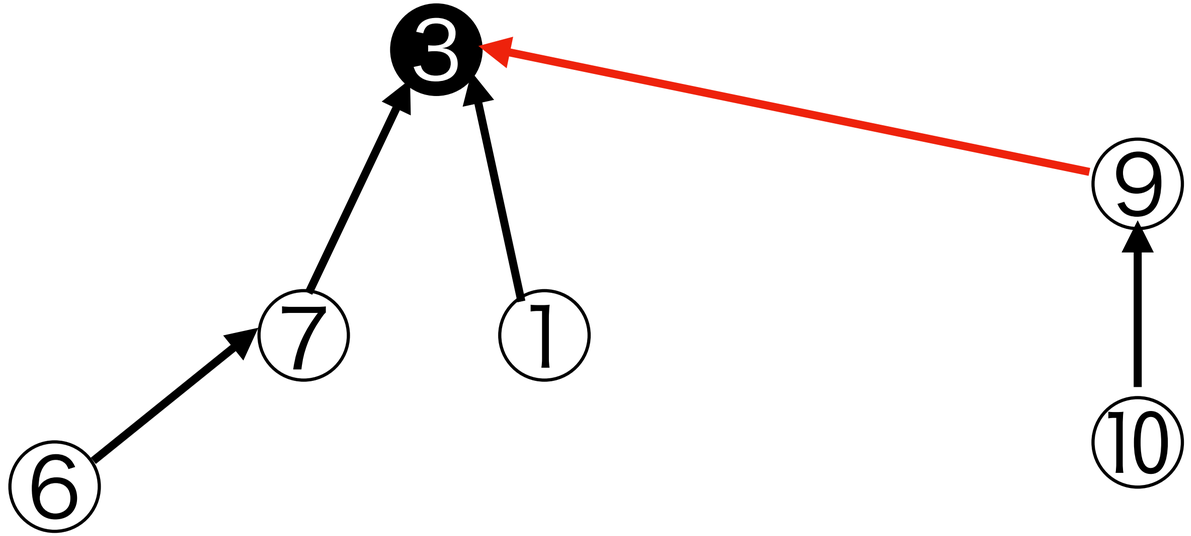
合併される側の根の親を

合併する側の根にする

Union Findの説明

・unite

2つのグループを合併



❶: 根

①: それ以外の要素

例: unite(3, 9)

合併完了!

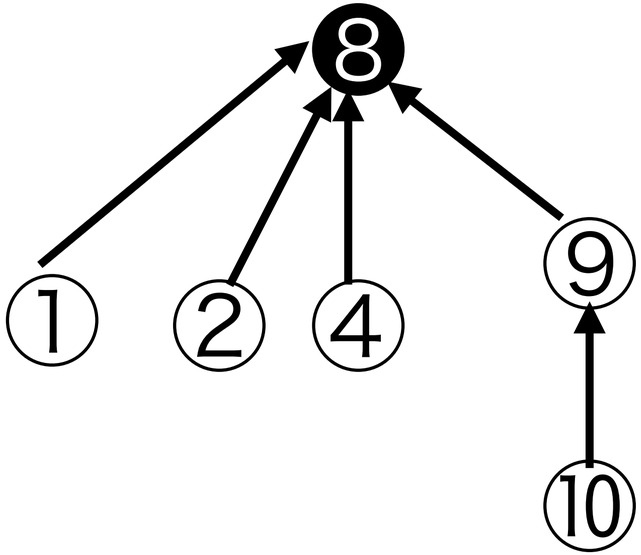
Union Findの説明

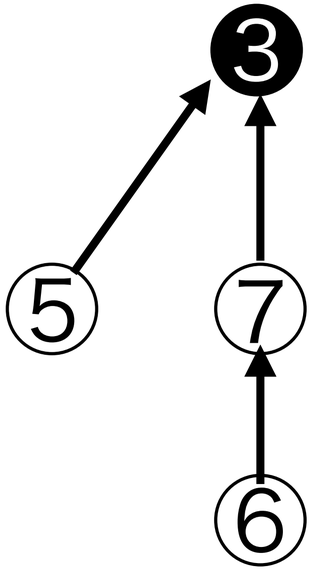
・unite

❶: 根

①: それ以外の要素

例: unite(8, 3)





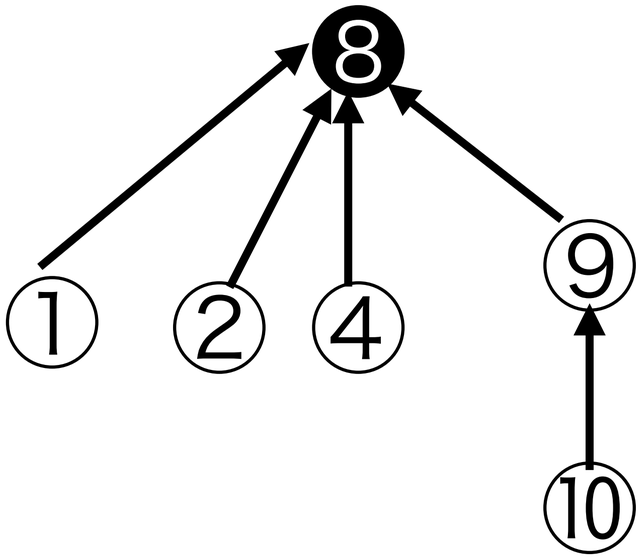
Union Findの説明

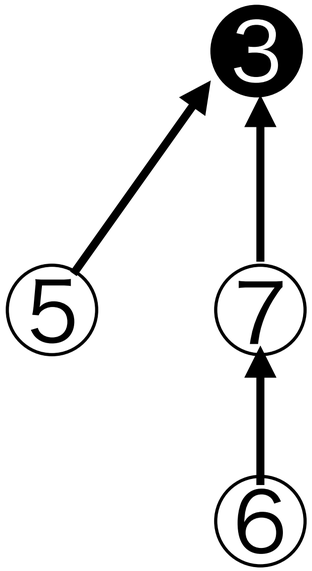
・unite

❶: 根

①: それ以外の要素

例: unite(8, 3)





合併される側の根の親を
合併する側の根にする

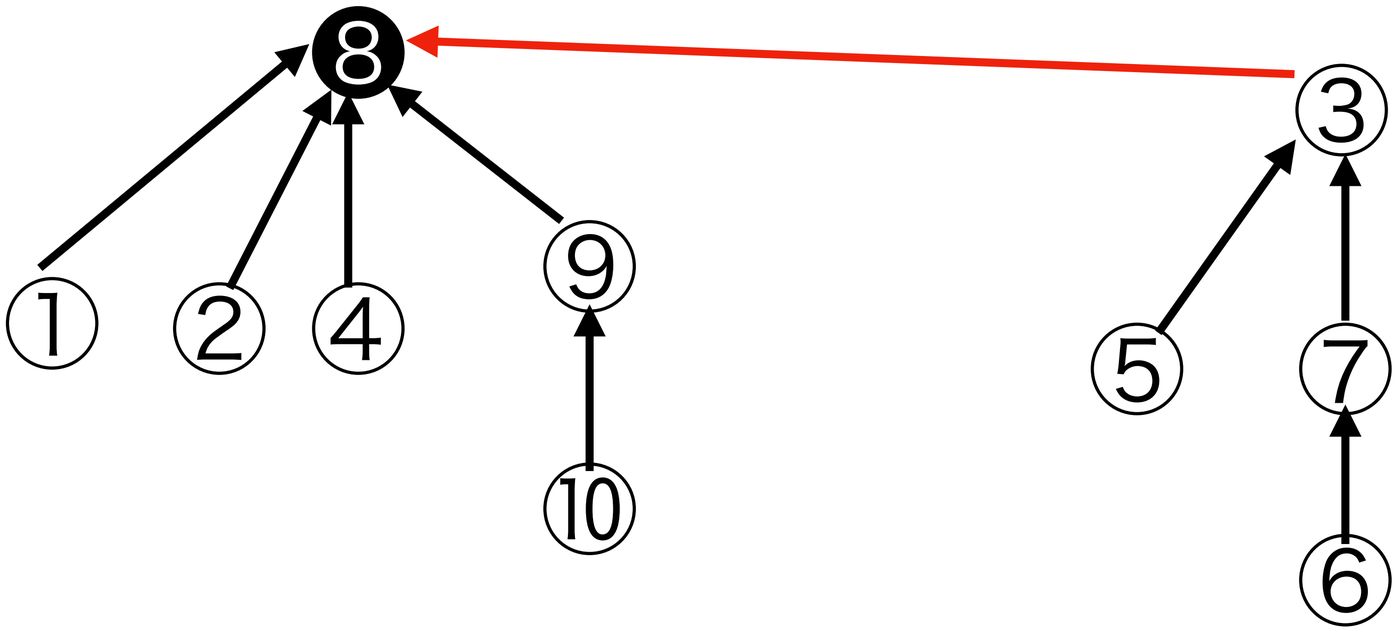
Union Findの説明

・unite

❶: 根

①: それ以外の要素

例: unite(8, 3)

