CS 5114: Theory of Algorithms

Clifford A. Shaffer

Department of Computer Science
Virginia Tech
Blacksburg, Virginia

Spring 2010

Copyright © 2010 by Clifford A. Shaffer

Title page

CS5114: Theory of Algorithms

- Emphasis: Creation of Algorithms
- Less important:
  - Analysis of algorithms
  - Problem statement
  - Programming
- Central Paradigm: Mathematical Induction
  - Find a way to solve a problem by solving one or more smaller problems

CS5114: Theory of Algorithms

Creation of algorithms comes through exploration, discovery, techniques, intuition: largely by **lots** of examples and **lots** of practice (HW exercises).

We will use Analysis of Algorithms as a tool.

Problem statement (in the software eng. sense) is not important because our problems are easily described, if not easily solved. Smaller problems may or may not be the same as the original problem.

Divide and conquer is a way of solving a problem by solving one more more smaller problems.

Claim on induction: The processes of constructing proofs and constructing algorithms are similar.

Review of Mathematical Induction

- The paradigm of **Mathematical Induction** can be used to solve an enormous range of problems.
- **Purpose**: To prove a parameterized theorem of the form:
  Theorem: $\forall n \geq c, \mathbf{P}(n)$.
  - Use only positive integers $\geq c$ for n.
- Sample $\mathbf{P}(n)$:
  $n + 1 \leq n^2$

Review of Mathematical Induction

$\mathbf{P}(n)$ is a statement containing n as a variable.

This sample $\mathbf{P}(n)$ is true for $n \geq 2$, but false for $n = 1$.

Principle of Mathematical Induction

- IF the following two statements are true:
  1. $\mathbf{P}(c)$ is true.
  2. For $n > c, \mathbf{P}(n - 1)$ is true $\rightarrow \mathbf{P}(n)$ is true.
  ... **THEN** we may conclude: $\forall n \geq c, \mathbf{P}(n)$.
- The assumption "$\mathbf{P}(n - 1)$ is true" is the **induction hypothesis**.
- Typical induction proof form:
  1. Base case
  2. State induction Hypothesis
  3. Prove the implication (induction step)
- What does this remind you of?

Principle of Mathematical Induction

Important: The goal is to prove the **implication**, not the theorem! That is, prove that $\mathbf{P}(n - 1) \rightarrow \mathbf{P}(n)$. **NOT** to prove $P(n)$. This is much easier, because we can assume that $\mathbf{P}(n)$ is true.

Consider the truth table for implication to see this. Since $A \rightarrow B$ is (vacuously) true when A is false, we can just assume that A is true since the implication is true anyway A is false. That is, we only need to worry that the implication could be false if A is true.

The power of induction is that the induction hypothesis "comes for free." We often try to make the most of the extra information provided by the induction hypothesis.

This is like recursion! There you have a base case and a recursive call that must make progress toward the base case.

Induction Example 1

Theorem: Let

$$S(n) = \sum_{i=1}^{n} i = 1 + 2 + \cdots + n.$$

Then, $\forall n \geq 1$, $S(n) = \frac{n(n+1)}{2}$.

Induction Example 2

Theorem: $\forall n \geq 1, \forall$ real x such that $1 + x > 0$,
$(1 + x)^n \geq 1 + nx$.

Induction Example 3

Theorem: 2¢ and 5¢ stamps can be used to form any denomination (for denominations ≥ 4).

Colorings

4-color problem: For any set of polygons, 4 colors are sufficient to guarentee that no two adjacent polygons share the same color.

Restrict the problem to regions formed by placing (infinite) lines in the plane. How many colors do we need?
Candidates:
- 4: Certainly
- 3: ?
- 2: ?
- 1: No!

Let's try it for 2...

2010-01-25   CS 5114

Induction Example 1

Theorem: Let
$S(n) = \sum_{i=1}^{n} i = 1 + 2 + \cdots + n.$
Then, $\forall n \geq 1$, $S(n) = \frac{n(n+1)}{2}$.

Base Case: $\mathbf{P}(n)$ is true since $S(1) = 1 = 1(1+1)/2$.
Induction Hypothesis: $S(i) = \frac{i(i+1)}{2}$ for $i < n$.
Induction Step:

$$\begin{aligned} S(n) &= S(n-1) + n = (n-1)n/2 + n \\ &= \frac{n(n+1)}{2} \end{aligned}$$

Therefore, $\mathbf{P}(n-1) \rightarrow \mathbf{P}(n)$.
By the principle of Mathematical Induction,
$\forall n \geq 1$, $S(n) = \frac{n(n+1)}{2}$.
MI is often an ideal tool for **verification** of a hypothesis.
Unfortunately it does not help to construct a hypothesis.

