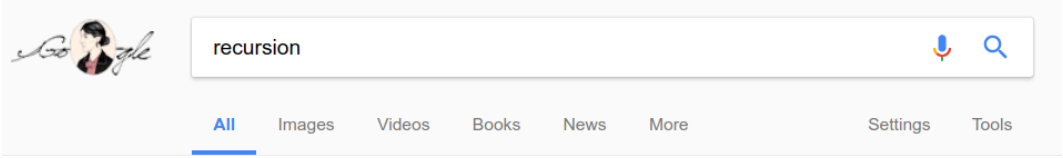
Lecture #0b: Recursion (and some Induction)

COSC 3020: Algorithms and Data Structures

Lars Kotthoff[1]
larsko@uwyo.edu

[1] with material from various sources


recursion
All
Images
Videos
Books
News
More
Settings
Tools
About 8,220,000 results (0.47 seconds)
Did you mean: recursion

Random Permutations (rPnastma detinoRmuo)

Problem: Permute a string so that every reordering of the string is equally likely.

Thinking Recursively

1. DO NOT START WITH CODE. Instead, write the story of the problem, in natural language.
2. Define the problem: What should be done given a particular input?
3. Identify and solve the (usually simple) base case(s).
4. Determine how to break the problem down into smaller problems of the same kind.
5. Call the function recursively to solve the smaller problems. Assume it works. Do not think about how!
6. Use the solutions to the smaller problems to solve the original problem.

Once you have all that, write out your solution in comments and then translate it into code.

Random Permutations (rPnastma detinoRmuo)

Problem: Permute a string so that every reordering of the string is equally likely.

Idea:

1. Pick a letter to be the first letter of the string. (Every letter should be equally likely.)
2. Pick the rest of the string to be a random permutation of the remaining string (without that letter).

Random Permutations (rPnastma detinoRmuo)

Problem: Permute a string so that every reordering of the string is equally likely.

```
function permute(str) {
  if(str.length > 1) {
    var i = Math.floor(Math.random() * str.length);
    return str[i] +
        permute(str.substring(0, i) +
                str.substring(i + 1));
  } else {
    return str;
  }
}
```

`Math.floor(Math.random()* str.length))` returns an integer from $\{0, 1, \ldots, n-1\}$ uniformly at random.

Induction and Recursion, Twins Separated at Birth?

	Induction	Recursion
Base case	Prove for some small value(s).	Calculate for some small value(s).
Inductive step	Break a larger case down into smaller ones that we assume work (the Induction Hypothesis).	Otherwise, break the problem down in terms of itself (smaller versions) and then call this function to solve the smaller versions, assuming it will work.

Differences

Recursion Usually goes from large to small – start with the full problem and break it down into smaller parts.

Induction Usually goes from small to large – start with the simplest (base) case and build up from there.

Proving a Recursive Algorithm is Correct

Just follow your code's lead and use induction.

Your base case(s)? Your code's base case(s).

How do you break down the inductive step? However your code breaks the problem down into smaller cases.

Inductive hypothesis? The recursive calls work for smaller-sized inputs.

Proving a Recursive Algorithm is Correct

```
// Pre: n >= 0.
// Post: returns n!
function fact(n) {
  if(n == 0)
    return 1;


  else
    return
      n * fact(n-1);

}
```

Prove: `fact(n)` $= n!$

Base case: $n = 0$

`fact(0)` returns 1 and $0! = 1$ by definition.

Inductive hyp: `fact(n)` returns $n!$ for all $n \leq k$.

Inductive step: For $n = k + 1$, code returns `n * fact(n-1)`. By IH, `fact(n-1)` is $(n-1)!$ and $n! = n \cdot (n-1)!$ by definition.

Proving a Recursive Algorithm is Correct

Problem: Prove that our algorithm for randomly permuting a string gives an equal chance of returning every permutation (assuming `Math.random()` works as advertised).

Base case: Strings of length 1 have only one permutation.

Induction hypothesis: Assume that the letters of a string of a length n are permuted randomly.

Inductive step: For a string of length $n+1$, we choose the first letter uniformly at random from the string, and call `permute` recursively on a smaller string of length n. This call works (by the induction hypothesis), and as the first letter was chosen randomly, the function works as advertised.