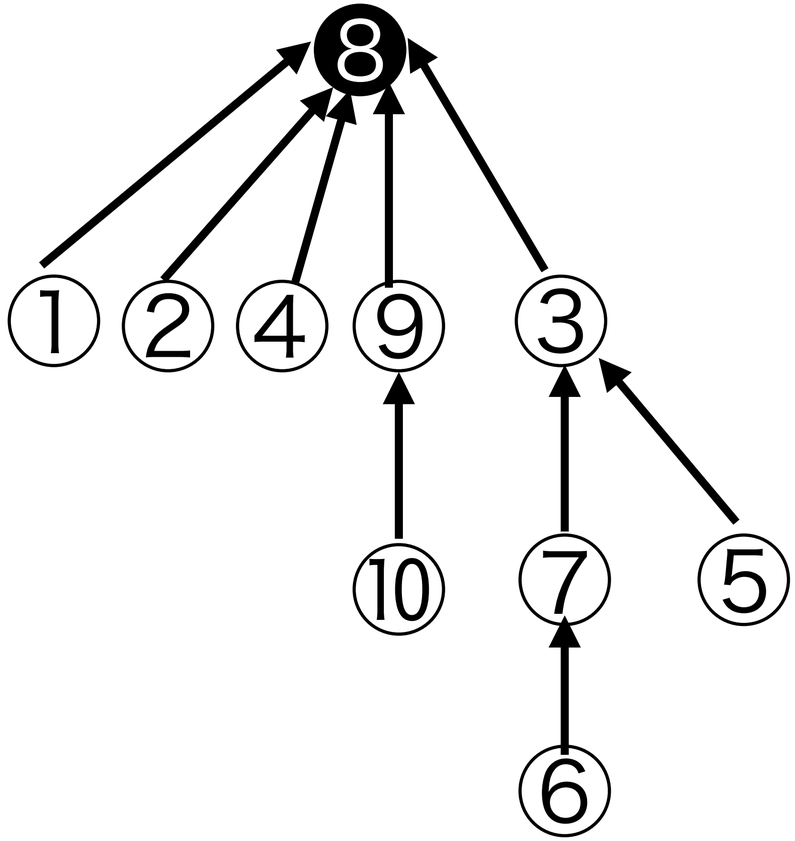


合併完了!

Union Findの説明

・find

要素がどのグループに属しているか調べる



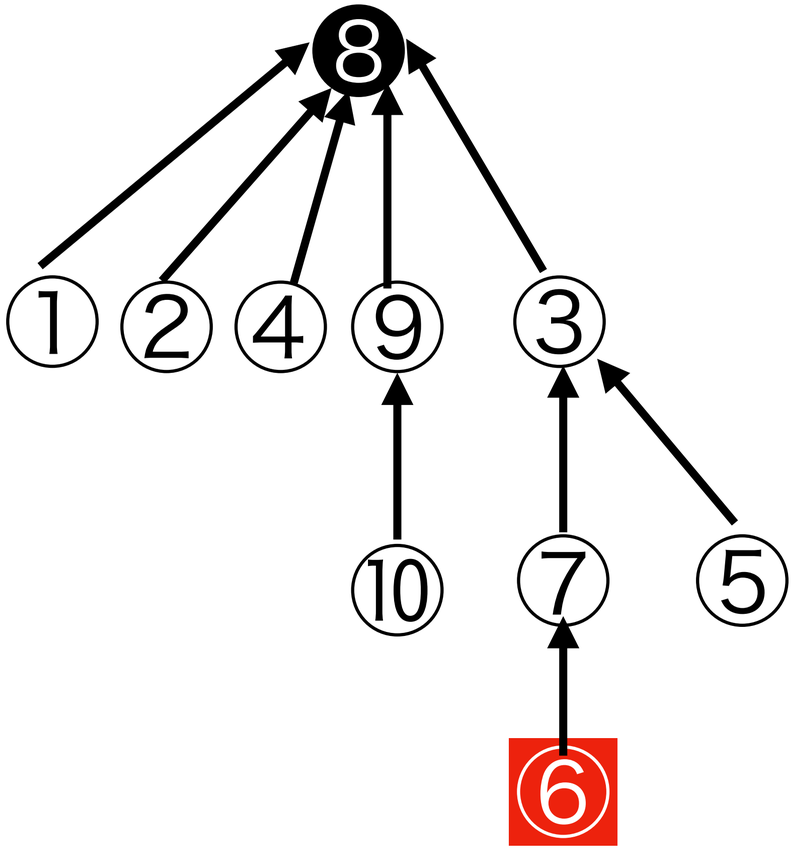
そのためには...

所属している木の

根（代表値）を見つける

Union Findの説明

・find



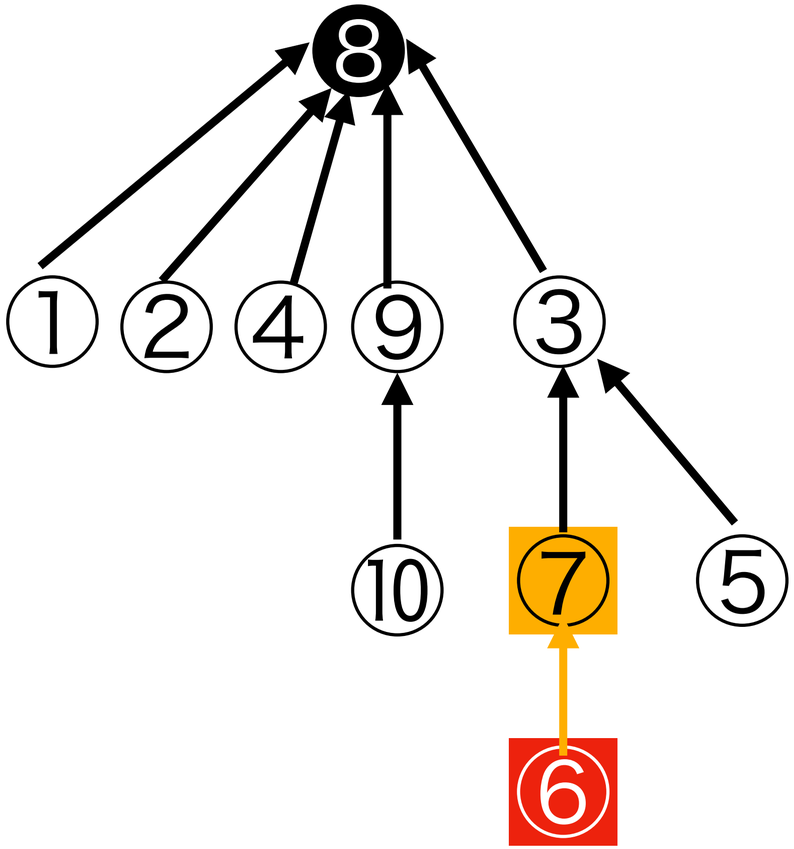
❶: 根

①: それ以外の要素

例: find(6)

Union Findの説明

・find



❶: 根

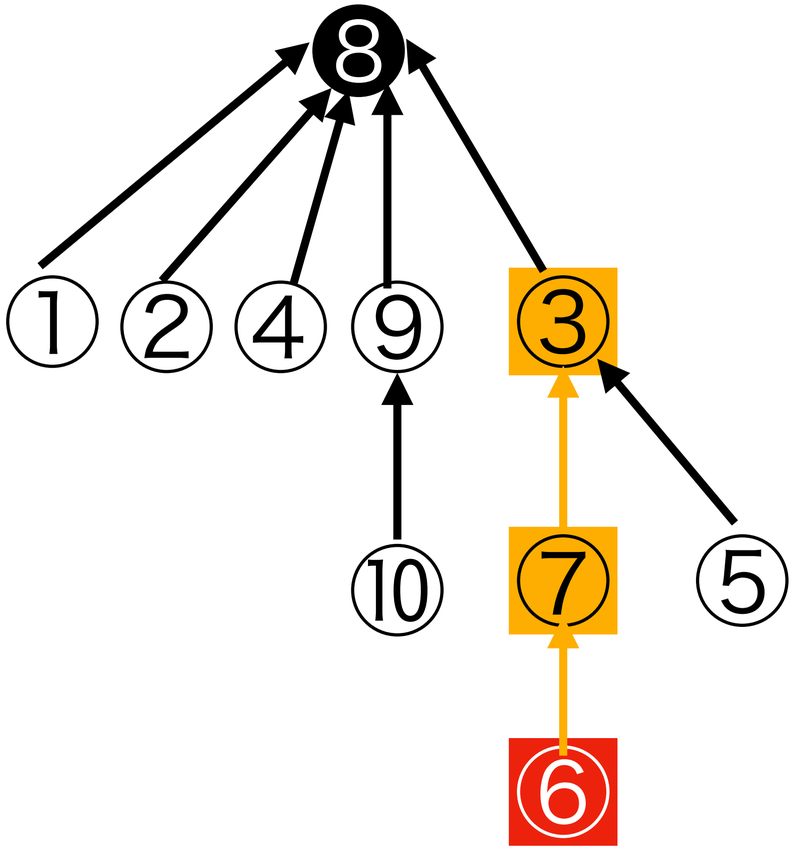
①: それ以外の要素

例: find(6)