2010-01-25   CS 5114

Induction Example 2

Induction Example 2

Theorem: $\forall n \geq 1, \forall$ real x such that $1 + x > 0$, $(1 + x)^n \geq 1 + nx$.

What do we do induction on? Can't be a real number, so must be n.
$\mathbf{P}(n) : (1 + x)^n \geq 1 + nx$.

Base Case: $(1 + x)^1 = 1 + x \geq 1 + 1x$
Induction Hypothesis: Assume $(1 + x)^{n-1} \geq 1 + (n-1)x$
Induction Step:

$$\begin{aligned} (1 + x)^n &= (1 + x)(1 + x)^{n-1} \\ &\geq (1 + x)(1 + (n-1)x) \\ &= 1 + nx - x + x + nx^2 - x^2 \\ &= 1 + nx + (n-1)x^2 \\ &\geq 1 + nx. \end{aligned}$$

2010-01-25   CS 5114

Induction Example 3

Induction Example 3

Theorem: 2¢ and 5¢ stamps can be used to form any denomination (for denominations ≥ 4).

Base case: $4 = 2 + 2$.

Induction Hypothesis: Assume $\mathbf{P}(k)$ for $4 \leq k < n$.

Induction Step:
Case 1: $n - 1$ is made up of all 2¢ stamps. Then, replace 2 of these with a 5¢ stamp.

Case 2: $n - 1$ includes a 5¢ stamp. Then, replace this with 3 2¢ stamps.

2010-01-25   CS 5114

Colorings

Colorings

4-color problem: For any set of polygons, 4 colors are sufficient to guarantee that no two adjacent polygons share the same color.

Restrict the problem to regions formed by placing (infinite) lines in the plane. How many colors do we need?
Candidates:
- 4: Certainly
- 3: ?
- 2: ?
- 1: No!

Let's try it for 2...

Induction is useful for much more than checking equations!

If we accept the statement about the general 4-color problem, then of course 4 colors is enough for our restricted version.

If 2 is enough, then of course we can do it with 3 or more.

Two-coloring Problem

Given: Regions formed by a collection of (infinite) lines in the plane.
Rule: Two regions that share an edge cannot be the same color.

Theorem: It is possible to two-color the regions formed by n lines.

Picking what to do induction on can be a problem. Lines? Regions? How can we "add a region?" We can't, so try induction on lines.
Base Case: $n = 1$. Any line divides the plane into two regions.
Induction Hypothesis: It is possible to two-color the regions formed by $n - 1$ lines.
Induction Step: Introduce the n'th line.
This line cuts some colored regions in two.
Reverse the region colors on one side of the n'th line.
A valid two-coloring results.

- Any boundary surviving the addition still has opposite colors.

- Any new boundary also has opposite colors after the switch.

Strong Induction

IF the following two statements are true:

1. **P**(c)
2. **P**$(i), i = 1, 2, \cdots, n - 1 \rightarrow$ **P**(n),

... **THEN** we may conclude: $\forall n \geq c$, **P**(n).

Advantage: We can use statements other than **P**$(n - 1)$ in proving **P**(n).

The previous examples were all very straightforward – simply add in the n'th item and justify that the IH is maintained.
Now we will see examples where we must do more sophisticated (creative!) maneuvers such as

- go backwards from n.

- prove a stronger IH.

to make the most of the IH.

Graph Problem

An **Independent Set** of vertices is one for which no two vertices are adjacent.

Theorem: Let $G = (V, E)$ be a **directed** graph. Then, G contains some independent set $S(G)$ such that every vertex can be reached from a vertex in $S(G)$ by a path of length at most 2.

Example: a graph with 3 vertices in a cycle. Pick any one vertex as $S(G)$.

It should be obvious that the theorem is true for an undirected graph.
Naive approach: Assume the theorem is true for any graph of $n - 1$ vertices. Now add the nth vertex and its edges. But this won't work for the graph $1 \leftarrow 2$. Initially, vertex 1 is the independent set. We can't add 2 to the graph. Nor can we reach it from 1.
Going forward is good for proving existance.
Going backward (from an arbitrary instance into the IH) is usually necessary to prove that a property holds in all instances. This is because going forward requires proving that you reach all of the possible instances.

