

Union Find Disjoint Sets

Robin Jadoul



OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA OLYMPIADE

July 12, 2016

Table of Contents

Disjoint sets

Union Find

Disjoint Sets

What?

- ▶ A collection of sets
- ▶ Not necessarily the programming datastructure
- ▶ No element in multiple sets
- ▶ Operations needed:
 - ▶ union: merge two sets
 - ▶ sameSet: are two elements in the same set

Disjoint Sets

Why?

- ▶ A common example: groups of friends
- ▶ $x \sim y \Leftrightarrow x$ is a friend of y
- ▶ $x \sim y \wedge y \sim z \Rightarrow x \sim z$
- ▶ queries: is x friends with y ?
- ▶ Can be considered as disjoint sets of people
- ▶ People in the same set are friends

Disjoint Sets

How?

- ▶ A vector of `std::sets`
- ▶ problems:
 - ▶ Complicated
 - ▶ Inefficient
 - ▶ Fiddling with both merging of sets, and managing the vector
- ▶ There must be something better
- ▶ Union-Find!

Table of Contents

Disjoint sets

Union Find

Union Find

First try

- ▶ Let's give every set a unique name (number)
- ▶ Have a mapping from element to set name (S)
- ▶ $sameSet(a, b)$
 return $S[a] == S[b]$
- ▶ $union(a, b)$
 $\forall c : \text{if } (S[c] == S[a]) \{ S[c] = S[b]; \}$
- ▶ Better, no need to manage the vector
- ▶ Still slow: need to walk S every union

Union Find

The datastructure

- ▶ We keep the previous idea, but improve it
- ▶ Make it a tree of *parents*, where the root is its own parent
- ▶ extra method: *getParent* walks that tree up
- ▶ *sameSet(a, b)*
 return *getParent(a) == getParent(b)*
- ▶ *union(a, b)*
 if (*!sameSet(a, b)*) *parent[getParent(b)] = getParent(a)*;

Union Find

Improvements

- ▶ *Heuristic by rank*
- ▶ keep extra information: the height (upper bound) of every set/tree
- ▶ Attach the tree with the smallest height to the higher one
- ▶ \Rightarrow Less distance to find the parents

Union Find

Improvements

- ▶ *Path compression*
- ▶ While traversing the tree
- ▶ Make every traversed element a direct child of its root
- ▶ \Rightarrow The height of the trees shrinks considerably

Union Find

Speed

- ▶ $\mathcal{O}(\alpha(n))$ (amortized)
- ▶ $\alpha(n)$ is the inverse *Ackermann function*
- ▶ $\alpha(n) < 4$ for all practical purposes
- ▶ $\mathcal{O}(\alpha(n)) \approx \mathcal{O}(1)$

Union Find

Common additions

- ▶ Keep the number of sets
- ▶ Keep the number of elements for every set
- ▶ Have an extra mapping from T to int if the elements are of type T