・要素6の親を辿る

Union Findの説明

・find



❶: 根

①: それ以外の要素

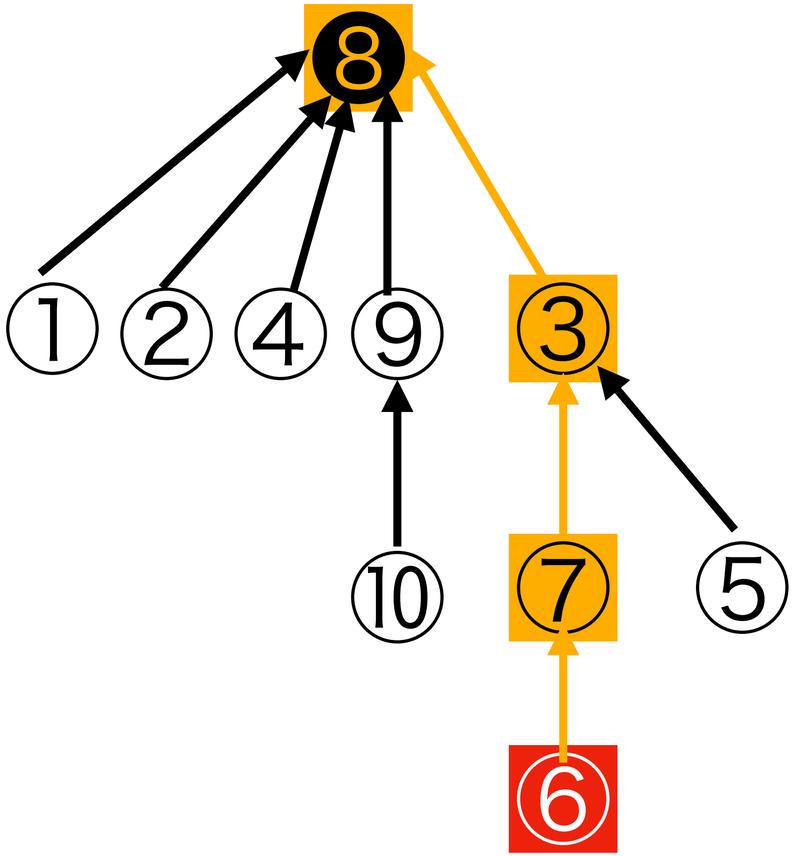
例: find(6)

・要素6の親を辿る

・要素7の親を辿る

Union Findの説明

・find



❶: 根

①: それ以外の要素

例: find(6)

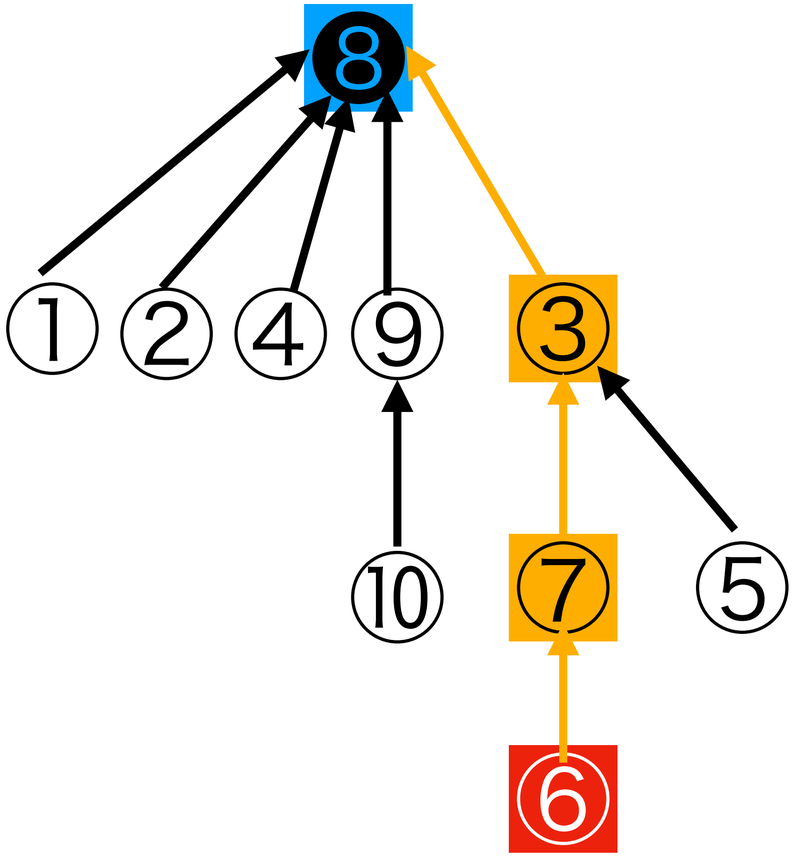
・要素6の親を辿る

・要素7の親を辿る

・要素3の親を辿る

Union Findの説明

・find



❶: 根

①: それ以外の要素

例: find(6)

・要素6の親を辿る

・要素7の親を辿る

・要素3の親を辿る

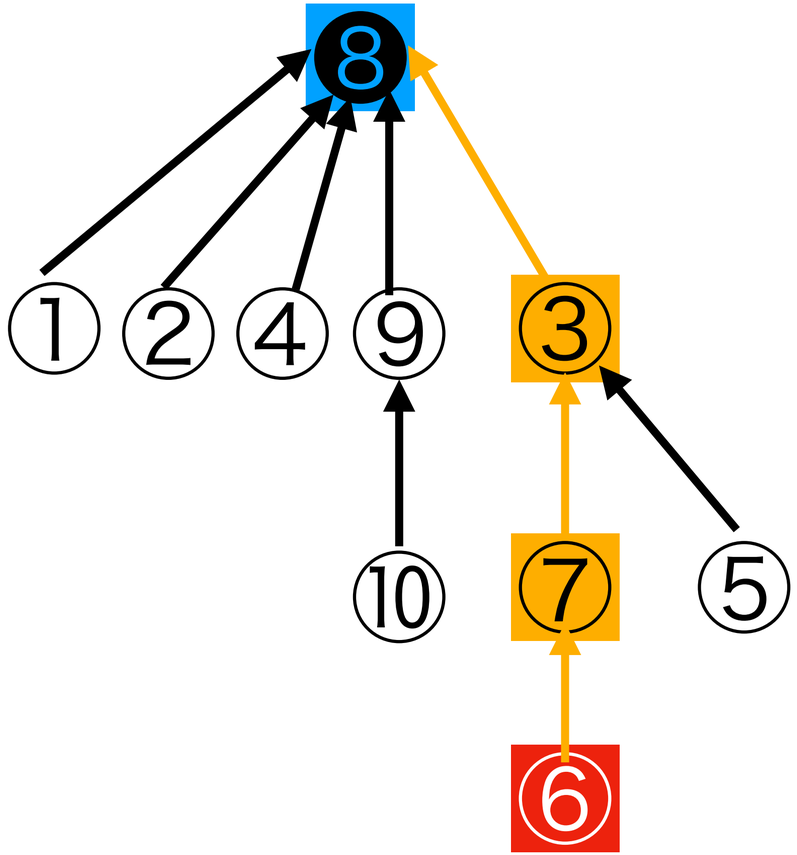
・要素8は親を持たない

=> つまり要素8が根

親を辿ることで根を求められる!

Union Findの説明

・find



つまり要素6は，根が要素8のグループに属している

ある2つの要素についてfindを行う
代表値が同じ → 同じグループ
代表値が違う → 違うグループ

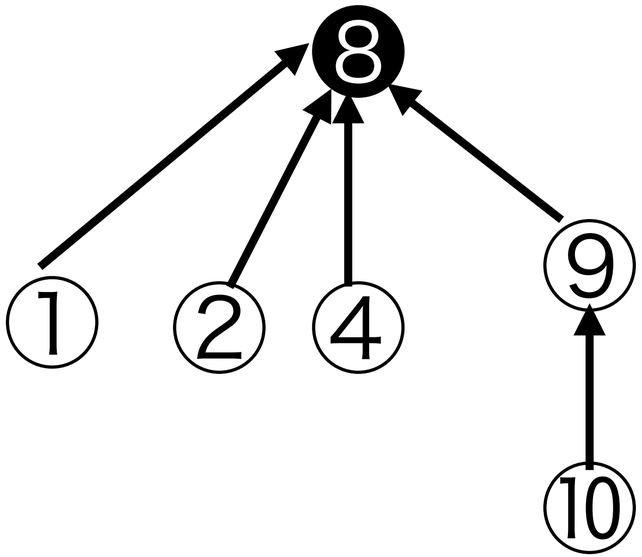
Union Findの説明

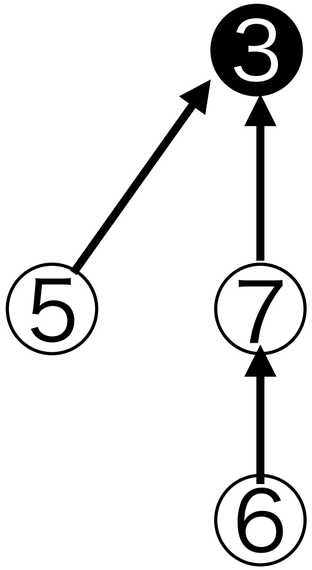
・複雑なunite

❶: 根

①: それ以外の要素

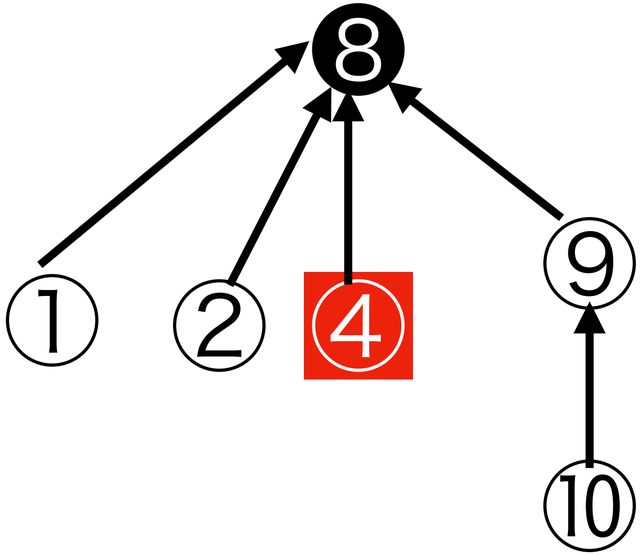
例: unite(4, 6)