Graph Problem (cont)

Theorem: Let $G = (V, E)$ be a **directed** graph. Then, G contains some independent set $S(G)$ such that every vertex can be reached from a vertex in $S(G)$ by a path of length at most 2.
Base Case: Easy if $n \leq 3$ because there can be no path of length > 2.
Induction Hypothesis: The theorem is true if $|V| < n$.
Induction Step $(n > 3)$:
Pick any $v \in V$.
Define: $N(v) = \{v\} \cup \{w \in V | (v, w) \in E\}$.
$H = G - N(v)$.
Since the number of vertices in H is less than n, there is an independent set $S(H)$ that satisfies the theorem for H.

$N(v)$ is all vertices reachable (directly) from v. That is, the Neighbors of v.
H is the graph induced by $V - N(v)$.

OK, so why remove both v and $N(v)$ from the graph? If we only remove v, we have the same problem as before. If G is $1 \rightarrow 2 \rightarrow 3$, and we remove 1, then the independent set for H must be vertex 2. We can't just add back 1. But if we remove both 1 and 2, then we'll be able to do something...

Graph Proof (cont)

There are two cases:

1. $S(H) \cup \{v\}$ is independent.
   Then $S(G) = S(H) \cup \{v\}$.
2. $S(H) \cup \{v\}$ is not independent.
   Let $w \in S(H)$ such that $(w, v) \in E$.
   Every vertex in $N(v)$ can be reached by w with path of length ≤ 2.
   So, set $S(G) = S(H)$.

By Strong Induction, the theorem holds for all G.

"$S(H) \cup \{v\}$ is not independent" means that there is an edge from something in $S(H)$ to v.
IMPORTANT: There cannot be an edge from v to $S(H)$ because whatever we can reach from v is in $N(v)$ and would have been removed in H.
We need strong induction for this proof because we don't know how many vertices are in $N(v)$.

Fibonacci Numbers

Define Fibonacci numbers inductively as:

$$
\begin{aligned}
F(1) &= F(2) = 1 \\
F(n) &= F(n-1) + F(n-2), n > 2.
\end{aligned}
$$

Theorem: $\forall n \geq 1, F(n)^2 + F(n+1)^2 = F(2n+1)$.

Induction Hypothesis:
$F(n-1)^2 + F(n)^2 = F(2n-1)$.

Expand both sides of the theorem, then cancel like terms:
$F(2n+1) = F(2n) + F(2n-1)$ and,

$$
\begin{aligned}
F(n)^2 + F(n+1)^2 &= F(n)^2 + (F(n) + F(n-1))^2 \\
&= F(n)^2 + F(n)^2 + 2F(n)F(n-1) + F(n-1)^2 \\
&= F(n)^2 + F(n-1)^2 + F(n)^2 + 2F(n)F(n-1) \\
&= F(2n-1) + F(n)^2 + 2F(n)F(n-1).
\end{aligned}
$$

Want: $F(n)^2 + F(n+1)^2 = F(2n+1) = F(2n) + F(2n-1)$
Steps above gave:
$F(2n) + F(2n-1) = F(2n-1) + F(n)^2 + 2F(n)F(n-1)$
So we need to show that: $F(n)^2 + 2F(n)F(n-1) = F(2n)$
To prove the original theorem, we must prove this. Since we must do it anyway, we should take advantage of this in our IH!

Fibonacci Numbers (cont)

With a stronger theorem comes a stronger IH!

Theorem:
$F(n)^2 + F(n+1)^2 = F(2n+1)$ and
$F(n)^2 + 2F(n)F(n-1) = F(2n)$.

Induction Hypothesis:
$F(n-1)^2 + F(n)^2 = F(2n-1)$ and
$F(n-1)^2 + 2F(n-1)F(n-2) = F(2n-2)$.

$F(n)^2 + 2F(n)F(n-1)$

$$
\begin{aligned}
&= F(n)^2 + 2(F(n-1) + F(n-2))F(n-1) \\
&= F(n)^2 + F(n-1)^2 + 2F(n-1)F(n-2) + F(n-1)^2 \\
&= F(2n-1) + F(2n-2) \\
&= F(2n).
\end{aligned}
$$

$$
\begin{aligned}
F(n)^2 + F(n+1)^2 &= F(n)^2 + [F(n) + F(n-1)]^2 \\
&= F(n)^2 + F(n)^2 + 2F(n)F(n-1) + F(n-1)^2 \\
&= F(n)^2 + F(2n) + F(n-1)^2 \\
&= F(2n-1) + F(2n) \\
&= F(2n+1).
\end{aligned}
$$

... which proves the theorem. The original result could not have been proved without the stronger induction hypothesis.

