Union Find

参照PDF

UNION Find Atcoer

はじめに

- いいアルゴリズは大きな差を生む
- 良いコードをシンプルにかける
- 計算量がわかれば比較ができるよ
- 最初はナイーブなアプローチを

問題定義

参照リンク

Dynamic connectivity. The input is a sequence of pairs of integers, where each integer represents an object of some type and we are to interpret the pair `p q` as meaning `p` is connected to `q`. We assume that "is connected to" is an *equivalence relation*:

- *symmetric*: If `p` is connected to `q`, then `q` is connected to `p`.
- *transitive*: If `p` is connected to `q` and `q` is connected to `r`, then `p` is connected to `r`.
- *reflexive*: `p` is connected to `p`.

An equivalence relation partitions the objects into *equivalence classes* or *connected components*.

ゴール

数値の組みが同値関係であるかどうか判定する効率的なアルゴリズムを書きたい。

アプリケーション

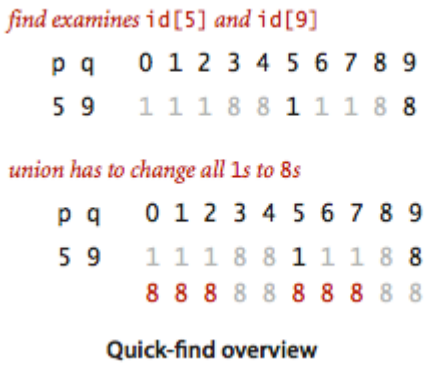
- ネットワーク　（数値がコンピュータを表してる）
- SNS　での友達判定
- 同じ集合かどうか（数学）

API の定義

```
public class UF
        UF(int n)
        void union(int p, int q) // qとpが繋がりを追加
        int find(int p)
        boolean connected(int p, int q) pとqが同値関係か判定するよ
        int count()
```

