Union Find

# Dynamic Connectivity

It automatically establishes ad-hoc network connections between nodes on a network.

Given two nodes, p, and q, in a network, the two nodes have an equivalence relation.

* Reflexive: p is connected to p
* 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

Connect through other nodes. If not connected, do not connect directly.

## Examples

p and q represent two brain regions, and the pair represents a network connection between these two brain regions.

## Terminology

Objects (nodes) are called **sites**, node pairs are **connections,** and the equivalence classes are the connected components.

Components can be represented by one of the sites (nodes)

## Implementation

### UnionFind

In the “Algorithm” book, they considered three different implementations for UnionFind, all based on the array data structure. The book abstracts the *Union* and *Find* functions implementation in a base class called UnionFind. Union and Find are later defined in three implementations ( Quickfind, QuickUnion, and WeightedQuickUnion). In our Python implementation, we mimicked this behavior by declaring a base class *UnionFind* with abstract methods *Find* and *Union*. These methods are implemented in the child classes (Quick Find, Quick Union, and WeightedQuickUnion). Also in our implementation, we separated the implementation from the interface. We added the function connect, which takes a map as input, iterates over the map, and performs the union operation.   
Data Structure  
Arrays: where the index of an array represents a node, and the value on that index is the connected component.

### PseudoCode

# References

## Reading

Algorithm Sedgewick et al. l, Chapter 1. Section 1.5

## Source Code

UnionFind.py

## Demo

#### Toy demo

runUnionFind.py

#### Real demo