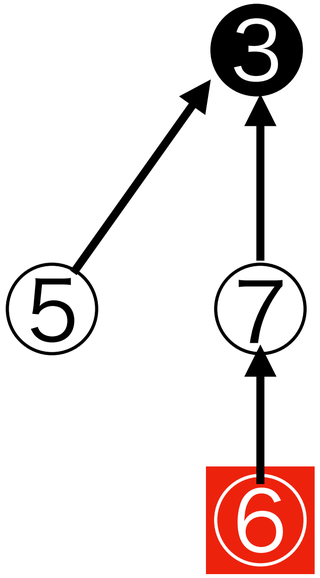




Union Findの説明

・複雑なunite





❶: 根

①: それ以外の要素

例: unite(4, 6)

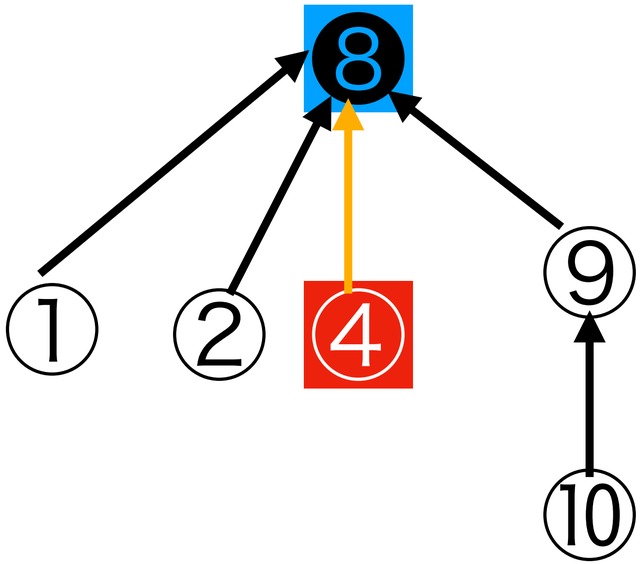
要素4, 6は根ではない...

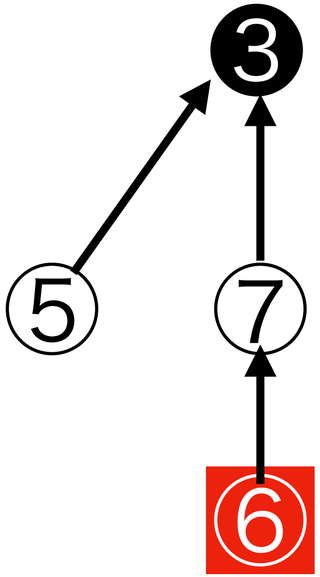
→まずfind(4), find(6)

で根を求める

Union Findの説明

・複雑なunite





❶: 根

①: それ以外の要素

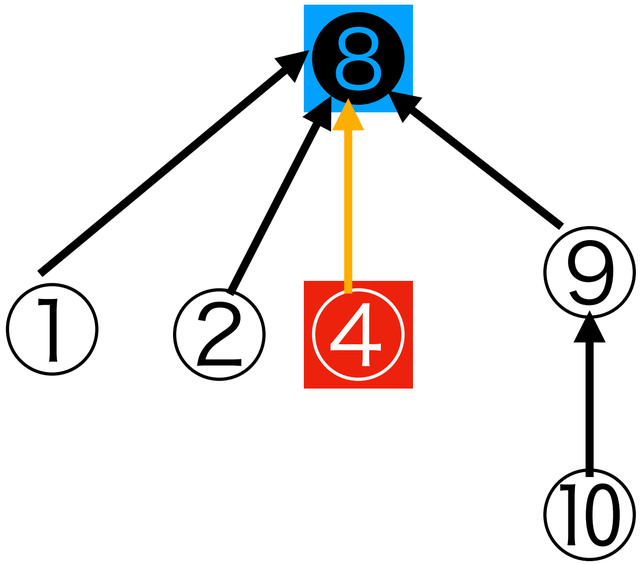
例: unite(4, 6)

find(4)を行う

要素4の根は要素8

Union Findの説明

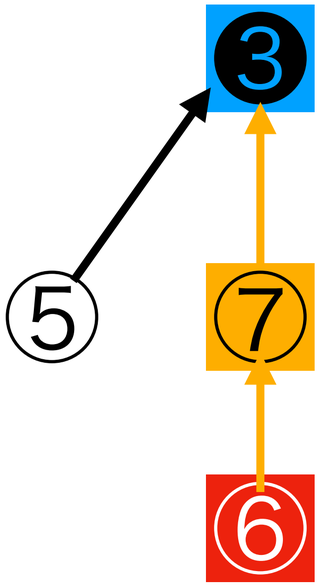
・複雑なunite



❶: 根

①: それ以外の要素

例: unite(4, 6)

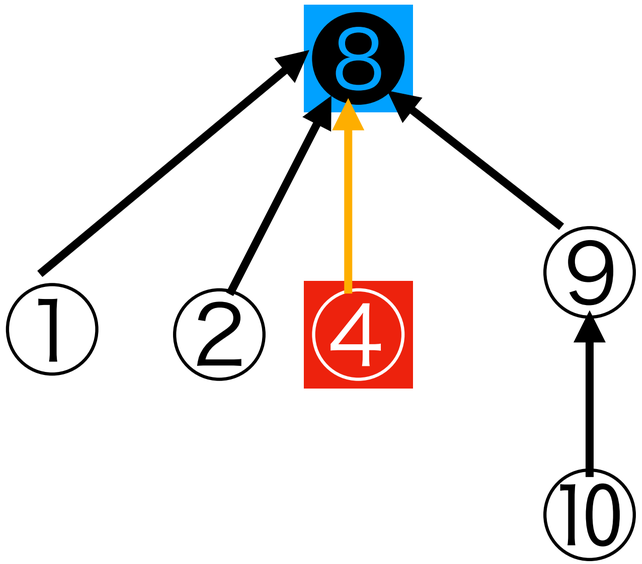


find(6)を行う

要素6の根は要素3

Union Findの説明

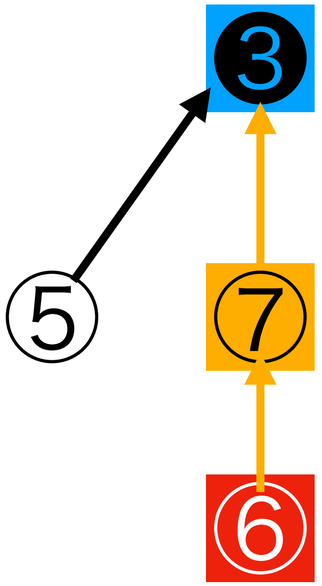
・複雑なunite



❶: 根

①: それ以外の要素

例: unite(4, 6)



根が求められた!

両方の根について
unite(8, 3)を行う

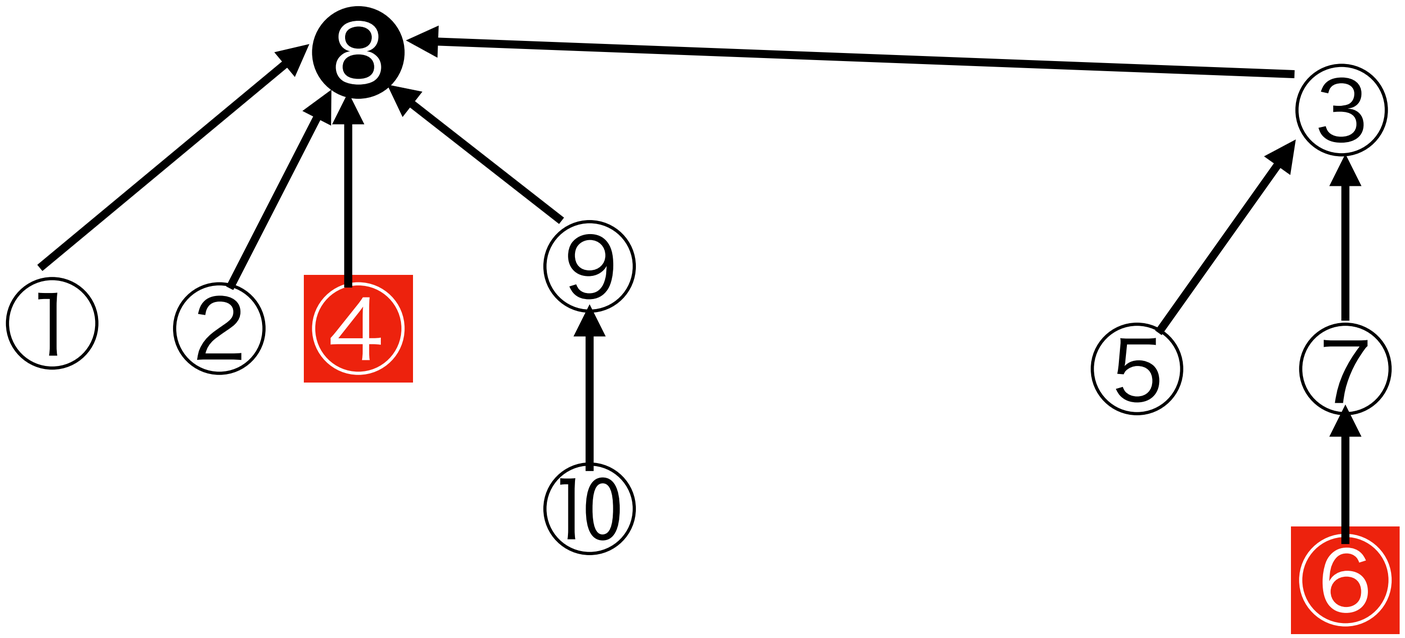
Union Findの説明

・複雑なunite

❶: 根

①: それ以外の要素

例: unite(4, 6)



合併完了!