その１　Quick-Find

- id[p] id[q] が同じかどうかみる
- union するときは　同じやつをすべて変える

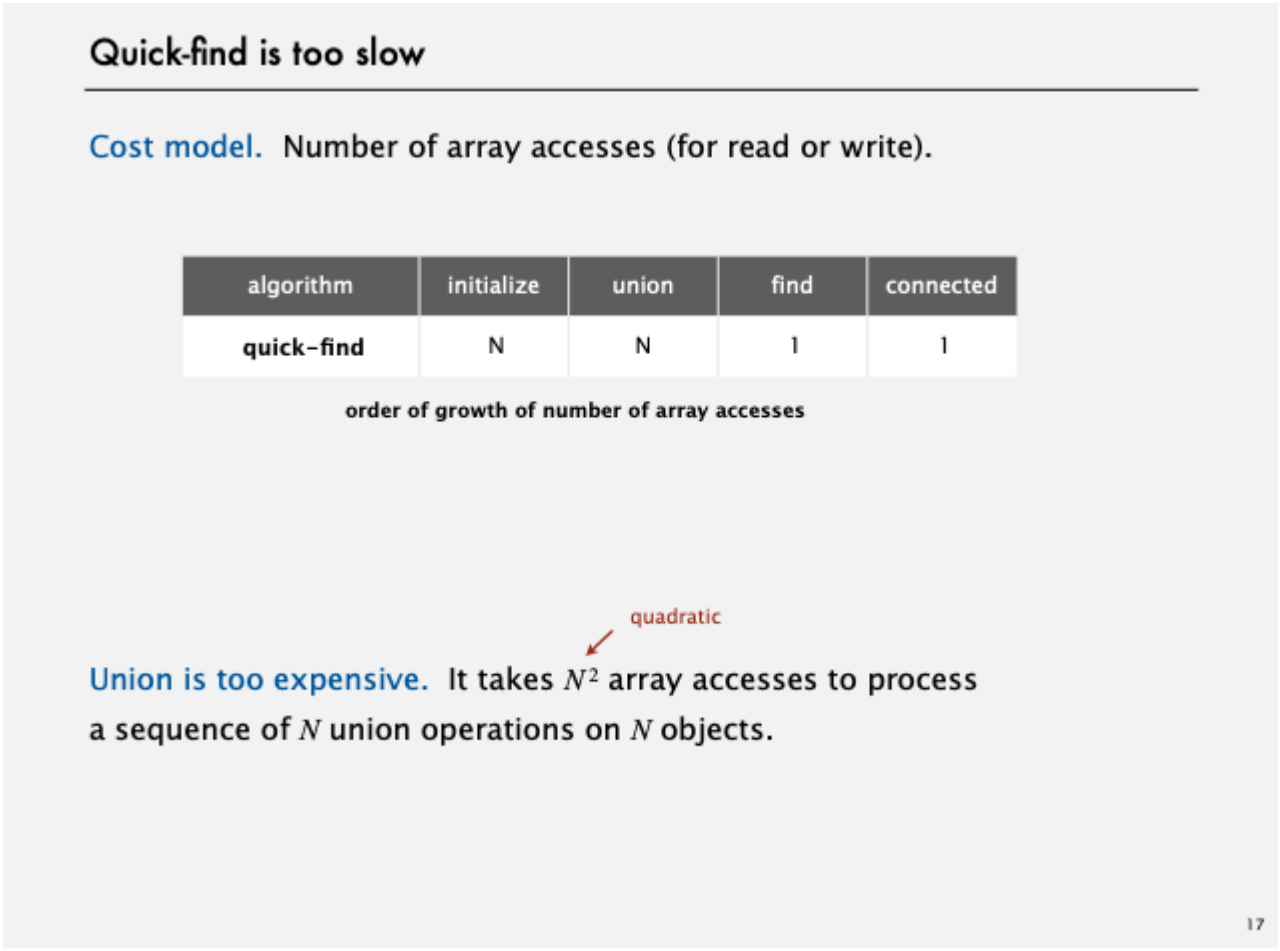


Quick-find overview

demo

一緒に実装してみよう

Performance



余力があれば。僕にはよくわからない証明

Proposition F. The quick-find algorithm uses one array access for each call to `find()` and between $N + 3$ and $2N + 1$ array accesses for each call to `union()` that combines two components.

Proof: Immediate from the code. Each call to `connected()` tests two entries in the `id[]` array, one for each of the two calls to `find()`. Each call to `union()` that combines two components does so by making two calls to `find()`, testing each of the N entries in the `id[]` array, and changing between 1 and $N - 1$ of them.

その2　Union-Find

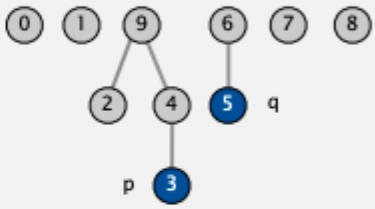
- union がおせえ
- id[p]が親の参照をもつようにしよう（つまり木構造にしてしまおう）

Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



Find. What is the root of p?

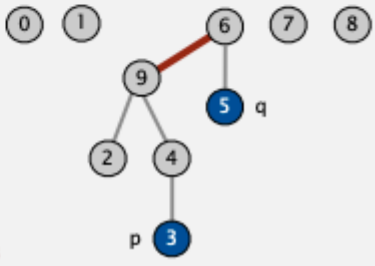
Connected. Do p and q have the same root?

root of 3 is 9
root of 5 is 6
3 and 5 are not connected

Union. To merge components containing p and q, set the id of p's root to the id of q's root.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	6

only one value changes



21

demo

Quick-union is also too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N

← worst case

† includes cost of finding roots

Quick-find defect.

- Union too expensive (N array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

- Trees can get tall.
- Find/connected too expensive (could be N array accesses).

25

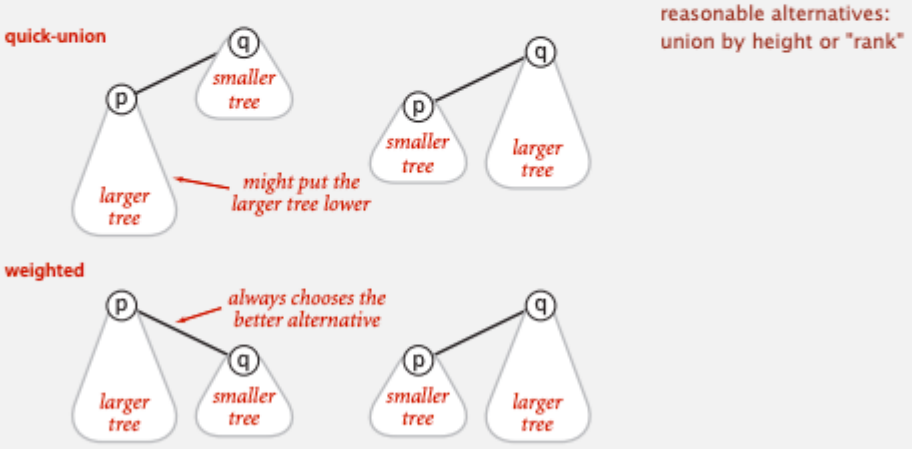
その3 Weighted-Union

Improvement 1: weighting

Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of objects).
- Balance by linking root of smaller tree to root of larger tree.

