ECE 2574: Data Structures and Algorithms - Applications of Recursion II

C. L. Wyatt

Today we will look at another application of recursion, a depth first search of a graph, as well as the relationship between recursion and mathematical induction.

- Representing problems as state-space graphs
- Searching state spaces using recursive depth-first search
- Recursion and recurrence relations

Many problems can be solved by searching

One represents the problem as a **state space**.
Starting at some **initial state** and following **state transisitions** leads to other states.
When you reach a **goal state** you have found the solution.

Example: Peg Solitaire

Example: 8 Queens

Example: Path Finding

Example Constraint Satisfaction

In each of these examples the goal state is at a fixed depth.

The search can proceed **depth-first** in a recursive manner

```
function recursive_dfs(state)

  if(state is goal)
     return state
  else
     for each successor of state
        return recursive_dfs(successor)
  end
endfunction
```

This can be converted to an iterative solution using a stack (next meeting)

Example mini-sudoku

Consider a simplified version of Sudoku in a 3x3 form.

- 3x3 square
- each number 1-3 must be used on each row and column exactly once

We can use Backtracking-Search to solve it.
See example code.

Mathematical Induction is a technique often used with verification proofs.

It is based on the following axiom:
Mathematical Induction: A property P(n) that involves an integer n is true for all n $>=$ 0 if

1. P(0) is true, and
2. if P(k) is true for any k $>=$ 0, then P(k+1) is true.

Step 1. is the base case.
Step 2. is the inductive step.

Induction Example: sum of first n positive integers

Prove:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Base Case:

$$\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2} = 1$$

Induction Step:

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

$$\sum_{i=1}^{k} i + (k+1) = \frac{k(k+1)}{2} + (k+1)$$

$$\sum_{i=1}^{k+1} i = \frac{(k+1)((k+1)+1)}{2}$$

Recursion defines a solution in terms of itself.

A recursive procedure is one whose evaluation at (non- initial) inputs involves invoking the procedure itself at another input.

Recurrence relation with an initial condition

fact(n) = n*fact(n-1) with fact(0) = 1 and fact(1) = 1

Recursive functions have a base case and (one or more) recursions.

Induction is a powerful tool to prove properties of recursive algorithms.

Induction and recursion are very similar concepts

- Induction has a base case
- Recursion has a base case
- Induction has an inductive step (assume k, show k+1)
- Recursion has a recursive step, compute at k by computing at f(k)

In general we use induction to prove 2 properties of algorithms:

- correctness and
- complexity

Properties of algorithms: why do we care ?

Correctness: we would like to know the algorithm solves the problem we want it to solve.
Complexity: we would also like to know how many resources we expect the algorithm to use.
Resources:

- How much memory ?
- How long will it take ?
- Under what assumptions about the inputs ?

Example using induction to prove correctness: Factorial

Prove the following function computes n!

```
function fact(in n:integer):integer
  if(n is 0) return 1
  else return n*fact(n-1)
endfunction
```

Base case: $n == 0$
This follows directly from the pseudo-code. $\text{fact}(0) = 1$.

Proving Factorial correct: Inductive step

Assume that fact(k) = k! = k*(k-1)*(k-2)* .... * 2 * 1
By definition, fact(k+1) returns (k+1)*fact(k)
We've assumed fact(k) returns k*(k-1)*(k-2)* .... * 2 * 1 and that it is correct.
Then fact(k+1) returns (k+1)* k*(k-1)*(k-2)* .... * 2 * 1 which is (k+1)! by definition.
Base case plus inductive conclusion prove algorithm correct.

Next Actions and Reminders

- Read CH Chapter 6 (it is a short chapter)
- Complete the warmup before noon on Wed 9/27
- P1 is due Wednesday by 11:55 pm