- Union Findとは
- Union Findの説明
- Union Findの計算量

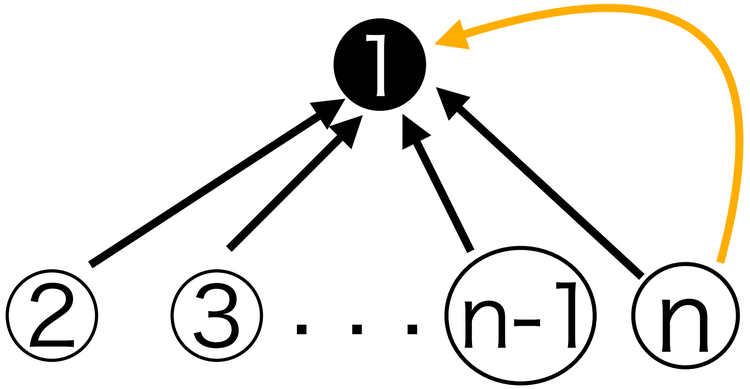
Union Findの計算量

・find　要素数をnとする

一回の探索で済む!

😎計算量が最も良い(小さい)場合

→ 全ての要素の親が根である



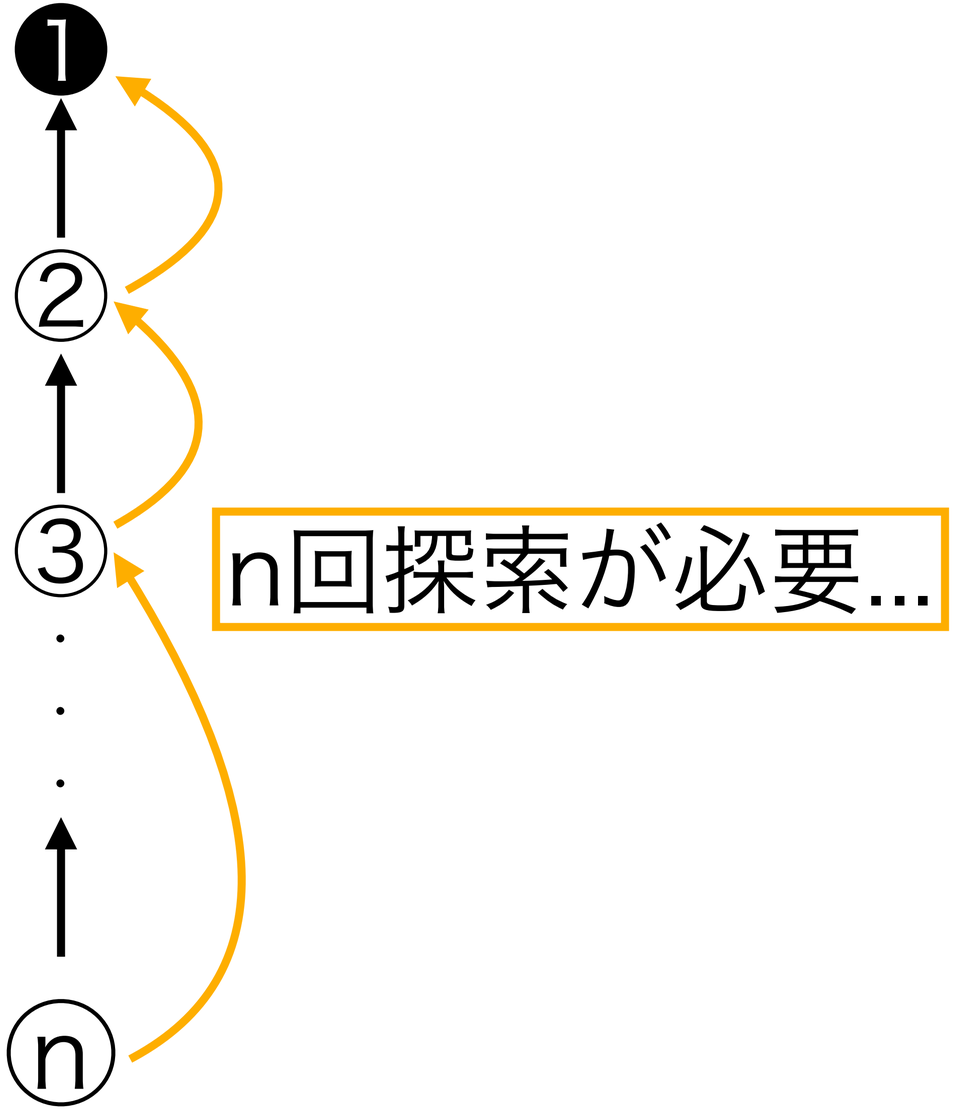
計算量はO(1)

Union Findの計算量

・find　要素数をnとする

😰計算量が最も悪い(大きい)場合

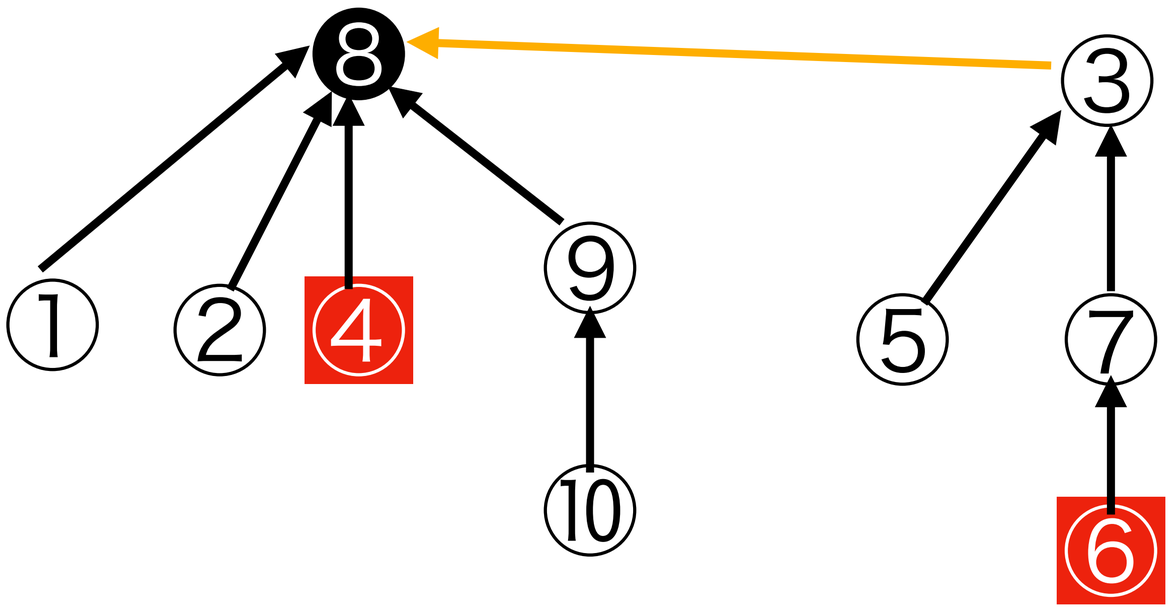
→ 全ての要素が一列に繋がる



計算量はO(n)

Union Findの計算量

・unite　要素数をnとする



合併の処理自体はO(1)
(親を付け替えるだけ)

まず二つの要素の根を求める
つまり2回findを行う
計算量はO(n)

Union Findの計算量

要素数nとすると，unite, findの計算量は...

find: ある要素が所属しているグループの根を求める

つまり根に着くまで，要素の親をさかのぼる

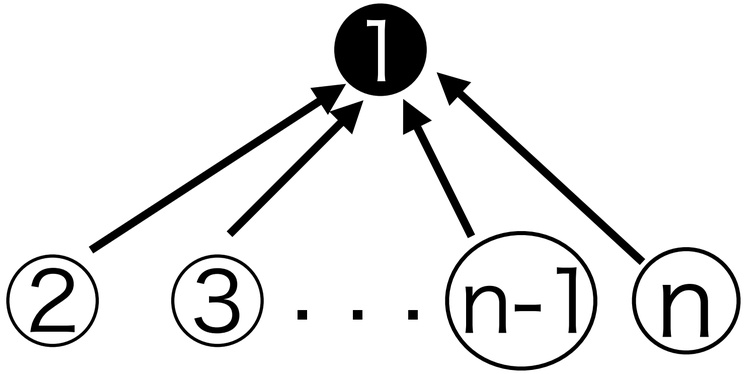
→計算量は最悪の場合O(n)

unite: 二つのグループを合併

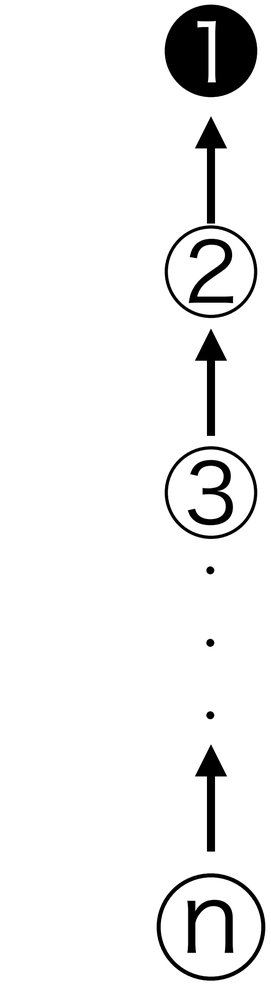
まず二つのグループの根を求めてから合併

→計算量はO(n)（合併の処理自体はO(1)）

Union Findの計算量



find: O(1)



find: O(n)

左の木と右の木では,
unite, findの結果は同じ

しかし計算量では明らかに
左の木の方が効率が良い

→効率の良い木を作りたい!

Union Findの計算量

unite, findの効率化

→経路短縮というテクニックを使う

　findの処理を再帰的に行う際に，木を短縮するという方法

再帰的ってなに...?

