Efficient Algorithms

Ekkapot Charoenwanit

Software Systems Engineering
TGGS
KMUTNB

Lecture 2: Mathematical Induction and Recurrence

What is Mathematical Induction?

- It is a very useful and powerful proof technique.
- It is a proof technique for mathematical/logical claims that involve natural numbers and discrete structures such as trees, graphs, computer programs etc.
- It can prove claims that might be otherwise hard to prove using other proof techniques.

Mathematical Induction

The principle of mathematical induction states that

for some property $P(n)$, if we have that

$P(n_0)$ and

$\forall k \geq n_0 \in \mathbb{N}\colon \underbrace{P(k)}_{\textit{Induction Hypothesis}} \Rightarrow P(k+1)$

then,

$\forall n \geq n_0 \in \mathbb{N}\colon P(n)$

Mathematical Induction

How induction works ***intuitively***:

Suppose $n_0 = 0$.

It is true for $n = 0$.
If it true for $n = 0$, it is true for $n = 1$.
If it true for $n = 1$, it is true for $n = 2$.
If it true for $n = 2$, it is true for $n = 3$.

…

The Domino Effect

Imagine an infinitely long line of dominos.

In order to get ***all the dominos*** to fall,

1. the ***first*** domino must fall
2. We must make sure that if ***any domino*** falls (the k^{th} one), we know ***the next one*** (the $k+1^{th}$ one) will also fall

Illustration by Courtesy of Wikipedia

Another Way of Thinking: Climbing a ladder

Imagine a ladder with an infinite number of steps.

In order to climb all the steps of the ladder without falling,

1. We must be able to climb to the ***first*** step
2. We must make sure that from an arbitrary k^{th} step, we can climb to ***the next one*** (the $k+1^{th}$ one) without falling

Mathematical Induction

Mathematical induction establishes a statement for ***natural numbers***, e.g.,

$$\forall n \geq n_0 \in \mathbb{N}:\ P(n)$$

- $P(n)$ is called a ***predicate***.
- $P(n)$ takes n as input and evaluates to either ***True*** or ***False***.
- Examples:
  - $P(n)\ \equiv 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$ for all $n \geq 1$
  - $P(n)\ \equiv\ 6^n - 1$ is divisible by 5 for all $n \geq 1$
  - $P(n)\ \equiv n < 2^n$ for all $n \geq 0$

Proof Template

Suppose $\forall n \geq n_0 \in \mathbb{N}\colon\ P(n)$ is a predicate we want to prove.

Step 0 (Preparatory Step):

We define the predicate $P(n)$ for $\forall n \geq n_0$.

Step 1 (Base Case):

We want to show that the base case is true. In other words, we must show that $P(n_0)$ holds.

Step 2 (Induction Hypothesis):

Assume that $P(k)$ holds for any integer $n = k$.

Step 3 (Inductive Step):

We must show that $P(k + 1)$ is also true.

Step 4 (Conclusion):

We can now conclude the claim $\forall n \geq n_0 \in \mathbb{N}\colon\ P(n)$ is true.

Example I: $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

Show that $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$ for all $n \geq 1$.

Preparatory Step:

Formulate the claim as a predicate:

$$P(n) \equiv \forall n \geq 1 : 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Now, we have to show that $P(n)$ holds for all $n \geq 1$using induction.

Example I: $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

Show that $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$ for all $n \geq 1$.

Base Case: $n = 1$

$L.H.S. = 1$

$R.H.S. = \dfrac{1(1+1)}{2} = 1$

Therefore, $P(0)$ is (***trivially***) true.

******NB:*** Most base cases of induction proofs are often trivially true.

Example I: $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

Show that $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$ for all $n \geq 1$.

Induction Hypothesis:

Assume true for $n = k$.

That is, $1 + 2 + 3 + \cdots + k = \frac{k(k+1)}{2}$ for any $n = k$.

Example I: $1+2+3+\cdots+n=\frac{n(n+1)}{2}$

Show that $1+2+3+\cdots+n=\frac{n(n+1)}{2}$ for all $n \geq 1$.

Inductive Step:

We have to show it is true for $n=k+1$.

That is, we have to show that $1+2+3+\cdots+k+(k+1)=\frac{(k+1)(k+2)}{2}$.

$1+2+3+\cdots+k=\frac{k(k+1)}{2}$ (I.H.)

$1+2+3+\cdots+k+(k+1)=\frac{k(k+1)}{2}+k+1$ (Adding $k+1$to both sides)

$1+2+3+\cdots+k+(k+1)=\frac{k(k+1)+2(k+1)}{2}$ (Simplifying R.H.S.)

