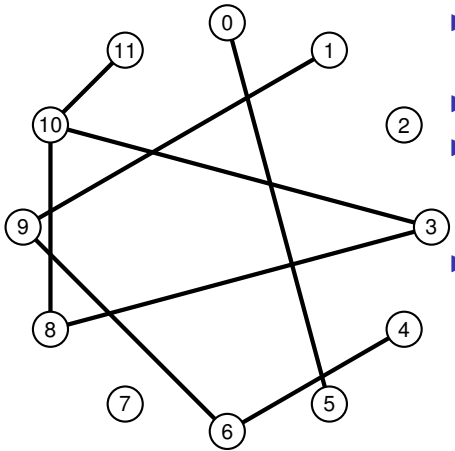
Cosc 201
Algorithms and Data Structures
Lecture 2 (26/2/2025)
Basic union-find

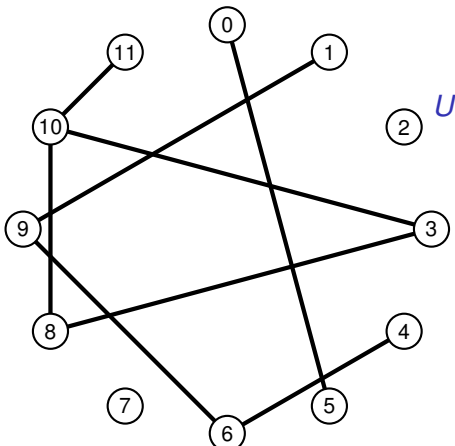
Brendan McCane
brendan.mccane@otago.ac.nz
and
Michael Albert

Disjoint set example



- There are 12 (in general, n) 'objects' (vertices, nodes, circles, dots . . .)
- Some pairs have been connected (by an edge).
- They form groups where two objects are in the same group if there is a sequence of edges between them.
- What are the groups in this example?

Dynamic disjoint set data type



Make(n) Make a set of n vertices with no edges between them.

Union(x, y) Connect x and y by an edge (do nothing if they are already connected by an edge)

Find(x) Find (and return) a representative of the group that x belongs to. If x and y are in the same group then we require that $\mathit{Find}(x) = \mathit{Find}(y)$.

A basic idea

- Use an array to keep track of the representatives associated with each vertex (i.e., the answer to the *Find* query on that vertex)
- Then the `find` method is trivial (just return the array value)
- How does `union` work?

Before and after

Suppose this is the ‘before’ situation:

x	0	1	2	3	4	5	6	7	8
$Find(x)$	4	6	2	6	4	4	6	7	4

What happens after:

- $Union(4, 5)$?
- $Union(1, 5)$?

Before and after

x	0	1	2	3	4	5	6	7	8
$Find(x)$	4	6	2	6	4	4	6	7	4

What happens after:

- $Union(4, 5)$?
  No change because $Find(4) = Find(5)$.
- $Union(1, 5)$?
  Since $Find(1)$ was 6 and $Find(5)$ was 4 we need to change all 6's to 4's or vice versa. For the sake of making a decision we'll always change first to second. So the new result is:

x	0	1	2	3	4	5	6	7	8
$Find(x)$	4	4	2	4	4	4	4	7	4

Union(x, y)

Assumptions: An array, *reps*, contains the representatives of the elements and x and y are valid indices for that array.

Outcome: All entries of *reps* whose value was *reps*$[x]$ have been changed to *reps*$[y]$

```
1: rx ← reps[x]
2: for i from 0 to n − 1 do
3:     if reps[i] = rx then
4:         reps[i] ← reps[y]
5:     end if
6: end for
```

What's wrong with that?

Union(x, y) in Java

```
public void union(int x, int y) {
  int rx = find(x);
  int ry = find(y);
  if (rx == ry) return;
  for (int i = 0; i < reps.length; i++) {
    if (reps[i] == rx) {
      reps[i] = ry;
    }
  }
}
```

One minor improvement is to check whether there is no need for a union – which avoids the loop in that case. Also, for readability, a call to `find` is used rather than just `rx = reps[x]`.