→ ある関数の中で，答えが求められるまで

　その関数自身を呼び出し続ける（再帰関数という）

🤔 ??

Union Findの計算量

再帰のイメージ
例: 関数find(4)

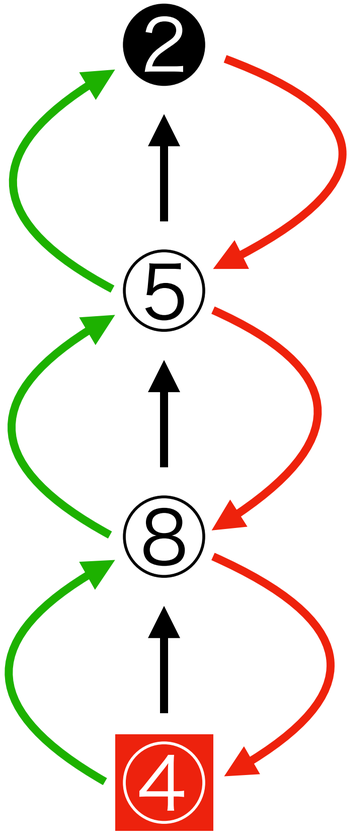
要素の親について,
再帰的に処理を行う

1. 要素4は根ではない
find(4)の中でfind(8)

2. 要素8は根ではない
find(8)の中でfind(5)

3. 要素5は根ではない
find(5)の中でfind(2)

4. 要素2は根!



5. find(2)は
find(5)に根2を返す

6. find(5)は
find(8)に根2を返す

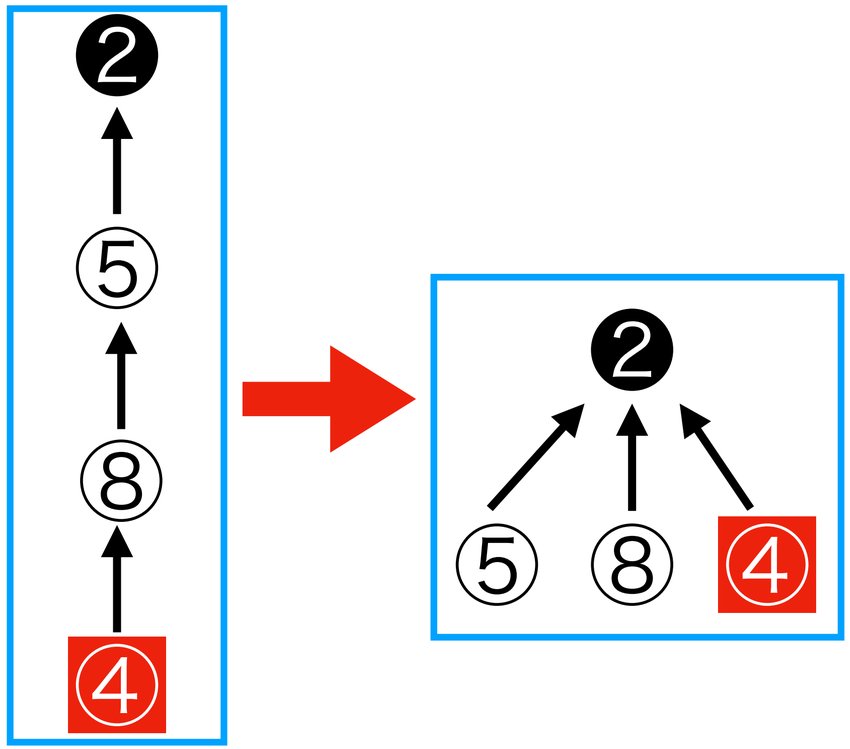
7. find(8)は
find(4)に根2を返す

8. find(4)は根2を返す
→find(4)は2

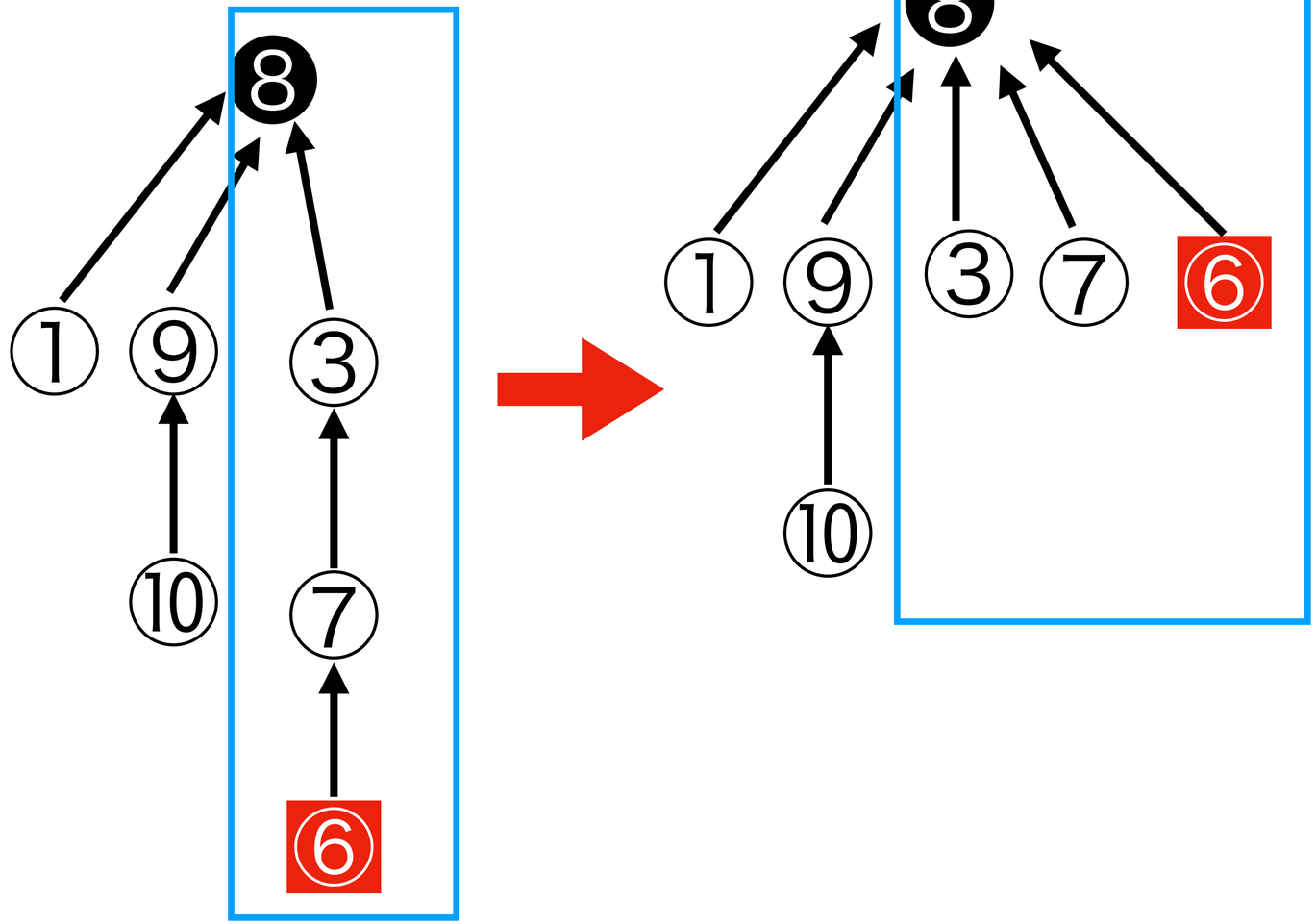
Union Findの計算量

経路短縮のイメージ

例: find(4)



例: find(6)



Union Findの計算量

経路短縮

findの処理を再帰的に行う際に

木を短縮

例: find(4)

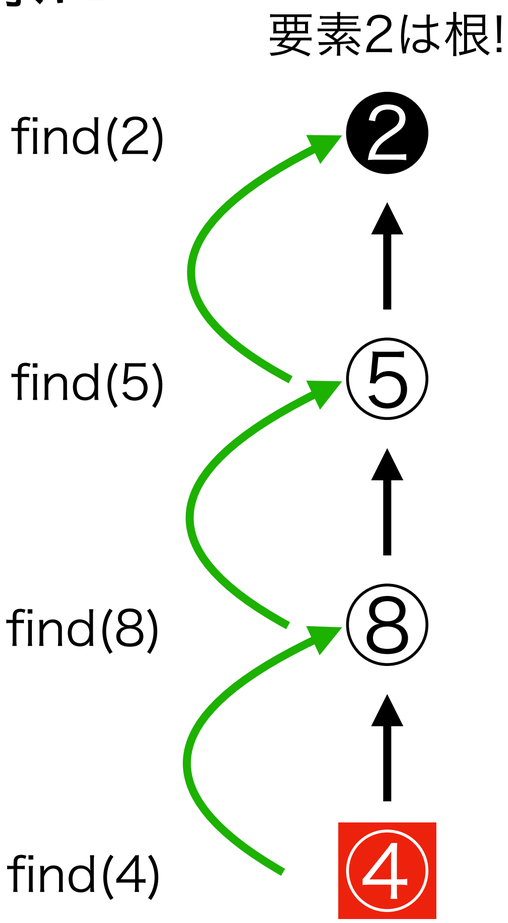
Union Findの計算量

経路短縮

findの処理を再帰的に行う際に

木を短縮

例: find(4)



