Discrete Structures II

Mwangi H. (Ph.D.)

CS Yr 3.1
Department of Computing
J.K.U.A.T.

March 11, 2024

- Recall existence proofs were quite easy. We just find one example which satisfies the statement.
- Proving that a statement holds for every possible choice is much more challenging.
- Mathematical Induction is a mathematical technique to reason about, and prove, that a statement holds for every choice of natural number.

Consider the predicate statement $P(n)$ for some natural number n. If we know that $P(0)$ is true, $P(1)$ is true, and $P(2)$ is true, then is $P(3)$ true? Is $P(1000)$true? This is the job of mathematical induction.

Example 1.1: Principle of Mathematical Induction

We look to prove that $P(n)$ is true for all natural numbers n that are greater than some natural number n_0

1. **Basis Step.** Show directly that $P(n_0)$ is true
2. **Inductive Step.** Show that the implication $P(k) \rightarrow P(k+1)$ is true for all $k \in \mathbb{N}, k \geq n_0$

If both 1 and 2 can be shown to be true, then by **induction** $P(n)$ is true for all natural numbers greater than or equal to n_0

- In this kind of proof, we prove the inductive step of mathematical induction by following the inductive hypothesis.
- As with proving any implication, we assume the left hand side is true, and then prove that the right hand side.
- In mathematical induction, assuming $P(k)$ is the inductive hypothesis and we "assume that the inductive hypothesis is true" in order to show $P(k+1)$ must also be true.
- Formally, mathematical induction gives us the following implication:

$$(P(n_0) \ \wedge \ \forall k \geq n_0 \ (P(k) \rightarrow P(k+1)) \ ) \quad \rightarrow \quad \forall n \geq n_0 \ P(n)$$

- *Both* parts of the left hand side must be true in order to imply the right-hand side. If either the base case does not hold or the inductive step does not hold, then induction does not apply.
- In the inductive step, we do not assume that $P(k)$ is true for all choices of k. Rather, we show that if $P(k)$ is true, then $P(k=1)$ must also be true.

Example 1.2: Proof by Induction

Recall the summation fromula $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Prove this formula is correct for all $n \geq 1$

Proof.

1. Basic step: $\sum_{i=1}^{1} i = 1 = \frac{1(2)}{2}$ Therefore, the formula is correct for $n = 1$
2. Inductive step. Assume the inductive hypothesis, that $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$
   We must show that
   $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$

$$\sum_{i=1}^{k+1} i = 1 + 2 + 3 + \cdots + k + (k+1)$$

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

$$= \frac{k(k+1)}{2} + (k+1) \qquad \text{by inductive hypothesis}$$

Proof Contd.

$$= \frac{k(k+1)+2(k+1)}{2}$$
$$= \frac{k^2+3k+2}{2}$$
$$= \frac{(k+1)(k+2)}{2}$$

Therefore, by induction, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for all $n \geq 1$ □

- Mathematical induction reasons about statements involving the natural numbers.
- However, recall that we can form a bijection between the natural numbers and any countably infinite set.
- Therefore, mathematical induction can be applied to any countably infinite set.

Example 1.3

Show that the sum of the first n odd positive integers is equal to n^2. That is $1 + 3 + 5 + \cdots + (2n - 1) = n^2$. e.g.

$1, 3, 5, 7, 9, 11$ is the first 6 odd integers and we have that

$1 + 3 + 5 + 7 + 9 + 11 = 36 = 6^2$

Proof.

We proceed by induction.

The sum of the first 1 positive integers is 1^2

Assume the inductive hypothesis, that

$$1 + 3 + 5 + 7 + \ldots + (2k - 1) = k^2$$

Then we have,

$$\begin{aligned} 1 + 3 + 5 + \ldots + (2k - 1) + (2k + 1) &= k^2 + (2k + 1) \quad \text{by the inductive hypothesis} \\ &= k^2 + 2k + 1 \\ &= (k + 1)^2 \end{aligned}$$

Therefore, by induction the sum of the first n positive odd integers is equal to n^2 for all positive integers $n \geq 1$ □

Example 1.4: Proof by Induction with Inequalities

Use mathematical induction to prove that $n < 2^n$ for all positive integers n

Proof.

Let $P(n)$ be "$n < 2^n$"

For the basic step, $P(1)$ is true since $1 < 2^1 = 2$

For the inductive step; Assume $P(k)$ is true. That is $k < 2^k$, for some positive integer k.

Show that $k + 1 < 2^{k+1}$ is also true

$$k < 2^k \qquad \text{by inductive hypothesis}$$

$$k + 1 < 2^k + 1$$

$$k + 1 < 2^k + 1 \leq 2^k + 2^k \qquad 1 \leq 2^k \text{ since } k \text{ is positive}$$

$$k + 1 < 2^k + 1 \leq 2 \cdot 2^k$$

$$k + 1 < 2^k + 1 \leq 2^{k+1}$$

$$k + 1 < 2^{k+1}$$

Hence, by induction, $n < 2^n$ for all $n \geq 1$ □

Strong Induction

- "Strong" induction is a variant of mathematical induction which is sometimes easier to use than the "normal" induction.
- Strong induction is also called the second principle of mathematical induction or complete induction.
- It is called strong induction, because it makes a stronger hypothesis than mathematical induction. Informally, strong induction assumes that the condition $P(k)$ holds for all value less than or equal to k
- Therefore, our inductive hypothesis is now:

$$P(n_0) \wedge P(n_0 + 1) \wedge \cdots \wedge P(k - 1) \wedge P(k) \rightarrow P(k + 1)$$

Example 1.5: Existence Proof for the Fundamental Theorem of Arithmetic

Theorem 1.1

If $n > 1$ is an integer then n can be write as a product of primes

Proof.

