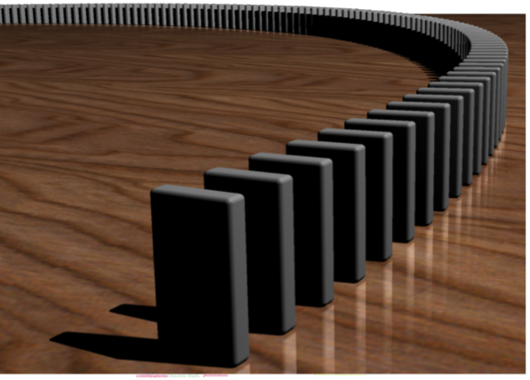
CS 220: Discrete Structures and their Applications

Recursive algorithms and induction

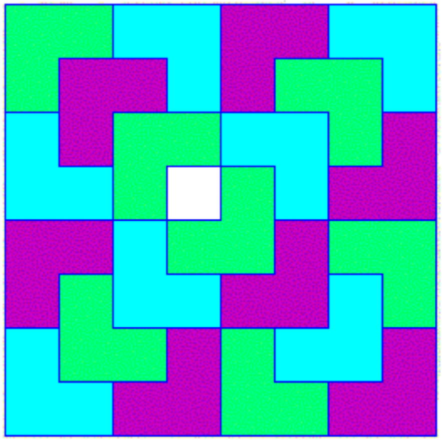
6.8 in zybooks



Colorado State University

Induction and Recursion

Several of the inductive proofs we looked at lead to recursive algorithms:



- The triomino tiling problem
- Making postage using 3 and 5 cent stamps
- Generating all subsets of a set recursively

Induction is useful for designing and proving the correctness of recursive algorithms

String reversal

Consider the following recursive algorithm for reversing a string:

```
reverse_string(s)
      if s is the empty string:
            return s
      let c be the first character in s
      remove c from s
      s' = reverse_string(s)
      return the string s' with c added to the end
```

String reversal

Proof of correctness of reverse_string

```
reverse_string(s)
      if s is the empty string:
            return s
      let c be the first character in s
      remove c from s
      s' = reverse_string(s)
      return the string s' with c added to the end
```

By induction on the length of the string

Base case: If s has length 0 the algorithm returns s which is its own reverse.

String reversal

Proof of correctness of reverse_string

```
reverse_string(s)
        if s is the empty string:
                return s
        let c be the first character in s
        remove c from s
        s' = reverse_string(s)
        return the string s' with c added to the end
```

Inductive step: assume that reverse_string works correctly for strings of length k and show that for k+1

Let s be a string of length k + 1. $s = c_1c_2 \ldots c_kc_{k+1}$.

reverse_string makes a recursive call whose input is $c_2 \ldots c_kc_{k+1}$.

By the induction hypothesis it returns the inverse: $c_{k+1}c_k \ldots c_2$

It then adds c_1 at the end, returning $c_{k+1}c_k \ldots c_2c_1$, which is the reverse of s

recursive power

```
def pow(x, n):
    #precondition: x and n are positive integers
    if (n == 0):
       return 1
    else :
       return x * pow(x, n-1)
    }
}
```

recursive power

```
def pow(x, n):
    #precondition: x and n are positive integers
    if (n == 0):
        return 1
    else :
        return x * pow(x, n-1)
```

Claim: the algorithm correctly computes x^n.

Proof: By induction on n

Basis step: n = 0: it correctly returns 1

Inductive step: assume that for n the algorithm correctly returns x^n.

Then for n+1 it returns $x\ x^n = x^{n+1}$.

Egyptian Exponentiation

In PA2 you are implementing an iterative exponentiation algorithm, based on the following recursive definition:

```
def pow(x, n):
    #precondition: x and n are positive integers
    if n == 0:
       return 1
    else if not (n/2 == n//2):
       return x * pow(x**2, n//2)
    else:
       return pow(x**2, n//2)
```

Does linear induction work for this algorithm? Why (not) ?

What do we need?

the power set

```
def powerset(s) :
    if len(s) == 0:
        return {frozenset()}
    else :
        element = s.pop()
        pwrset = powerset(s)
        return pwrset.union({ x.union({element})
                    for x in pwrset})
```