$1+2+3+\cdots+k+(k+1)=\frac{(k+1)(k+2)}{2}$ (Simplifying R.H.S.)

We have just shown that $P(k+1)$ is true.

Conclusion: Therefore, we have shown that $\forall n \geq 1\colon\ P(n)$ ■

Example II: $5 \mid 6^n - 1$

Show that $5 \mid 6^n - 1$ for all $n \geq 1$.

Preparatory Step:

Formulate the claim as a predicate:

$$P(n) \equiv \forall n \geq 1:\ 5 \mid 6^n - 1$$

Now, we have to show that $P(n)$ holds for all $n \geq 1$using induction.

Example II: $5 \mid 6^n - 1$

Show that $5 \mid 6^n - 1$ for all $n \geq 1$.

<u>Base Case:</u> $n = 1$

$$5 \mid 6^1 - 1 = 5$$

Therefore, $P(0)$ is (***trivially***) true.

Example II: $5 \mid 6^n - 1$

Show that $5 \mid 6^n - 1$ for all $n \geq 1$.

Induction Hypothesis:

Assume true for $n = k$.

That is, $5 \mid 6^k - 1$ for any $n = k$.

Example II: $5 \mid 6^n - 1$

Show that $5 \mid 6^n - 1$ for all $n \geq 1$.

Inductive Step:

We have to show it is true for $n = k + 1$.

That is, we have to show that $5 \mid 6^{k+1} - 1$.

$5 \mid 6^k - 1$	(I.H.)
$6^k - 1 = 5q$ for some $q \in \mathbb{N}$	(Definition of being divisible by 5)
$6(6^k - 1) = 30q$	(Multiplying both sides by 6)
$6^{k+1} - 6 \ = 30q$	(Expanding L.H.S.)
$6^{k+1} - 1 - 5 \ = 30q$	($-6 = -1 - 5$)
$6^{k+1} - 1 \ = 30q + 5$	(Adding 5 to both sides)
$6^{k+1} - 1 \ = 5(6q + 1)$	($30q + 5 = 5(6q + 1)$)
$5 \mid 6^{k+1} - 1$	(Definition of being divisible by 5)

We have just shown that $P(k+1)$ is true.

Conclusion: Therefore, we have shown that $\forall n \geq 1$: $P(n)$ ■

Example III: $n < 2^n$

Show that $n < 2^n$ for all $n \geq 0$.

Preparatory Step:

Formulate the claim as a predicate:

$$P(n) \equiv \forall n \geq 0: n < 2^n$$

Now, we have to show that $P(n)$ holds for all $n \geq 0$ using induction.

Example III: $n < 2^n$

Show that $n < 2^n$ for all $n \geq 0$.

Base Case: $n = 0$

$R.H.S. = 0$

$L.H.S. = 2^0 = 1$

$R.H.S. < \ L.H.S.$

Therefore, $P(0)$ is (***trivially***) true.

Example III: $n < 2^n$

Show that $n < 2^n$ for all $n \geq 0$.

Induction Hypothesis:

Assume true for $n = k$.

That is, $k < 2^k$ for any $n = k$.

Example III: $n < 2^n$

Show that $n < 2^n$ for all $n \geq 0$.

Inductive Step:

We have to show it is true for $n = k + 1$.

That is, we have to show that $k + 1 < 2^{k+1}$.

$k < 2^k$	(I.H.)
$k + 1 < 2^k + 1$	(Adding 1 to both sides)
$2^k + 1 \leq 2^k + 2^k$	($1 \leq 2^k$ for any $k \geq 0$)
$2^k + 1 \leq 2 \cdot 2^k$	($2^k + 2^k = 2 \cdot 2^k$)
$2^k + 1 \leq 2^{k+1}$	($2 \cdot 2^k = 2^{k+1}$)
$k + 1 \quad < 2^{k+1}$	(Transitivity of $<$ and $\leq$)

We have just shown that $P(k + 1)$ is true.

Conclusion: Therefore, we have shown that $\forall n \geq 1\colon\ P(n)$ ■

Proof of $n \in O(2^n)$

We can now adapt the result of the previous induction proof to show that $n \in O(2^n)$.

Proof:

From the previous induction proof, we know that

$$n < 2^n \qquad \text{for all } n \geq 0$$

$$n \leq 2^n \qquad \text{for all } n \geq 0$$

$$n \leq 1 \cdot 2^n \qquad \text{for all } n \geq 0$$

Therefore, we can choose $n_0 = 0$ and $c = 1$ ■

Revisiting the Dominos

Let's revisit the dominos.