Another Example

Theorem: All horses are the same color.

Proof: $\mathbf{P}(n)$: If S is a set of n horses, then all horses in S have the same color.
Base case: $n = 1$ is easy.
Induction Hypothesis: Assume $\mathbf{P}(i), i < n$.
Induction Step:
- Let S be a set of horses, $|S| = n$.
- Let S' be $S - \{h\}$ for some horse h.
- By IH, all horses in S' have the same color.
- Let h' be some horse in S'.
- IH implies $\{h, h'\}$ have all the same color.

Therefore, $\mathbf{P}(n)$ holds.

The problem is that the base case does not give enough strength to give the **particular** instance of $n = 2$ used in the last step.

Algorithm Analysis

- We want to "measure" algorithms.
- What do we measure?

- What factors affect measurement?

- Objective: Measures that are independent of all factors except input.

2010-01-25   CS 5114

└─ Algorithm Analysis

- We want to "measure" algorithms.
- What do we measure?
- What factors affect measurement?
- Objective: Measures that are independent of all factors except input.

What do we measure?
Time and space to run; ease of implementation (this changes with language and tools); code size

What affects measurement?
Computer speed and architecture; Programming language and compiler; System load; Programmer skill; Specifics of input (size, arrangement)

If you compare two programs running on the same computer under the same conditions, all the other factors (should) cancel out.
Want to measure the relative efficiency of two algorithms without needing to implement them on a real computer.

Time Complexity

- Time and space are the most important computer resources.
- Function of input: **T**(input)
- Growth of time with size of input:
  - Establish an (integer) **size** n for inputs
  - n numbers in a list
  - n edges in a graph
- Consider time for all inputs of size n:
  - Time varies widely with specific input
  - Best case
  - Average case
  - Worst case
- Time complexity **T**(n) counts **steps** in an algorithm.

Sometimes analyze in terms of more than one variable.
Best case usually not of interest.
Average case is usually what we want, but can be hard to measure.
Worst case appropriate for "real-time" applications, often best we can do in terms of measurement.
Examples of "steps:" comparisons, assignments, arithmetic/logical operations. What we choose for "step" depends on the algorithm. Step cost must be "constant" – not dependent on n.

Asymptotic Analysis

- It is undesirable/impossible to count the exact number of steps in most algorithms.
  - Instead, concentrate on main characteristics.
- Solution: Asymptotic analysis
  - Ignore small cases:
    - Consider behavior approaching infinity
  - Ignore constant factors, low order terms:
    - $2n^2$ looks the same as $5n^2 + n$ to us.

Undesirable to count number of machine instructions or steps because issues like processor speed muddy the waters.

O Notation

O notation is a measure for "upper bound" of a growth rate.
- pronounced "Big-oh"

Definition: For **T**(n) a non-negatively valued function, **T**(n) is in the set $O(f(n))$ if there exist two positive constants c and n_0 such that **T**$(n) \leq cf(n)$ for all $n > n_0$.

Examples:
- $5n + 8 \in O(n)$
- $2n^2 + n \log n \in O(n^2) \in O(n^3 + 5n^2)$
- $2n^2 + n \log n \in O(n^2) \in O(n^3 + n^2)$

Remember: The time equation is for some particular set of inputs – best, worst, or average case.
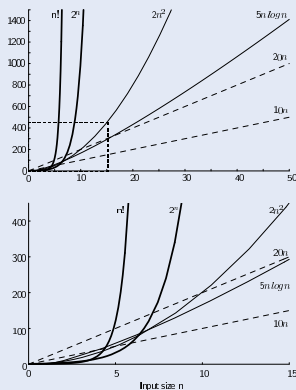
O Notation (cont)

We seek the "simplest" and "strongest" f.

Big-O is somewhat like "$\leq$":
  $n^2 \in O(n^3)$ and $n^2 \log n \in O(n^3)$, but
- $n^2 \neq n^2 \log n$
- $n^2 \in O(n^2)$ while $n^2 \log n \notin O(n^2)$

2010-01-25   CS 5114

└─O Notation (cont)

O Notation (cont)

We seek the "simplest" and "strongest" f.

