COMP 182 Algorithmic Thinking

Mathematical Induction

Luay Nakhleh
Computer Science
Rice University

Reading Material

- Chapter 5, Section 1-4

$$[P(1) \land \forall k(P(k) \to P(k+1))] \to \forall n P(n)$$

Why Is It Valid?

- The well-ordering property of positive integers:
    - Every nonempty subset of the set of positive integers has a least element (an element that is smaller than or equal to every other element in the subset).

Why Is It Valid?

Why Is It Valid?

- Assume $P(1)$ is true and $P(k) \to P(k+1)$ for every k.

Why Is It Valid?

- Assume $P(1)$ is true and $P(k) \rightarrow P(k+1)$ for every k.
- Define $S = \{n$: n is positive integer and $P(n)$ is False$\}$.

Why Is It Valid?

- Assume $P(1)$ is true and $P(k) \to P(k+1)$ for every k.
- Define $S=\{n$: n is positive integer and $P(n)$ is False$\}$.
- By the well-ordering property of the positive integers, S has a least element; call it m.

Why Is It Valid?

- Assume $P(1)$ is true and $P(k)\rightarrow P(k+1)$ for every k.
- Define $S=\{n$: n is positive integer and $P(n)$ is False$\}$.
- By the well-ordering property of the positive integers, S has a least element; call it m.
- $m>1$ (why?).

Why Is It Valid?

- Assume $P(1)$ is true and $P(k) \rightarrow P(k+1)$ for every k.
- Define $S=\{n$: n is positive integer and $P(n)$ is False$\}$.
- By the well-ordering property of the positive integers, S has a least element; call it m.
- $m>1$ (why?).
- Furthermore, $P(m-1)$ is true (why?)

Why Is It Valid?

- Assume $P(1)$ is true and $P(k) \rightarrow P(k+1)$ for every k.
- Define $S = \{n$: n is positive integer and $P(n)$ is False$\}$.
- By the well-ordering property of the positive integers, S has a least element; call it m.
- $m>1$ (why?).
- Furthermore, $P(m-1)$ is true (why?)
- Contradiction! (why?)

Why Is It Valid?

- Assume $P(1)$ is true and $P(k) \rightarrow P(k+1)$ for every k.
- Define $S=\{n$: n is positive integer and $P(n)$ is False$\}$.
- By the well-ordering property of the positive integers, S has a least element; call it m.
- $m>1$ (why?).
- Furthermore, $P(m-1)$ is true (why?)
- Contradiction! (why?)
- Therefore, S must be empty. In other words, $P(n)$ is true for all positive integers n.

To prove that ***P*(*n*) is true for all positive integers *n***, where *P*(*n*) is a propositional function, we complete two steps:

Basis Step: We verify that ***P*(1) is true**.

Inductive Step: We show that the conditional statement ***P*(*k*)→*P*(*k*+1) is true for all positive integers *k***.

The assumption that *P*(*k*) is true is called the **inductive hypothesis**.

Examples

- Prove that:
  - $1+2+\ldots+n = n(n+1)/2$.
  - $1+3+5+\ldots+(2n-1) = n^2$.
  - $1+2+2^2+\ldots+2^n = 2^{n+1}-1$.
  - If S is a finite set with n elements, where n is a nonnegative integer, then S has 2^n subsets.

Strong Induction

- Sometimes we cannot easily prove a result using the mathematical induction technique we have seen; instead we can use another form of mathematical induction called <u>strong induction</u>.
- Strong Induction: To prove *P(n)* is true for all positive integers *n*, where *P(n)* is a propositional function, we complete two steps:
  - Basis Step: We verify that the proposition *P(1)* is true.
  - Inductive Step: We show that the conditional statement $[P(1) \wedge P(2) \wedge \ldots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers *k*.

Example

- Prove that if n is an integer greater than 1, then either n is prime or n can be written as the product of primes.