- The inductive approach we are using can be sometimes insufficient to prove some claims.
- So far, to show that the $k + 1^{th}$ domino falls, we must show that the k^{th} one falls.
- Actually, with the knowledge that $k + 1^{th}$ domino falls, we know much more.
  - We know that the $1^{th}, 2^{th}, \dots k^{th}$ dominos fall.
  - This variant of mathematical induction is known as ***strong induction*** or ***complete induction***.
  - The first variant we showed earlier is known as ***weak induction***.

Strong Mathematical Induction

The principle of strong mathematical induction states that

for some property $P(n)$, if we have that

$P(n_0)$ and

$$\forall k \geq n_0 \in \mathbb{N}: P(n_0) \wedge P(n_0+1) \wedge \cdots \wedge P(k) \Rightarrow P(k+1)$$

then,

$$\forall n \geq n_0 \in \mathbb{N}: P(n)$$

Induction Hypothesis

Example I: Every integer $n > 1$ can be written as a product of primes.

First Attempt (using the weak version):

Base Case: $n = 2$

2 is a prime number so it can be written as a (***trivial***) product of primes.

Induction Hypothesis:

Assume true for $n = k$. That is, an arbitrary integer $k \geq 2$ can be written as a product of primes.

Inductive Step:

We need to show that $k + 1$ can be written as a product of primes.

Example I: Every integer $n > 1$ can be written as a product of primes.

First Attempt (using the weak version):

Inductive Step:

We need to show that $k + 1$ can be written as a product of primes.

We are bound to get stuck at this step, trying to establish $\mathrm{P}(k+1)$ from $\mathrm{P}(k)$.

Why? : Because there is no obvious relation between the factorization of k and the factorization of $k + 1$.

How can we make use of knowing that $14 = 2 \times 7$ to establish the fact that $15 = 3 \times 5$?

Example I: Every integer $n > 1$ can be written as a product of primes.

Second Attempt (using the strong version):

Induction Hypothesis:

Assume true for $n = 2,3,\dots,k$. That is, an arbitrary integer $2 \leq n \leq k$ can be written as a product of primes.

In other words, we assume $P(2) \land P(3) \land \cdots \land P(k)$ is true.

Inductive Step:

We need to show that $k + 1$ can be written as a product of primes.

We split our consideration into two cases as follows.

Example I: Every integer $n > 1$ can be written as a product of primes.

Case I: $k+1$ is a prime.

This is trivially true because a prime can be trivially written as a product of primes.

Case II: $k+1$ is a composite number.

Therefore, there exist integers $1 \leq a, b \leq k$ such that

$$k + 1 = a \cdot b$$

Since $1 \leq a, b \leq k$, we can invoke our I.H.

Thus,

a and b can be written as a product of primes. (I.H.)

Thus, $k+1$ is also a product of primes.

Conclusion: Every integer $n > 1$ can be written as a product of primes.■

Example II: $n = 4a + 5b$ for all $n \geq 12$.

Preparatory Step:

Let $P(n)$ be the proposition that any integer $n = 4a + 5b$ for some $a, b \in \mathbb{N}$ for all $n \geq 12$.

Base Cases:

We have ***FOUR base cases*** to prove, namely, $P(12), P(13), P(14)$ and $P(15)$.

We will explain later why we need ***FOUR base cases towards the end of the proof.

Example II: $n = 4a + 5b$ for all $n \geq 12$.

Base Cases:

Case I: $\boldsymbol{n = 12}$

$$12 = 4(3) + 5(0)$$

Case II: $\boldsymbol{n = 13}$

$$13 = 4(2) + 5(1)$$

Case III: $\boldsymbol{n = 14}$

$$14 = 4(1) + 5(2)$$

Case IV: $\boldsymbol{n = 15}$

$$15 = 4(0) + 5(3)$$

Example II: $n = 4a + 5b$ for all $n \geq 12$.

Induction Hypothesis:

Assume the proposition is true for $12 \leq n \leq k$.

That is, $k = 4a + 5b$ for some integers a, b.

Inductive Step:

We must show it is true for $k + 1$. In other words, $k + 1 = 4c + 5d$ for some integers c, d.

Example II: $n = 4a + 5b$ for all $n \geq 12$.