reasonable alternatives: union by height or "rank"

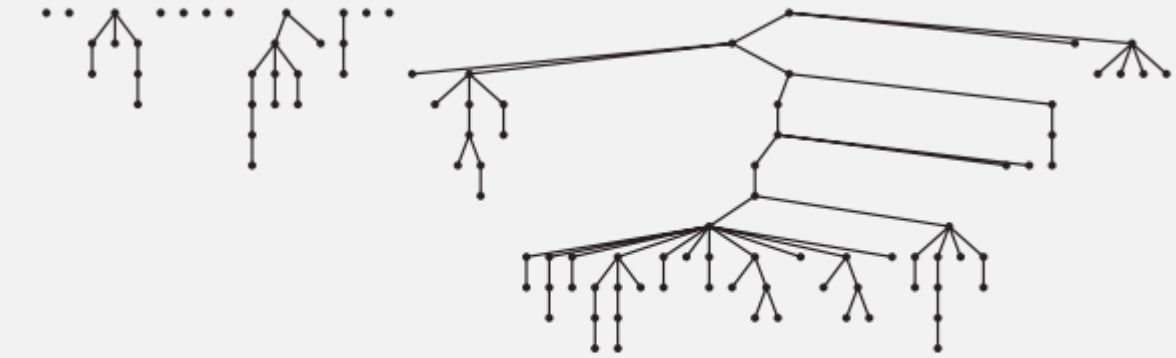


27

Demo

Quick-union and weighted quick-union example

quick-union



average distance to root: 5.11

weighted

average distance to root: 1.52

Quick-union and weighted quick-union (100 sites, 88 `union()` operations)

30

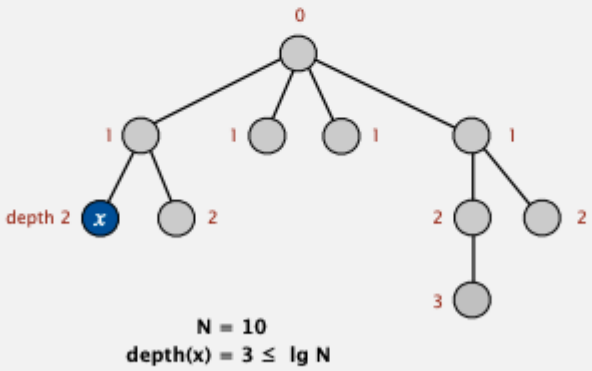
Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p.
- Union: takes constant time, given roots.

lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.



32

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p.
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N
weighted QU	N	lg N †	lg N	lg N

† includes cost of finding roots

Q. Stop at guaranteed acceptable performance?

A. No, easy to improve further.

34

- 余力があれば証明

Definition. The *size* of a tree is its number of nodes. The *depth* of a node in a tree is the number of links on the path from it to the root. The *height* of a tree is the maximum depth among its nodes.