根を見つけるまでは

通常のfindと同じ

Union Findの計算量

経路短縮

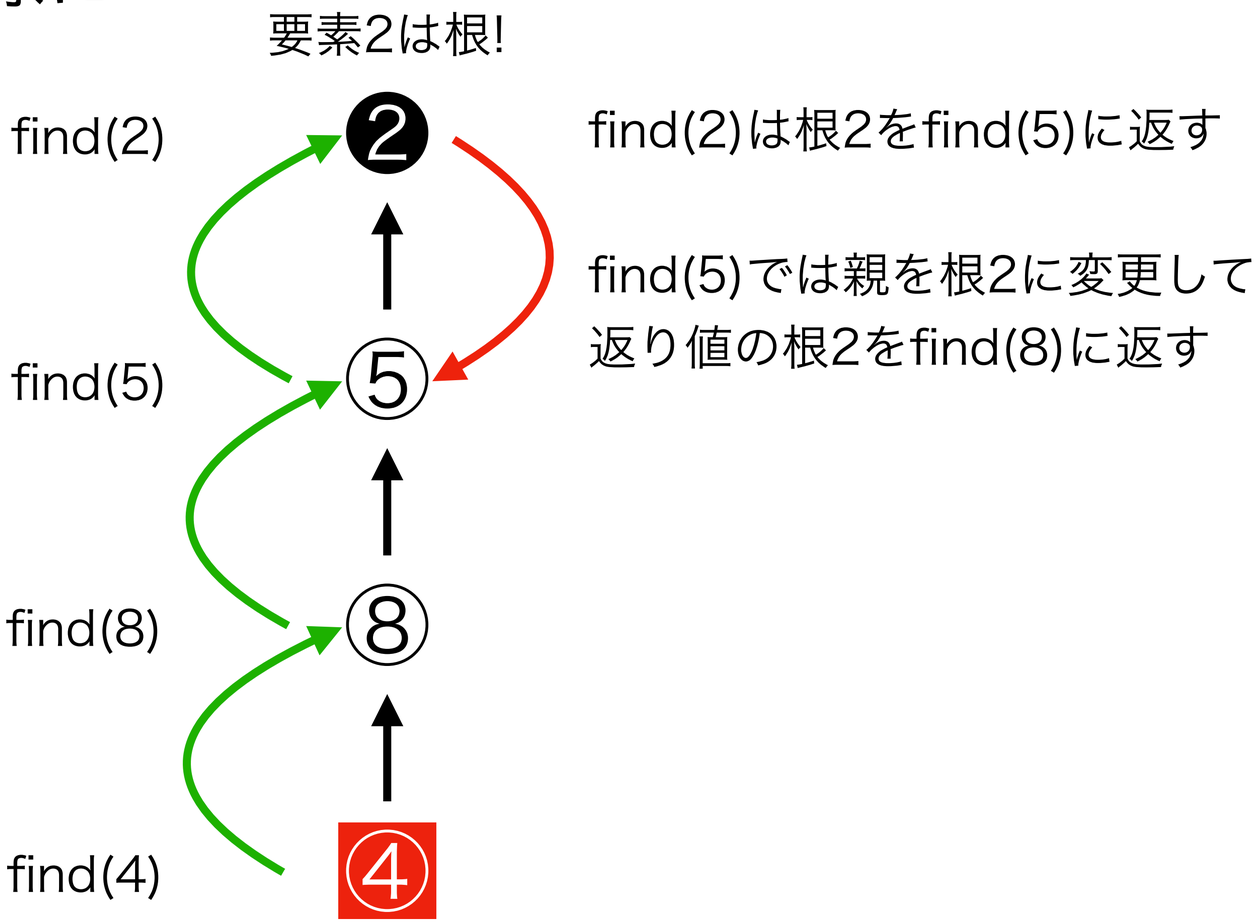
findの処理を再帰的に行う際に

木を短縮

自分の親を根に変更してから

返り値を返す

例: find(4)



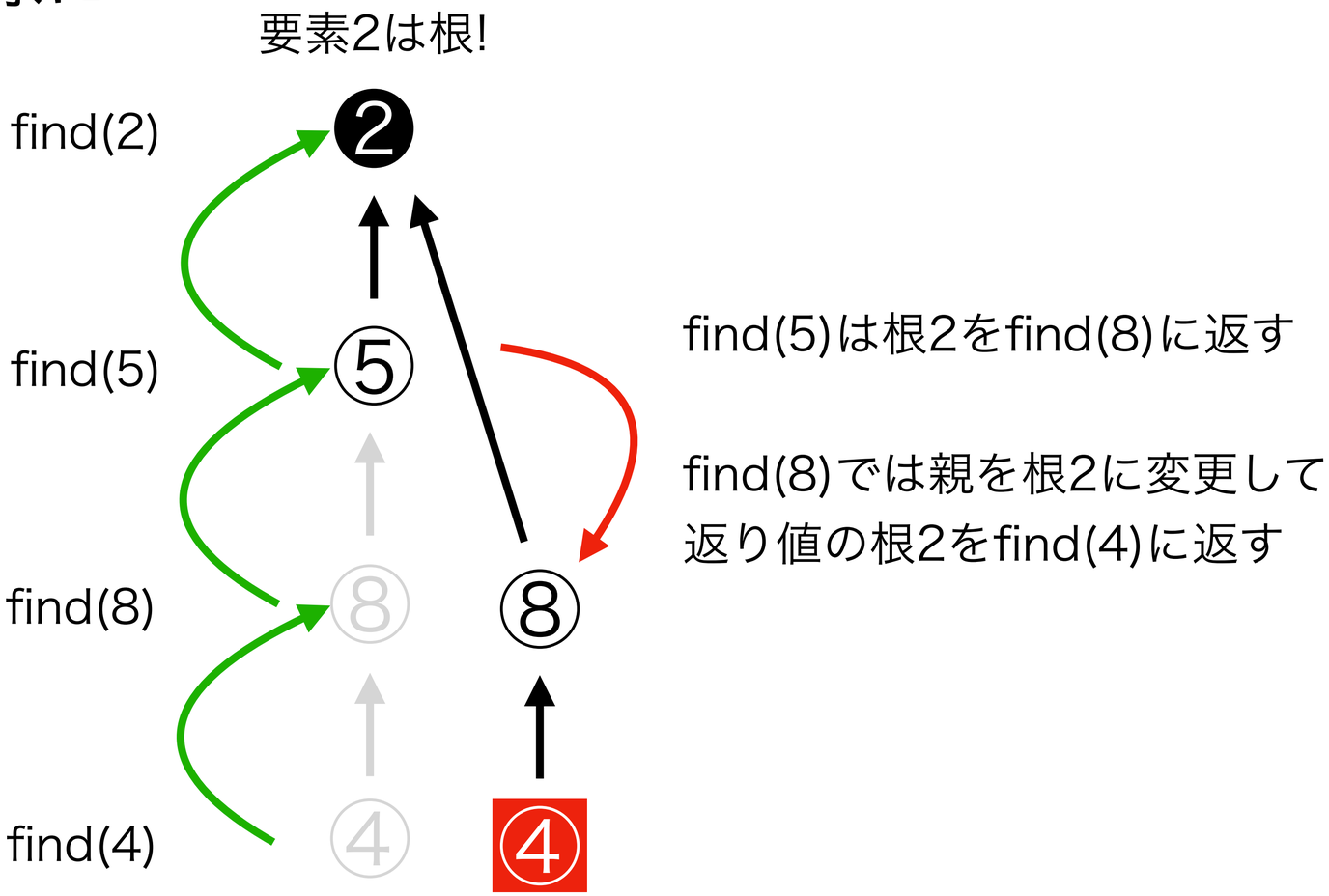
Union Findの計算量

経路短縮

findの処理を再帰的に行う際に
木を短縮

自分の親を根に変更してから
返り値を返す

例: find(4)