Inductive Step:

$k - 3 = 4x + 5y$ (I.H.)
$k - 3 + 4 = 4x + 5y + 4$ (Adding 4 to both sides)
$k + 1 = 4x + 5y + 4$ ($-3 + 4 = 1$)
$k + 1 = 4(x + 1) + 5y$ ($4x + 4 = 4(x + 1)$)

Since $x + 1$ and y are integers, we have found $c = x + 1, d = y$.
Thus, $P(k + 1)$ is true.

Conclusion: $n = 4a + 5b$ for all $n \geq 12$. ■

Example II: $n = 4a + 5b$ for all $n \geq 12$.

Why are ***FOUR base cases*** needed?

At the start of the inductive step, we chose to invoke our induction hypothesis at $k - 3$.

To make our proof valid, we need to make sure that

$$k - 3 \geq 12$$
$$k \geq 15$$

Therefore, the result of inductive step is valid if and only if $n \geq 15$.

Therefore, we need ***separate proofs*** for $n = 12,13,14$ and 15.

$n = 15$ is included as a base case as it acts as the entry point during the inductive step.

Summary

- Mathematical induction is a very powerful tool for proving statements for natural numbers and discrete structures.
  - However, it does not provide any **intuition** as to why the result is true

- We can use other proof techniques to prove statements mathematical induction can be used to prove.
  - ,e.g., direct proof, proof by contradiction, proof by contrapositive
  - These proofs provide ***intuition and logic*** behind the result.

Recursion

A ***circular*** (aka ***recursive***) definition is often regarded as not useful.

However, in mathematics, especially in computer science, we always come across ***recursive*** definitions to define concepts in terms of themselves.

Many algorithms can be expressed ***recursively*** more naturally than ***iteratively***.

Example I: The Factorial of a Number

We can define the factorial of a number recursively as follows:

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

We can see great similarity between ***recursion*** and ***induction***. Actually, they are closely related.

- The base case of a recursive definition is akin to the base case of mathematical induction.
- The recursive case is akin to the inductive step in mathematical induction.

Example I: The Factorial of a Number

```
procedure FACTORIAL(n)
    if n = 0 then
        return 1
    else
        return n * FACTORIAL(n − 1)
```

We can analyze the time complexity of this recursive algorithm as follows:

The base case: $T(0) = c_1$

The recursive case: $T(n) = T(n-1) + c_2$

Example I: The Factorial of a Number

We can analyze the time complexity of this recursive algorithm as follows:

The base case: $T(0) = c_1$

The recursive case: $T(n) = T(n-1) + c_2$

c_1 and c_2 are constants.

Q: How do we solve the above ***recurrence relation***, i.e., express the recurrence as a ***closed-form*** formula?

A: Many ways !!!

Example I: The Factorial of a Number

One simple method is to do ***repeated substitutions*** and see the unfolding pattern:

We start with the time complexity of the outermost call:

$T(n) = T(n-1) + c_2$ ***Eq.1***

Substituting $T(n-1) = T(n-2) + c_2$ into ***Eq.1*** gives

$T(n) = (T(n-2) + c_2) + c_2$ ***Eq.2***

Substituting $T(n-2) = T(n-3) + c_2$ into ***Eq.2*** gives

$T(n) = ((T(n-3) + c_2) + c_2) + c_2$ ***Eq.3***

....

Substituting $T(n-k) = T(n-k-1) + c_2$ into ***Eq.k*** gives

$T(n) = ((T(n-k-1) + c_2) + c_2) + c_2) + \cdots)))$ ***Eq.k+1***

Example I: The Factorial of a Number

We can now see the unfolding pattern:

$$T(n) = T(n-k-1) + (k+1)c_2$$

To reach the base case and terminate the algorithm, we set $n - k - 1 = 0$.

Thus, $n = k + 1$.

$$T(n) = T(0) + nc_2$$

Since $T(0) = c_1$

$$T(n) = c_1 + nc_2 \in \Theta(n) \blacksquare$$

NB: *We can use other methods such as using the* ***generating function*** *to solve this recurrence.*

Space Complexity

What about the ***space complexity***?

Each time a recursive call is invoked, a ***stack frame*** is dynamically allocated to hold the local variables within that call.

Therefore, the space complexity $S(n)$ of a recursive algorithm is proportional to the ***maximum depth*** of recursive calls.

.

Space Complexity: The Factorial Algorithm

For the factorial algorithm, the ***maximum depth*** of recursive calls is $n + 1$. Therefore, $S(n) \propto (n + 1)$.

Looking at the algorithm, we can see that each recursive call requires $\Theta(1)$ space.

- ❑ space requirement is independent of the problem size.