What do the operations cost?

- Remember (from COMP162) that when we are thinking about the cost of operations we consider only the worst case, and we ignore (multiplicative) constants.
- We also like to ignore small terms, since they have even less effect than multiplying by a constant does.
- Big-O notation captures **upper bounds** on the cost of an algorithm/method in terms of a measure (almost universally denoted n) of the size of the input.
- We're going to introduce a new player, that gives a bit more precision and avoids having to say things like "best possible O-estimate".

Introducing big-Θ

Recall the definition of $f(n) = O(g(n))$:

$f(n) = O(g(n))$ *if there is some constant* $A > 0$ *such that for all sufficiently large n,*

$$f(n) \leqslant A \times g(n).$$

The definition of $f(n) = \Theta(g(n))$ is similar:

$f(n) = \Theta(g(n))$ *if there are constants* $0 < B < A$ *such that for all sufficiently large n,*

$$B \times g(n) \leqslant f(n) \leqslant A \times g(n).$$

That is, O says essentially that $g(n)$ provides an upper bound for $f(n)$ while Θ says that it provides both a lower and an upper bound – always up to ignoring multiplication by a constant.

Examples

Which of the following are true (and why)?

- $10n^2 = \Theta(n^2)$
- $n^2 = O(2^n)$
- $n^2 = \Theta(2^n)$
- $n^2 = \Theta(10n^2 + 3n)$

Observations

- It's correct to think of $f(n) = \Theta(g(n))$ as saying *f and g have similar rates of growth*.
- Indeed, if $f(n) = \Theta(g(n))$ then also $g(n) = \Theta(f(n))$.
- And in turn, this is the same as $f(n) = O(g(n))$ and $g(n) = O(f(n))$.
- Normal usage in $f(n) = \Theta(g(n))$ is for g to be some very simple expression while f might be more complex (or not completely known). So

  $$10n^2 + 3n = \Theta(n^2)$$

  looks fine, while

  $$n^2 = \Theta(10n^2 + 3n)$$

  looks odd, even though both are true.

And what about Union-Find?

For our basic array-based implementation the cost of the three operations is as follows:

Operation	Cost	Reason
$Make(n)$	$\Theta(n)$	Allocation and initialisation of n places in memory takes constant time per place.
$Find(x)$	$\Theta(1)$	Looking up a value in an array takes constant time.
$Union(x, y)$	$O(n)$	The whole array needs to be examined to change representatives.

Next time we will try to see if we can improve the behaviour of *Union* and what that might cost us elsewhere.

Worst case costs for the basic implementation

Operation	Cost	Reason
$Make(n)$	$\Theta(n)$	Allocation and initialisation of n places in memory takes constant time per place.
$Find(x)$	$\Theta(1)$	Looking up a value in an array takes constant time.
$Union(x, y)$	$O(n)$	The whole array needs to be examined to change representatives.

- $Union(x, y)$ has constant cost if x and y are already in the same set, but has linear cost if they are in different sets.
- The cost of *Union* is a problem because we might call it up to $n - 1$ times with pairs in different sets.
- The total cost then will be $\Theta(n^2)$ which is too much to handle cases which are truly large. Can we do better?

An idea

- For *Find* to work, every element needs to be able to identify their representative.
- But this doesn't have to be in one step.
- "Who's your representative?"
- "It's either that one, or whoever their representative is."
- Then *Union*(x, y) can be:
  - Find the representatives *rx* of *x* and *ry* of *y*.
  - Make *ry* the representative of *rx*.
- Remember not to try to write any code before working out some examples by hand!

An example

Make(9)
Union(6, 8)
Union(3, 4)
Union(7, 1)
Union(7, 4)
Union(1, 8)