Big-O is somewhat like "$\leq$":
$n^2 \in O(n^3)$ and $n^2 \log n \in O(n^3)$, but
$n^2 \neq n^2 \log n$
$n^2 \in O(n^2)$ while $n^2 \log n \notin O(n^2)$

A common misunderstanding:

- "The best case for my algorithm is $n = 1$ because that is the fastest." WRONG!

- Big-oh refers to a growth rate as n grows to ∞.

- Best case is defined for the input of size n that is cheapest among all inputs of size n.

Growth Rate Graph
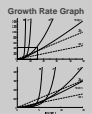
2^n is an exponential algorithm. $10n$ and $20n$ differ only by a constant.

Speedups

What happens when we buy a computer 10 times faster?

$\mathbf{T}(n)$	n	n'	Change	n'/n
$10n$	1,000	10,000	$n' = 10n$	10
$20n$	500	5,000	$n' = 10n$	10
$5n \log n$	250	1,842	$\sqrt{10}n < n' < 10n$	7.37
$2n^2$	70	223	$n' = \sqrt{10}n$	3.16
2^n	13	16	$n' = n + 3$	$--$

n: Size of input that can be processed in one hour (10,000 steps).

n': Size of input that can be processed in one hour on the new machine (100,000 steps).

How much speedup? 10 times. More important: How much increase in problem size for same time? Depends on growth rate.
For n^2, if $n = 1000$, then n' would be 1003.
Compare $\mathbf{T}(n) = n^2$ to $\mathbf{T}(n) = n \log n$. For $n > 58$, it is faster to have the $\Theta(n \log n)$ algorithm than to have a computer that is 10 times faster.

Some Rules for Use

Definition: f is **monotonically growing** if $n_1 \geq n_2$ implies $f(n_1) \geq f(n_2)$.
We typically assume our time complexity function is monotonically growing.

Theorem 3.1: Suppose f is monotonically growing.
$\forall c > 0$ and $\forall a > 1, (f(n))^c \in O(a^{f(n)})$
In other words, an **exponential** function grows faster than a **polynomial** function.

Lemma 3.2: If $f(n) \in O(s(n))$ and $g(n) \in O(r(n))$ then
- $f(n) + g(n) \in O(s(n) + r(n)) \equiv O(\max(s(n), r(n)))$
- $f(n)g(n) \in O(s(n)r(n))$.
- If $s(n) \in O(h(n))$ then $f(n) \in O(h(n))$
- For any constant k, $f(n) \in O(ks(n))$

Assume monitonic growth because larger problems should take longer to solve. However, many real problems have "cyclically growing" behavior.
Is $O(2^{f(n)}) \in O(3^{f(n)})$? Yes, but not vice versa.
$3^n = 1.5^n \times 2^n$ so no constant could ever make 2^n bigger than 3^n for all n. functional composition

Other Asymptotic Notation

$\Omega(f(n))$ – lower bound ($\geq$)
Definition: For $\mathbf{T}(n)$ a non-negatively valued function, $\mathbf{T}(n)$ is in the set $\Omega(g(n))$ if there exist two positive constants c and n_0 such that $\mathbf{T}(n) \geq cg(n)$ for all $n > n_0$.
Ex: $n^2 \log n \in \Omega(n^2)$.

$\Theta(f(n))$ – Exact bound ($=$)
Definition: $g(n) = \Theta(f(n))$ if $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$.
Important!: It is Θ if it is both in big-Oh and in Ω.
Ex: $5n^3 + 4n^2 + 9n + 7 = \Theta(n^3)$

Other Asymptotic Notation (cont)

$o(f(n))$ – little o ($<$)
Definition: $g(n) \in o(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
Ex: $n^2 \in o(n^3)$

$\omega(f(n))$ – little omega ($>$)
Definition: $g(n) \in w(f(n))$ if $f(n) \in o(g(n))$.
Ex: $n^5 \in w(n^2)$

$\infty(f(n))$
Definition: $T(n) = \infty(f(n))$ if $T(n) = O(f(n))$ but the constant in the O is so large that the algorithm is impractical.

A common misunderstanding:

- Confusing worst case with upper bound.

- Upper bound refers to a growth rate.

- Worst case refers to the worst input from among the choices for possible inputs of a given size.

We won't use these too much.