Therefore, the total space complexity $S(n) = (n + 1) \cdot \Theta(1) = \Theta(n)$. ■

The Recursion Tree Method

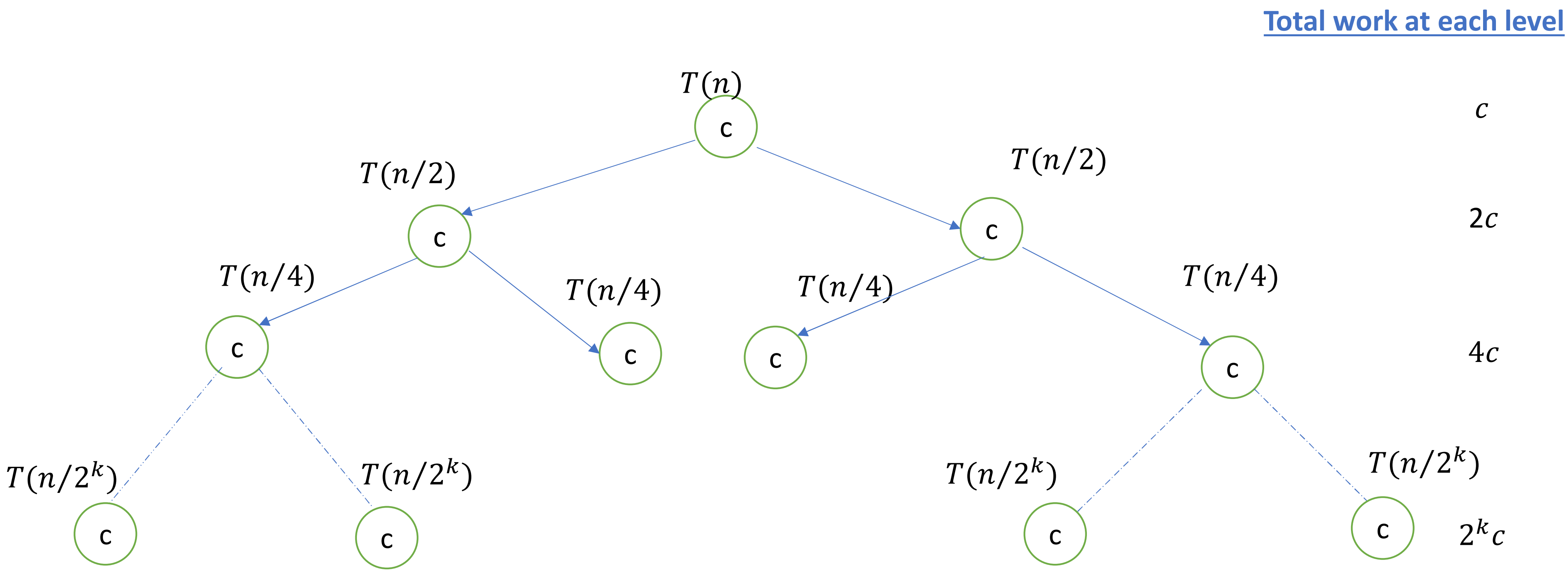
Suppose there is a recursive algorithm whose time complexity follows the following recurrence relation:

The base case: $T(1) = c$

The recursive case: $T(n) = 2 \cdot T(n/2) + c$

We can also use the ***repeated substitution*** method as we did previously.

The Recursion Tree Method



$$T(n) = c(1 + 2 + 4 + \cdots + 2^k)$$

The Recursion Tree Method

The time complexity is the sum of all the work done at every level of the recursion tree:

$$T(n) = c\left(1 + 2 + 4 + \cdots + 2^k\right) = \frac{c(1)(2^{k+1}-1)}{2-1} = c(2 \cdot 2^k - 1)$$

When the algorithm terminates, all the recursive calls at the leaves are done.

That is when the problem size $n/2^k$ reduces to 1.

$$n/2^k = 1$$

So $\quad n = 2^k$.

Therefore, $T(n) = c\left(2 \cdot 2^k - 1\right) = c(2n - 1) = \Theta(n)$ ∎

Summary

Advantages:

- Many algorithms can be naturally implemented using recursion and hence can be coded faster.
- Recursive code tends to be more concise and more easily understood.

Disavantages:

- Recursive algorithms require additional space for stack frames.
- The space complexity is proportional to the maximum depth of recursive calls.
- It incurs additional call overhead that can potentially increase running time.

More on solving recurrence relations

We will encounter situations where we need to solve recurrence relations later as we progress through the course.

Master Theorem will be introduced when we talk about the ***divide and conquer*** strategy.

Next time, we will cover ***data structures***.