Recursively Defined Functions/Sets/Structures and Structural Induction

- The set of full binary trees can be defined recursively by the following steps:
  - Basis Step: There is a full binary tree consisting only of a single node r.
  - Recursive Step: If $T1$ and $T2$ are disjoint full binary trees, there is a full binary tree, denoted by $T1 \cdot T2$, consisting of a root r together with edges connecting the root r to each of the roots of the left subtree $T1$ and the right subtree $T2$.

Recursively Defined Sets/Structures and Structural Induction

- We can use mathematical induction over the set of positive integers and a recursive definition to prove a result about a recursively defined set/structure.
- However, instead of using mathematical induction directly to prove results about recursively defined sets/structures, we can use a more convenient form of induction known as structural induction.
- A proof by structural induction consists of two steps:
  - Basis Step: Show that the result holds for all elements specified in the basis step of the recursive definition to be in the set.
  - Recursive Step: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

Recursively Defined Sets/Structures and Structural Induction

- To use structural induction on full binary trees:
  - Basis Step: Show that the result holds for the tree that consists of a single node.
  - Recursive Step: Show that if the statement is true for the trees *T1* and *T2* , then it is true for the tree *T1*⋅*T2* that is formed by the recursive step.

Example

- We define the height $h(T)$ of a full binary tree T recursively.
  - Basis Step: The height of the full binary tree T consisting of only a root r is $h(T) = 0$.
  - Recursive Step: If $T1$ and $T2$ are full binary trees, then the full binary tree $T=T1 \cdot T2$ has height $h(T) = 1+max(h(T1),h(T2))$.
- Let $n(T)$ denote the number of nodes in a full binary tree (observe that $n(T)=1+n(T1)+n(T2)$, where $T1$ and $T2$ are the left and right children of the root of T).
- Prove that if T is a full binary tree, then $n(T) \leq 2^{h(T)+1}-1$.

Strings, Recursively Defined Sets, and Structural Induction

- Strings play an important role in computer science.
- Strings are defined over a given alphabet Σ.
- For example, every "English string" is defined over the alphabet $\Sigma=\{a,..,z,A,..,Z\}$.
- DNA strings are defined over the alphabet $\Sigma=\{A,C,T,G\}$.
- Binary strings are defined over the alphabet $\Sigma=\{0,1\}$.
- We denote by Σ^* the set of all strings defined over the alphabet Σ. This set includes a special string, ε, which is the empty string (the string that contains no symbols).

Strings, Recursively Defined Sets, and Structural Induction

- The set Σ^* of strings over the alphabet Σ can be defined recursively by
  - Basis Step: $\varepsilon \in \Sigma^*$.
  - Recursive Step: If $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma$.
- Using this definition, we can "list" the strings over the alphabet $\Sigma=\{0,1\}$: $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$
- Recursive definitions can be used to define operations or functions on the elements of recursively defined sets.

Recursively Defined Sets/Structures and Structural Induction

- To use structural induction to prove $P(w)$ is true for all $w \in \Sigma^*$:
  - Basis Step: Show that $P(\varepsilon)$ is true.
  - Recursive Step: Show that if $P(w)$ is true for $w \in \Sigma^*$, then $P(wa)$ is true for $a \in \Sigma$.

Strings, Recursively Defined Sets, and Structural Induction

- We define the concatenation of strings over alphabet Σ as follows:
  - Basis Step: If $w \in \Sigma^*$, then $w\varepsilon = w$.
  - Recursive Step: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1(w_2x) = (w_1w_2)x$.
- Give a recursive definition of $l(w)$, the length of the string w.
- Use structural induction to prove that $l(xy) = l(x) + l(y)$, where x and y are two strings in Σ^*.

Strings, Recursively Defined Sets, and Structural Induction

- We define the concatenation of strings over alphabet Σ as follows:
    - Basis Step: If $w \in \Sigma^*$, then $w\varepsilon = w$.
    - Recursive Step: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1(w_2x) = (w_1w_2)x$.

$P(y): l(xy) = l(x) + l(y)$

- Give a recursive definition of $l(w)$, the length of the string w.
- Use structural induction to prove that $l(xy) = l(x) + l(y)$, where x and y are two strings in Σ^*.