Union Findの計算量

経路短縮

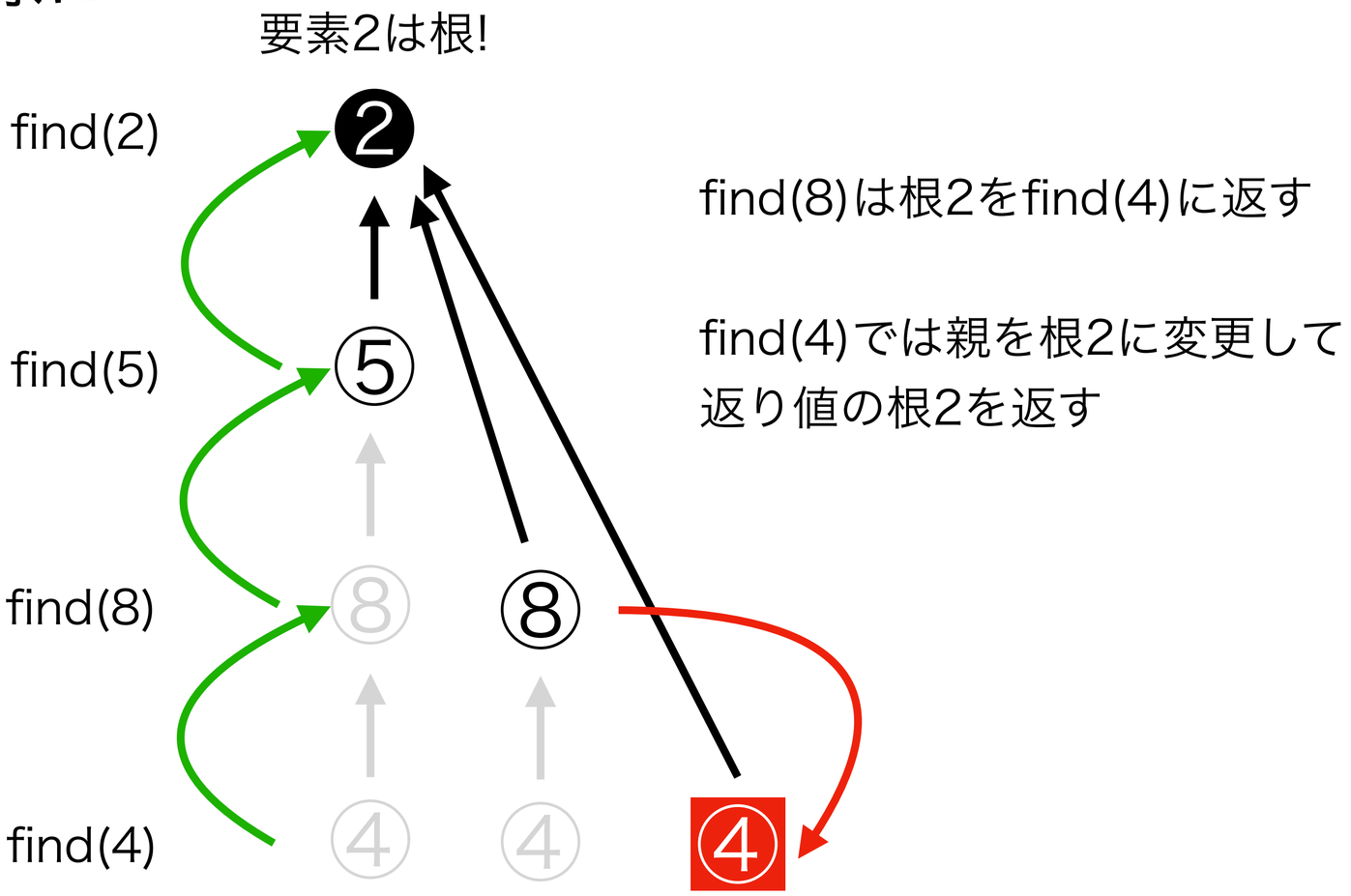
findの処理を再帰的に行う際に

木を短縮

自分の親を根に変更してから

返り値を返す

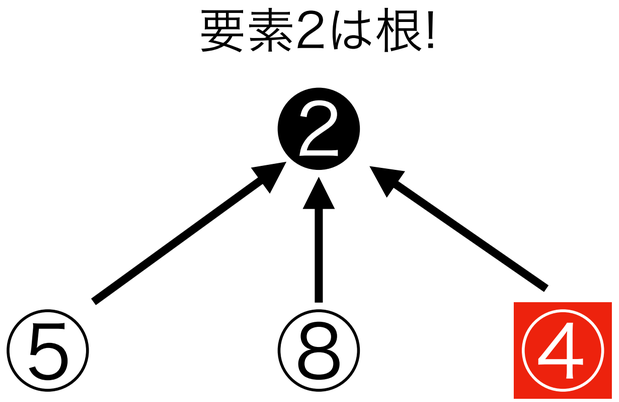
例: find(4)



Union Findの計算量

例: find(4)

経路短縮



find(4)の結果2が得られる

経路短縮によって木の高さを短縮することによって,
計算量が改善される

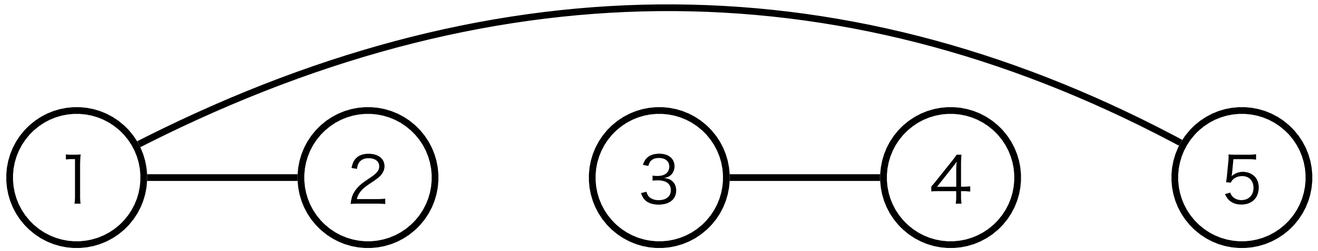
この場合のfindの計算量は大体O(logn)

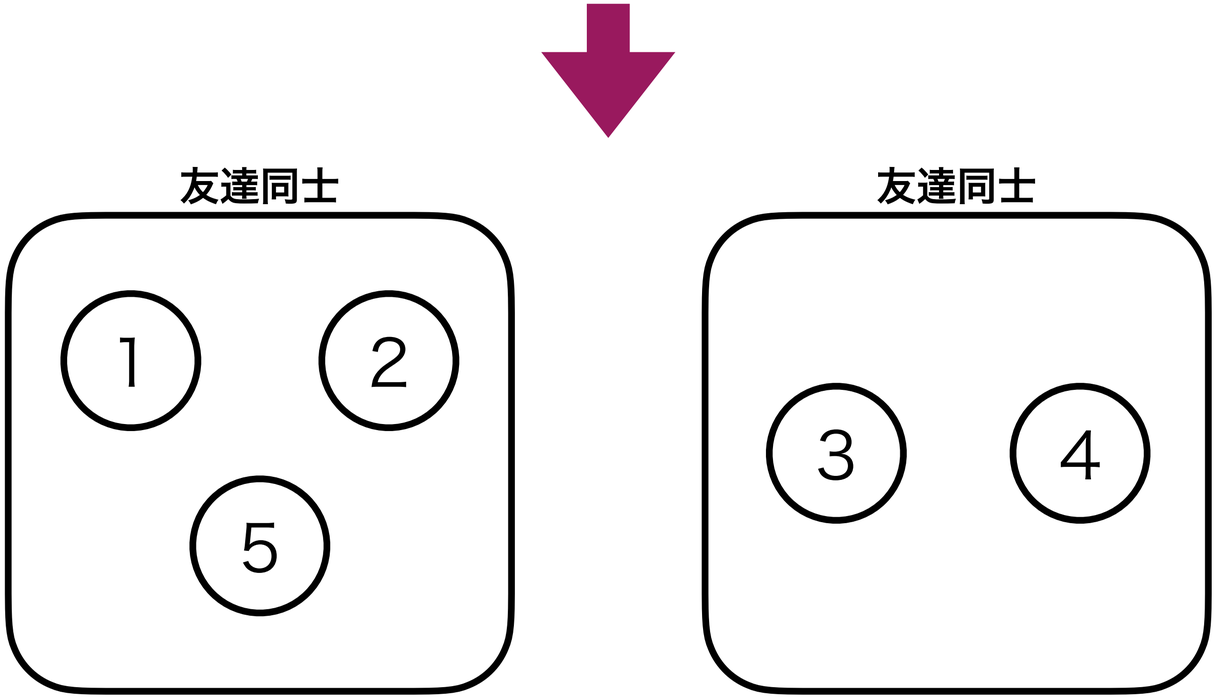
一緒に解いてみよう

- **Friends**を一緒に解いてみよう

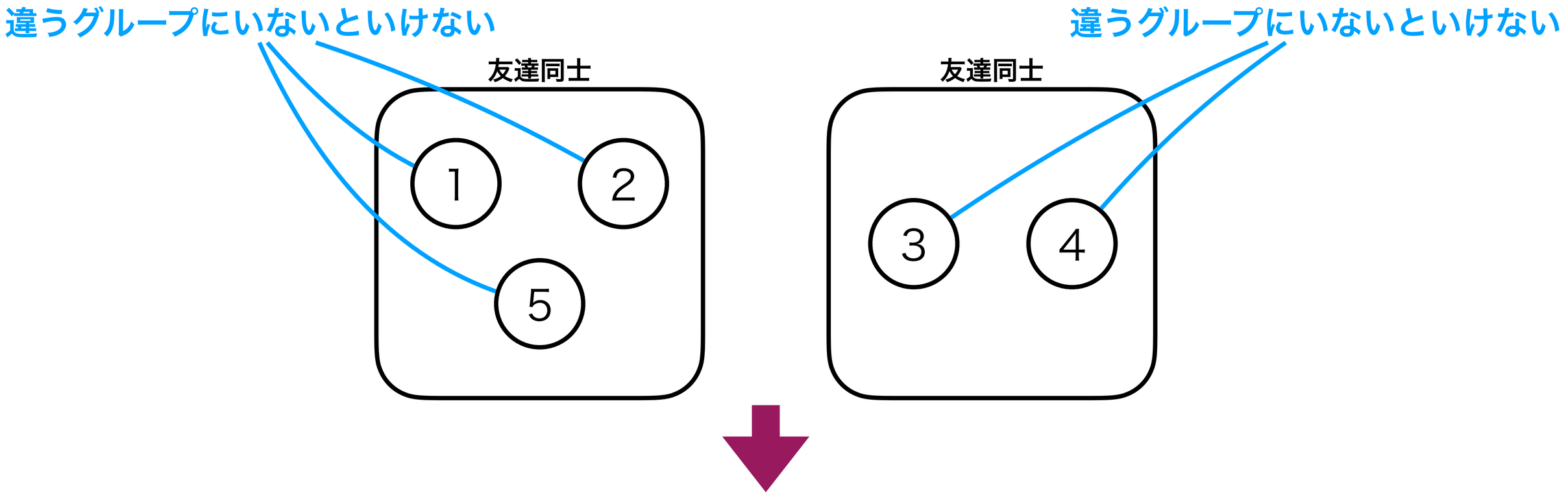
  https://atcoder.jp/contests/abc177/tasks/abc177_d

入力例1を考えてみよう





入力例1を考えてみよう



少なくとも3つのグループが必要

&

4つ以上のグループは必要ない

答えは3

ライブコーディング！

自分で解いてみよう！

- **KAIBUNsyo**を自分で解いてみよう！

  https://atcoder.jp/contests/abc206/tasks/abc206_d