Let $P(n)$ be "n can be written as a product of primes"

Basis step: $P(2)$ is true since 2 is itself a prime.

Inductive step: Let the inductive hypothesis be that $P(i)$ is true for all i, $2 \leq i \leq k$.

We show $P(k+1)$ must be true.

If $k+1$ is prime, then it is already written as a product of primes (itself) and thus $P(k+1)$ is true. Otherwise $k+1$ is a composite number and can be written as the product of two integers greater than 1 say a and b.

That is $k+1 = ab$ with $2 \leq a < k+1$ and $2 \leq b < k+1$. This is obvious since k is greater than 2 and thus both a and **b** must be greater that 2

By the inductive hypothesis, $P(a)$ is true and $P(b)$ is true. Since $k+1 = a \cdot b$, then $P(k+1)$ is also a product of primes by replacing a and b by their respective product of primes.

Therefore, by strong induction, $P(n)$ is true for all $n \geq 2$ □

Recall that the **well-ordering principle** states that every non-empty subset of the positive integers has a least element.

This can be proved by induction.

In fact, this property is equivalent to mathematical induction and can be used to prove statements just like induction.

Definition 1.1

A set is well-ordered if every one of its non-empty subsets has a least element.

The natural numbers N is well-ordered under the ordering induced by "less than".

As another example, the list of words in the English language are well-ordered under lexicographical ordering (dictionary order).

Proof by well-ordering

Proposition 1.1: Proof by well-ordering

Euclidean division states that if a is an integer and b is a positive integer, then there are integers q and r such that $a = bq + r$ and $0 \leq r < d$

Proof.

Let a, b be integers satisfying the hypothesis of Euclidean division. Let S be the set of non-negative integers of the form $a - bq$ for some integer q. The set is non-empty because we can choose q arbitrarily.

By the well-ordering principle, S has a least element. Call it r. Since $r \in S$ and r is non-negative. Moreover, $r = a - bq_0$ for some integer q_0

It must also be that $r < b$ Proceed by contradiction. Assume that $r \geq b$. Then, another element of S is $a - b(q_0 + 1)$ where $a - b(q_0 + 1) = a - bq_0 - b = r - b$. Where $r - b \geq 0$ since $r \geq b$. But r was the smallest element of S, so it cannot be that $r \geq b$. Hence $r < b$

By the well-ordering principle integers q, r exist satisfying $a = bq + r$ and $0 \leq r < b$ □

Exercises

1. Use mathematical induction to prove that $2^n < n!$ for every integer $n \geq 4$
2. Use mathematical induction to prove that $n^3 - n$ is divisible by 3 for all positive integers n.
3. Use mathematical induction to prove that $n^2 + n$ is divisible by 2 for all integers $n \geq 1$
4. Use mathematical induction to prove that $5^n - 1$ is divisible by 4 for all integers $n \geq 1$

- Recursion is arguably the most powerful tool for a computer scientist.
- It is used throughout theory and practice.
- Unfortunately, it is one of the more difficult concepts to understand in computer science.
- This section examines recursion in the context of mathematics and computer science.
- We will use the principle of induction to make logical arguments and proofs involving recursive constructs and structures.

Definition 2.1: Recursive Definition

A recursive definition of a set defines elements of the set in terms of other elements in the set.

A recursively-defined set is a set with a recursive definition. For example, the natural numbers are a recursively-defined set.

Example 2.1: Recursively-defined natural numbers

Let $\mathbb{N}$ be the set of numbers defined as follows:

1. 0 is in $\mathbb{N}$.
2. If an integer n is in $\mathbb{N}$ then $n+1$ is in $\mathbb{N}$

Recursive definitions have two crucial parts.

1. The Basis Step or "Base Case" It explicitly defines one more elements in the set.
2. The Recursive Step/Production Rule. It defines how to construct new elements of the set from other elements.

Since relations, functions, sequences are all themselves defined as certain kinds of sets, a recursive definition also applies to all of those discrete structures.

Example 2.2: Recursively-defined sequences

Let $a_0 = 1$ and $a_1 = 3$. Define $a_n = 2a_{n-1} + a_{n-2}$. Then, this sequence is: $1, 3, 7, 17, 41, 99, 239, 577, 1393, \ldots$

The Exclusion Rule

- Given a recursive definition of a set, it is possible that many different sets satisfy the definition.
- Consider again the recursive definition of the natural numbers in example 2.1
- The set $\{0, 1, 1.5, 2, 2.5, 3, 3.5 \ldots\}$ satisfies the definition
- Therefore, to get a unique set the exclusion rule states that the set contains nothing other than those elements specified in the basis step and that are generated by applications of the recursive step.
- Using the exclusion rule along with the previous recursive definition of the natural numbers, the natural numbers are uniquely the set:

$$\{0, 1, 2, 3, 4, 5, 6, \ldots\}$$

- Always assume the exclusion rule, unless otherwise stated.

