Cosc 201
Algorithms and Data Structures
Lecture 4 (5/3/2024)
Induction I

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

UNIVERSITY OF OTAGO
Te Whare Wānanga o Otāgo
NEW ZEALAND

Recap

We found that by maintaining ‘local representatives’ we could ensure that the time required by *Union* could be some constant times the time required by *Find*.

- For a representative, let its rank be the length of the longest chain of ‘local representatives’ that reaches it.
- When forming a union - consider the two representatives involved. If one of them is larger rank, make it the local (and global) representative of the other.
- If they are of equal rank, make the second the representative of the first.
- Then the maximum of the two ranks can increase only if they were equal to begin with.

Recap: Java for third version of union-find

```
public void make(int n) {
  reps = new int[n];
  rank = new int[n];
  for (int i = 0; i < n; i++) {
    reps[i] = i;
    rank[i] = 0;
  }
  groups = n;
}

public int find(int x) {
  while (x != reps[x]) {
    x = reps[x];
  }
  return x;
}
```

```
public void union(int x, int y) {
  rootUnion(find(x), find(y));
}

private void rootUnion(int x, int y) {
  if (x == y) {
    return;
  }
  groups--;
  if (rank[x] > rank[y]) {
    reps[y] = x;
  } else if (rank[y] > rank[x]) {
    reps[x] = y;
  } else { // ranks are equal
    reps[x] = y;
    rank[y]++;
  }
}
```

Minimum size of a set of rank k

Question: *If a set has a representative of rank k how large must it be?*

- For $k = 0$ we're talking about a 'bare point', so it must have size at least 1.
- For $k = 1$ the representative must have at least one child, so its set has size at least 2.
- For larger k – how did that set get formed? It must have been formed by the union of two sets of rank $k - 1$. So its size must be at least twice the minimum size of a set of rank $k - 1$.
- That gives 1, 2, 4, 8, 16 . . . – an obvious pattern.

Our first theorem

A set of rank k must contain at least 2^k elements.

Now we want to prove this formally. To do that, we introduce a new technique: induction. I want to recap the argument in detail to show how it works.

The ingredients

- Initially, every element is its own representative and every element has rank 0.
- When we do a union operation, if the two representatives have different ranks, the ranks stay the same, and the one of smaller rank is assigned the one of larger rank as its parent.
- When we do a union operation, if the two representatives have the same rank, then one becomes the parent of the other, and its rank is increased by one.

Rank 0

The minimum (and only) size of a rank 0 representative is 1 (this happens initially).

Rank 1

To get a rank 1 representative, we form a union of either a rank 0 and a rank 1 set or two rank 0 sets.

For the minimum possible size, it must be the second case, and the two rank 0 sets must each be of minimum size (1), so this gives a minimum size for a rank 1 set of 2.

Rank 2

To get a rank 2 representative, we form a union of either a rank 2 and a rank 0 or 1 set or two rank 1 sets.

For the minimum possible size, it must be the second case, and the two rank 1 sets must each be of minimum size (2), so this gives a minimum size for a rank 2 set of 4.

Rank 3

To get a rank 3 representative, we form a union of either a rank 3 and a rank 0, 1 or 2 set or two rank 2 sets.

For the minimum possible size, it must be the second case, and the two rank 2 sets must each be of minimum size (4), so this gives a minimum size for a rank 3 set of 8.

And so on

To get a rank $k+1$ representative, we form a union of either a rank $k+1$ and a rank j set for some $j < k+1$ or two rank k sets.

For the minimum possible size, it must be the second case, and the two rank k sets must each be of minimum size, which we are assuming is 2^k so this gives a minimum size for a rank $k+1$ set of

$$2^k + 2^k = 2 \times 2^k = 2^{k+1}.$$

Working your PECS

The previous example was an argument by induction. Carrying out an argument by induction happens in four phases.

- Preparation
- Execution
- Checking
- Satisfaction

Preparation

The preparation phase of an argument by induction looks something like this.

- Isolate the property that you are trying to verify and the parameter, n, associated with it.
- Confirm by hand that for small values of the parameter the property is true.
- Convince yourself that to confirm it's true for $n = 5$ it really helps to know that it's true for $n = 4$ (or for all $n < 5$).
- Pause and reflect. If you're happy that you understand what's going on proceed to execution. If not, repeat some or all of the above.

Execution

The execution phase of an argument by induction is the most technical and prescribed (once you're an induction expert you can take some liberties).

It consists of four parts: a statement, verification of the base case(s), the inductive step, and a conclusion.

- We will prove by induction that, for every non-negative integer n, *insert property to verify here*.
- For $n = 0$, *the property* is true because *explicit verification of this case*.
- For any $k \geqslant 0$, assuming *the property* is true for k (or, for all $j \leqslant k$), *the property* is true at $k + 1$ because *explain why we can take a step up*.
- Therefore, by induction, *the property* is true for all n.

The text in black above is boilerplate that can be used in **every** argument by induction. The text in red needs to be replaced on a case by case basis.

Checking and Satisfaction

Checking is basically debugging. The problem is there's no compiler to find syntax errors.

- Have you forgotten anything? Like the base case?
- Does the inductive step work from 0 to 1? Or do we need to verify a few small cases before dropping into the inductive step.
- Does the inductive step really only use smaller values of the parameter? Otherwise the argument is circular (if it uses the same value of the parameter) or completely undefined (if it uses larger ones!)
- Ideally, show your argument to a peer and see if it convinces them (and don't let them be polite!)
- Go back to Execution, or possibly even Preparation if necessary.

Assuming that Checking is complete, Satisfaction can commence.