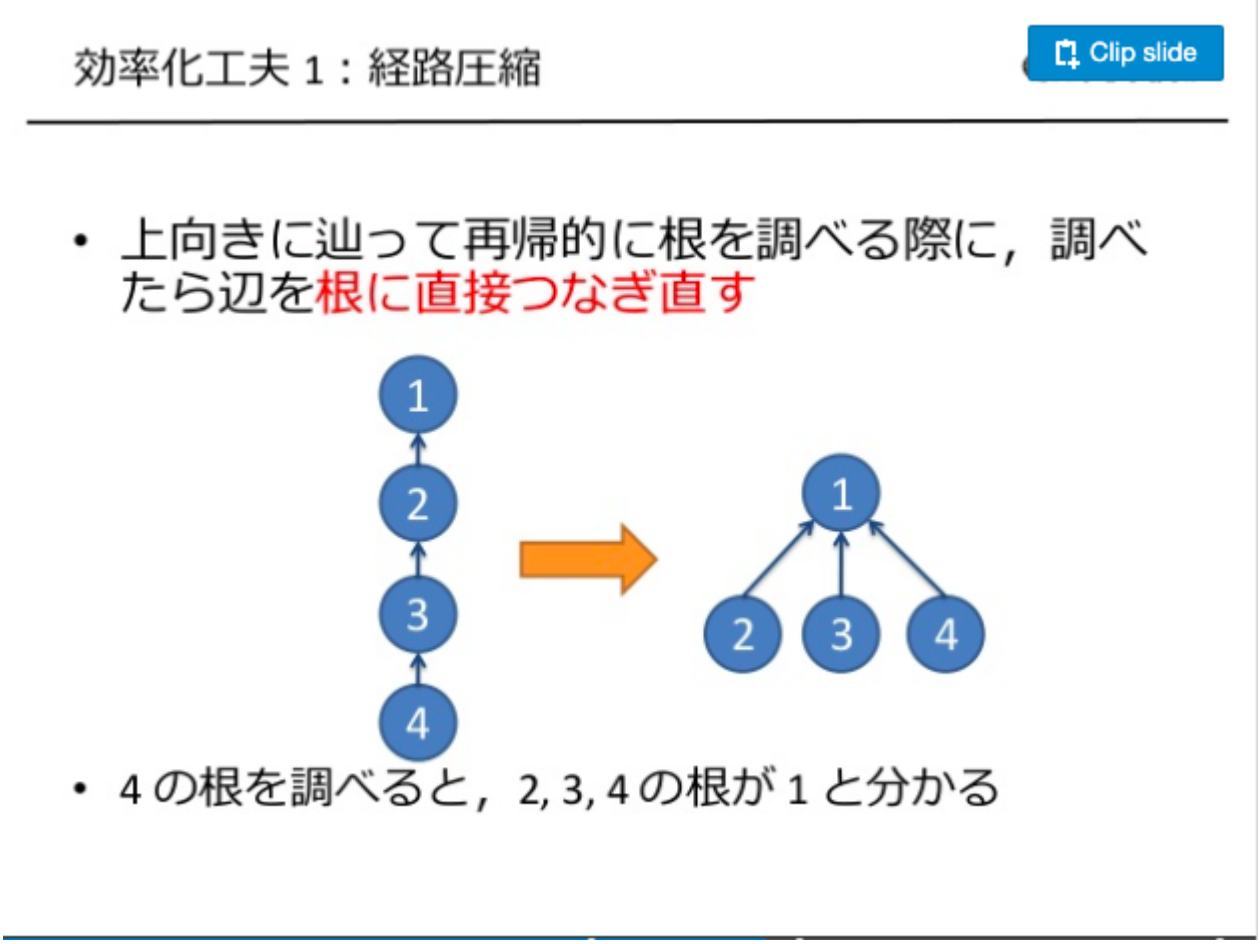
Proposition H. The depth of any node in a forest built by weighted quick-union for N sites is at most lg N.

Proof: We prove a stronger fact by (strong) induction: The height of every tree of size k in the forest is at most lg k. The base case follows from the fact that the tree height is 0 when k is 1. By the inductive hypothesis, assume that the tree height of a tree of size i is at most lg i for all $i < k$. When we combine a tree of size i with a tree of size j with $i \le j$ and $i + j = k$, we increase the depth of each node in the smaller set by 1, but they are now in a tree of size $i + j = k$, so the property is preserved because $1 + \lg i = \lg(i + i) \le \lg(i + j) = \lg k$.

Corollary. For weighted quick-union with N sites, the worst-case order of growth of the cost of `find()`, `connected()`, and `union()` is log N.

Proof. Each operation does at most a constant number of array accesses for each node on the path from a node to a root in the forest.

- 改善 その2



Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes $\leq c(N + M \lg^* N)$ array accesses.

- Analysis can be improved to $N + M\,\alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

iterated lg function

Linear-time algorithm for M union-find ops on N objects?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

Amazing fact. [Fredman-Saks] No linear-time algorithm exists.

in "cell-probe" model of computation

41

まとめ

algorithm	order of growth for N sites (worst case)		
	constructor	union	find
quick-find	N	N	1
quick-union	N	*tree height*	*tree height*
weighted quick-union	N	$\lg N$	$\lg N$
weighted quick-union with path compresson	N	*very, very nearly, but not quite* 1 (*amortized*) (see EXERCISE 1.5.13)	
impossible	N	1	1

Performance characteristics of union-find algorithms

Summary

Key point. Weighted quick union (and/or path compression) makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	M N
quick-union	M N
weighted QU	N + M log N
QU + path compression	N + M log N
weighted QU + path compression	N + M lg* N

order of growth for M union-find operations on a set of N objects

Ex. [10^9 unions and finds with 10^9 objects]

- WQUPC reduces time from 30 years to 6 seconds.
- Supercomputer won't help much; good algorithm enables solution.

42

(Atcorder 問題)(https://www.hamayanhamayan.com/entry/2017/10/04/101826)