Recursion and Induction

- Mathematical induction, and its variant strong mathematical induction, can be used to prove that a recursive algorithm is correct, that is, that it produces the desired output for all possible input values.
- Consider the following recursive algorithm:

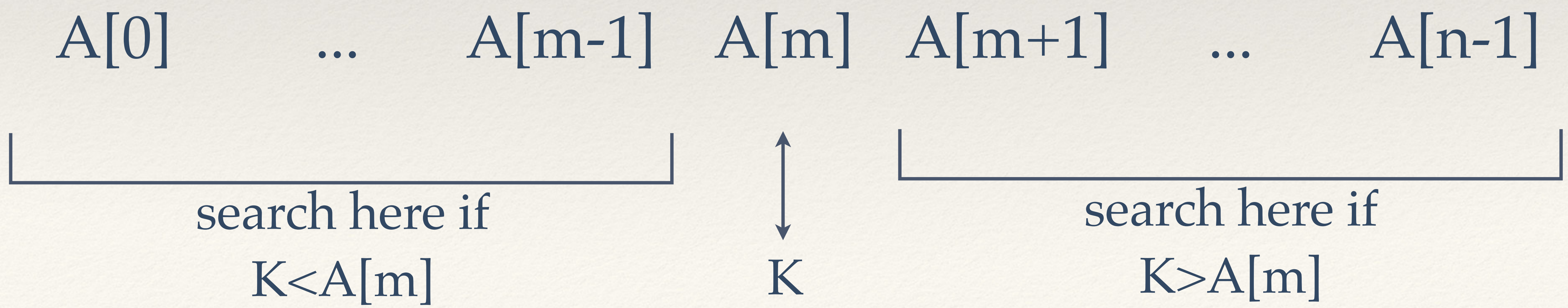
```
Mystery
Input: Nonzero real number a, and nonnegative integer n.
Output:....

If n=0 then
    Return 1;
Else
    Return a∗Mystery(a,n-1);
```

What does algorithm **Mystery** compute? Prove your answer.

Recursion and Induction: Binary Search

- Binary search is an efficient algorithm for searching in a sorted array.
- It works by comparing a search key K with the array's middle element A[m].
- If they match, the algorithm terminates; otherwise, the same operation is repeated recursively for the first half of the array if K<A[m], and for the second half if K>A[m].



Recursion and Induction: Binary Search

Algorithm 4: RecursiveBinarySearch.

```
   Input: An array $A[0 \dots n-1]$ sorted in ascending order, a search key $K$, and left/right boundaries $l$ and $r$.
   Output: An index of the array's element that is equal to $K$ or -1 if there is no such element.
1  if $l > r$ then
2      return -1;
3  $m \leftarrow \lfloor (l+r)/2 \rfloor$;                // $m$ is the index of the middle entry
4  if $K = A[m]$ then
5      return $m$;
   else if $K < A[m]$ then
6      return $RecursiveBinarySearch(A, K, l, m-1)$
   else
7      return $RecursiveBinarySearch(A, K, m+1, r)$
```

Recursion and Induction: Binary Search

- Though binary search is clearly based on a recursive idea, it can be easily implemented as a nonrecursive algorithm, too.

Algorithm 5: BinarySearch.

Input: An array $A[0 \ldots n-1]$ sorted in ascending order, and a search key K.
Output: An index of the array's element that is equal to K or -1 if there is no such element.

```
1 $l \leftarrow 0$;                          // The left boundary of the region being searched
2 $r \leftarrow n-1$;                        // The right boundary of the region being searched
3 while $l \leq r$ do
4     $m \leftarrow \lfloor (l+r)/2 \rfloor$;             // $m$ is the index of the middle entry
5     if $K = A[m]$ then
6         return $m$;
      else if $K < A[m]$ then
7         $r \leftarrow m-1$;
      else
8         $l \leftarrow m+1$;
  return -1
```

Questions?