Execution phase for the union-find claim

(Description of algorithm omitted)

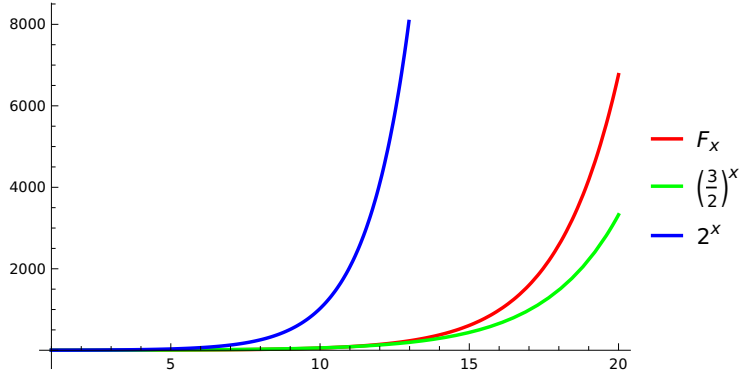
- We will prove by induction that, for every non-negative integer n, *the minimum possible size of a set whose representative has rank n is 2^n.*
- For $n = 0$, this is true because *every representative of rank 0 is a single point and $2^0 = 1$.*
- For any $k \geq 0$, assuming *this* is true for all $j \leqslant k$, *this* is true at $k + 1$ because *to form a set whose representative has rank $k + 1$ we take the union of two sets, one of whose representatives is already of rank $k + 1$, or two sets of rank k. Only the latter case can produce a set of smallest possible size and the smallest size in this case is twice the size of the smallest possible set of rank k, that is $2 \times 2^k = 2^{k+1}$, as we claimed.*
- Therefore, by induction, for all n, *the minimum possible size of a set whose representative has rank n is 2^n.*

Another example

Remember the Fibonacci numbers?

$$f_0 = f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \quad \text{for } n > 1.$$

For various reasons, we're interested in understanding how quickly they grow. Here is a plot:



Fibonacci numbers

If we plot $(n, log\, f_n)$ we get something that looks very much like a straight line. It suggests that f_n is somewhere in the neighbourhood of $(1.6)^n$. Maybe we can prove something like this:

$$(3/2)^n \leqslant f_n \leqslant 2^n$$

Preparation

First we should look at the data.

n	$(3/2)^n$	f_n	2^n
0	1	1	1
1	$3/2$	1	2
2	$9/4$	2	4
3	$27/8$	3	8
4	$81/16$	5	16
5	$243/32$	8	32
6	$729/64$	13	64

More preparation

What's needed is a concrete example of an inductive step.

$$(3/2)^5 \leqslant f_5 \leqslant 2^5$$
$$(3/2)^6 \leqslant f_6 \leqslant 2^6$$

So, why can we conclude:

$$(3/2)^7 \leqslant f_7 \leqslant 2^7$$

Even more preparation

Carrying out a concrete example of an inductive step.

In a chained inequality like this it's generally easiest to do the two halves separately.

$$
\begin{aligned}
f_7 &= f_6 + f_5 \\
&\geqslant (3/2)^6 + (3/2)^5 \\
&= (3/2)^6 \, (1 + 2/3) \\
&= (3/2)^6 \times (5/3) \\
&\geqslant (3/2)^7 \quad \text{since } 5/3 > 3/2.
\end{aligned}
$$

$$
\begin{aligned}
f_7 &= f_6 + f_5 \\
&\leqslant 2^6 + 2^5 \\
&= 2^6 \, (1 + 1/2) \\
&= 2^6 \times (3/2) \\
&\leqslant 2^7 \quad \text{since } 3/2 < 2.
\end{aligned}
$$

Execution

- We will prove by induction that, for every integer $n \geqslant 5$, $(3/2)^n \leqslant f_n \leqslant 2^n$
- For $n = 5, 6$, this is true because *we computed the values in those cases and checked.*
- For any $k \geqslant 6$, assuming *this* is true for all $5 \leqslant j \leqslant k$, *this* is true at $k + 1$ because *repeat computations on previous slide changing all 7s, 6s and 5s, to $k + 1$, k and $k - 1$ respectively.*
- Therefore, by induction, for all $n \geqslant 5$, $(3/2)^n \leqslant f_n \leqslant 2^n$

Checking and Satisfaction

- Checking is already done in this case (after all, this is a lecture)
- Satisfaction? I'll let you be the judge of that but ...
- There's more to dig up from our preparatory work. The problem with the lower bound was that it failed at $n = 1$. We could have fixed that by changing the statement from $(3/2)^n \leqslant f_n$ to $(3/2)^{n-1} \leqslant f_n$, or $(2/3) \times (3/2)^n \leqslant f_n$. If we did that, and thought a bit more ...
- We might be able to see that if $\varphi \approx 1.6$ is the number[1] that satisfies $\varphi + 1/\varphi = \varphi^2$, then $\varphi^{n-1} \leqslant f_n \leqslant \varphi^n$ and so $f_n = \Theta(\varphi^n)$.
- Exact formulas for f_n exist.

[1] In fact $\varphi = (1 + \sqrt{5})/2$, called the golden ratio

Isn't this just maths?

- When we want to analyse the time-complexity of algorithms, particularly recursive ones, the inductive approach is essential.
- It can even help with understanding nested loops or ones with complex control conditions.
- The obvious way it works in the recursive case is "The time required to do the task at size n is *some basic amount of processing, usually constant* plus *the time required for one or more instances of the same task at smaller values*"
- That gives recurrences like the Fibonacci one to give upper and lower bounds on the time required and can hopefully lead to a full analysis.