Example 2.3: Recursive Summation

Let (a_n) be a sequence

Give a recursive definition for the summation

$\sum_{i=0}^{n} a_i$

Solution

Basis:

$\sum_{k=0}^{0} a_0 = a_0$

Recursive Step/Production Step:

$\sum_{i=0}^{n+1} a_i = \sum_{i=0}^{n} a_i + a_{n+1}$

Recursive definitions can lead to some very abstract, yet very useful definitions.

Recursive Functions and Algorithms

In computer science, the most important recursive definition is that of a recursive algorithm. We have already seen recursive functions several times in this course. The factorial function $Fact(n)$ can be defined recursively as:

$$Fact(n) = \begin{cases} 1, & n = 0, 1 \\ Fact(n-1) \cdot n, & \text{otherwise} \end{cases}$$

We can also define the Fibonacci function as the function which takes n as argument and returns the nth Fibonacci number:

$$Fib(n) = \begin{cases} n, & n = 0, 1 \\ Fib(n-1) + Fib(n-2), & \text{otherwise} \end{cases}$$

In eithercase the definition can be converted to recursive algorithms in the sense of computer science. That is, functions which call themselves.

Recursive Exercises

1. Write a java function which recursively computes 7^n for some n
2. Rewrite a recursive Euclidean algorithm function in Java. Euclidean algorithm computes a GCD between two integers by computing a a sequence of remainders until the remainder is 0.

Proving Recursive Algorithms

- Recursive definitions and mathematical induction are naturally intertwined. For a recursive algorithm, we have a base case, then, further solutions are build up modifications of that base case.
- This is exactly the intention of induction. We have a basis step that can be proved directly, then, we rely on the inductive hypothesis to guarantee the correctness for all n

Proposition 2.1: Recursive Correctness via Induction

Consider the following function which computes x^n for some integer x and some non-negative integer n

```
def power(x, n) :
 if (n == 0) :
  return 1
 else :
  return x * power(x, n-1)
```

Proof.

Proof of 2.1 Here we need to proof the function properly computed x^n Proceed by induction.

For the basis step, power(x,0) clearly returns clearly returns $x^0 = 1$ from the first if condition.

For the inductive step, assume that power(x,k) returns the correct result x^k for some $k \geq 0$

We will show that power(x,k+1) returns the correct result.

By the definition of power we see that power(x,k+1) returns x * power(x, k). By the inductive hypothesis, power(x,k)=x^k.

Thus we have power(x,k+1) $= x \cdot x^k = x^{k+1}$

Therefore, by mathematical induction, power(x,n) correctly computed x^n for all $n \geq 0$ □

Exercises

Give a recursive definition for the following sets.

1. $\{10, 12, 14, 16, 18, 20, \ldots\}$
2. $\{3x \mid x \in \mathbb{Z}^+\}$
3. $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \ldots\}$
4. Given Strings as Sequences of characters from an alphabet create strings.

   For some alphabet Σ the set of all possible strings from characters in Σ is denoted by Σ^*

   For example, if $\Sigma = \{0, 1\}$ then
   $\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

   Let $\Sigma = \{a, b\}$ Give a recursive definition for the set Σ^*

Loop Invariant

Subsection 1 focussed on how to prove recursive algorithms using induction, an extemely powerful too to reason about he correctness of algorithms. In this section we discuss the application of induction to computer science in the form of loop invariants

Definition 3.1: Loop Invariant

A loop invariant is a property of a program that is true before and after each iteration of a loop.

Lets begin with simple example: Consider the following Python program which computes the maximum element in a list of integers.

Example 3.1

```
def max(L) :
 m = L[0];
 i = 1;
 while (i < len(L)) :
  if (L[i] > m) :
   m = L[i]
  i = i + 1

 return m

print(max([4,12,5,1,56,1,4,7])) - output 56
```

Loop Invariant

- A loop invariant for this program would be that **m** is always the maximum element of **L** between the index 0 and i-1, inclusive.
- Before the loop **m = L[0]** and **i=1**. Hence, **m** is the maximum element of the **sub-list L[0:i]**.
- After the first iteration of the loop, we have **i=2**. Does the loop invariant hold? That is, is **m** equal to the maximum element in **L[0:2]**? Surely so, we check if **L[1]** $>$ **m** and, if so, assign m to be **L[1]**.
- To prove the overall correctness of the function **max(L)** we must apply induction.

Loop Invariant

- Proving that a loop invariant holds for every loop in a program is one step toward proving or verifying that the entire program is correct.
- Generally, formal program verification is a very difficult task. For more information, consider reading the articles Loop Invariant and Formal Verification.
- Later in your studies in computer science, you will see loop invariants a program verification again. When that time comes, remember induction!