Unit #3: Recursion, Induction, and Loop Invariants

CPSC 221: Basic Algorithms and Data Structures

Anthony Estey, Ed Knorr, and Mehrdad Oveisi

2016W2

Unit Outline

- Thinking Recursively
- Recursion Examples
- Analyzing Recursion: Induction and Recurrences
- Analyzing Iteration: Loop Invariants
- How Computers Handle Recursion
    - Recursion and the Call Stack
    - Iteration and Explicit Stacks
    - Tail Recursion

Learning Goals

- Describe the relationship between recursion and induction.
- Prove that a program is correct using loop invariants and induction.
- Become more comfortable writing recursive algorithms.
- Convert between iterative and recursive algorithms.
- Describe how a computer implements recursion.
- Draw a recursion tree for a recursive algorithm.

$\underbrace{\text{Random Permutations}}_{\text{example input}}$ ($\underbrace{\text{rPnastma detinoRmuo}}_{\text{example output}}$)

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

Many possibilities.

One possible algorithm:

◇ Randomly pick one letter, L, from input string

◇ Delete L from the input

◇ Add L to the output string

◇ Repeat until no letters left in the input

Can we solve this problem using recursion?

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 the steps of your solution as comments, and then fill in the code for each comment.

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

It's slightly simpler if we pick a letter to be the **last** letter of the output.

So change (1) to this

Random Permutations (rPnastma detinoRmuo)

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

Passing by reference is useful:
- ◇ to modify the parameter (as S in permute here)
- ◇ to avoid copying large items; use with "const" to make unmodifiable

```
// randomly permute the first n characters of S
void permute(string & S, int n) {
  if( n > 1 ) {
    int i = rand() % n; // swap a random character of S
    char tmp = S[i];    // with the last character
    S[i] = S[n-1];
    S[n-1] = tmp;
    permute(S, n-1);    // randomly permute S[0..n-2]
  }
}
```

S is "passed by reference"
n is "passed by value"

swapping ith letter with the last letter

e.g.: (2017 % 10) == 7

Recall that `rand() % n` returns an integer from $\{0, 1, \ldots n-1\}$ uniformly at random.

```
int i = 3;  int & j = i;  i++;  j++;
cout << i << "==" << j; // outputs: 5==5
```

A reference to a variable is an alias, just giving a new name to an existing memory location.

Random Permutations (rPnastma detinoRmuo)

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

```
// randomly permute the first n characters of S
void permute(string & S, int n) {
  if( n > 1 ) {
    int i = rand() % n; // swap a random character of S
    char tmp = S[i];    // with the last character
    S[i] = S[n-1];
    S[n-1] = tmp;
    permute(S, n-1);    // randomly permute S[0..n-2]
  }
}
```

```
Example usage:

string myStr = "ABCDEFG";
permute(myStr, myStr.length());
```

S	n
ABCDEFG	7
ABGDEFC	6
AFGDEBC	5
AFEDGBC	4
DFEAGBC	3
DFEAGBC	2
DFEAGBC	1

Induction and Recursion: Twins Separated at Birth?

Induction:

Base Case

Prove for some small value(s).

Inductive Step: Break a larger case down into smaller ones that we assume work (the Induction Hypothesis).

Recursion:

Base Case

Calculate for some small value(s).

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.

X is true for n=1
If X is true for n=k, then X is true for n=k+1
Therefore,
X is true for n=1, 2, 3, 4, ...

Proving that 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? Assume that: The recursive calls work for smaller-sized inputs.

Proving that a Recursive Algorithm is Correct

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









  else
    return n*fact(n-1);
}
```

$$n! = \begin{cases} 1 & \text{if } n=0 \\ n\,(n-1)! & \text{if } n > 0 \end{cases}$$

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

Base case: $n = 0$:
`fact(0)` returns 1; and
$0! = 1$, by definition

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

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

Proving that a Recursive Algorithm is Correct

n! is num of permutations
1/n! is prob. of one of permutations

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

Induction of the length of S:

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

Induction Hypothesis: Assume that our call to `permute(S, n-1)` works (i.e., it randomly permutes the first `n-1` characters of `S`).

We choose the last letter uniformly at random from the string. To get a random permutation, we need only randomly permute the remaining letters. `permute(S, n-1)` does exactly that.

The last letter is OK (selected at random)
The rest of the letters are also OK (selected at random)
Thus all of the letters are OK (selected at random)

Very loosely speaking