CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 3

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Introduction

In this lecture we look at more examples of induction especially in the design and analysis of algorithms.

2 Examples of Mathematical Induction

2.1 A Simple Inequality

Theorem 1 *For all natural numbers* n *we have* $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^n} < 1$.

Proof: The proof is by induction on n. The theorem is trivially true for $n = 1$. We assume the inequality for n and prove for $n + 1$. By induction hypothesis we know that the sum of the first n terms is less than 1. However on adding the $\frac{1}{2^{n+1}}$ the sum might exceed 1. The **trick** is to use induction in a different order. Look at the last n terms of the summation: we have $\frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^n} + \frac{1}{2^{n+1}} = \frac{1}{2}\left(\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n}\right) < \frac{1}{2}$ due to the induction hypothesis. Now we add $\frac{1}{2}$ to both sides to obtain the desired result. ■

2.2 Arithmetic-Geometric Mean Theorem

Theorem 2 *If* $x_1, x_2, \ldots, x_n$ *are all positive reals then we have* $(x_1 x_2 \ldots x_n)^{\frac{1}{n}} \leq \frac{x_1 + x_2 + \ldots + x_n}{n}$.

Proof: The proof is by induction on n. The induction hypothesis is the same as the statement, however it different to the one that we are doing till now: we will use the principle of **reversed induction**.

Principle of Reversed Induction
If a statement P is true for an infinite subset of natural numbers and if its truth for n implies its truth for $n-1$, then P is true for all natural numbers.

The principle of reverse induction is correct since it holds for an infinite set, for every natural number k, there is a number $m > k$ in the set and we can use reverse induction step to go backwards from m to k.

First we show it is correct for an infinite subset (all powers of 2) using regular induction. For $n = 1$ it is trivial and for $n = 2$, we have $\sqrt{x_1x_2} \leq \frac{x_1+x_2}{2}$ which is correct since $(x_1 - x_2)^2 \geq 0$. Now we assume it is true for $n = 2^k$ and we now prove it for $2n = 2^{k+1}$. Let $y_1 = (x_1x_2 \dots x_n)^{\frac{1}{n}}$ and $y_2 = (x_{n+1}x_{n+2} \dots x_{2n})^{\frac{1}{n}}$. Then we have $(x_1x_2 \dots x_n \dots x_{2n})^{\frac{1}{2n}} = \sqrt{y_1y_2} \leq \frac{y_1+y_2}{2}$. Now induction hypothesis for n gives $y_1 \leq \frac{x_1+x_2+\ldots+x_n}{n}$ and $y_2 \leq \frac{x_{n+1}+x_{n+2}+\ldots+x_{2n}}{n}$. So we have $\frac{y_1+y_2}{2} \leq \frac{1}{2}(\frac{x_1+x_2+\ldots+x_n+\ldots+x_{2n}}{n})$ which is what we wanted to prove.

Now we use reverse induction to prove the theorem for all n. We assume the statement is true for n and prove it for $n-1$. Let $z = \frac{x_1+x_2+\ldots+x_{n-1}}{n-1}$. We apply the statement for the n numbers $x_1, x_2, \ldots, x_{n-1}, z$ to get $(x_1x_2 \dots x_{n-1}z)^{\frac{1}{n}} \leq \frac{x_1+x_2+\ldots+x_{n-1}+z}{n} = \frac{(n-1)z+z}{n} = z$ which implies $(x_1x_2 \dots x_{n-1}z) \leq z^n$. This gives $(x_1x_2 \dots x_{n-1}) \leq z^{n-1}$ which gives $(x_1x_2 \dots x_{n-1})^{\frac{1}{n-1}} \leq z = \frac{x_1+x_2+\ldots+x_{n-1}}{n-1}$ which is what we wanted to show. ■

2.3 Loop Invariants

Induction is a very good tool for proving the correctness of algorithms. Assume a program has a loop that is supposed to compute a certain value. We can use induction on the number of times this loop is executed to prove that the result is correct. An induction hypothesis which reflects the relationships between the variables during the loop execution is called a **loop invariant**.

2.4 Converting a number to its binary representation

Algorithm 1 Convert-To-Binary(n)

Input: A positive integer n.
Output: An array of bits b corresponding to the binary representation of n.

```
1: t:=n (a new variable to preserve $n$)
2: k=0;
3: while $t > 0$ do
4:     $k = k + 1$
5:     $b[k] = t \bmod 2$
6:     $t = \lfloor \frac{t}{2} \rfloor$
7: end while
```

Theorem 3 *When Algorithm 1 terminates it stores the binary representation of* n *in the array* b.

Proof: The proof is by induction on k, the number of time the loop is executed. In this case the induction hypothesis is the loop invariant. The Induction Hypothesis is "If m is the integer represented by the binary array $b[12\ldots k]$, then $n = 2^k t + m$". Intuitively it says that at step k of the loop, the binary array represents the k least significant bits of n, and that the value of t, when shifted by k corresponds to the rest of the bits.

To prove the correctness of the algorithm we need to show

1. The hypothesis is true at the beginning (basis).
2. The truth of the hypothesis at step k implies the truth of the hypothesis at step $k+1$.
3. When the loop terminates the induction hypothesis implies the correctness of the algorithm.

The first point is easy to show: $k = 0 = m$ (since the array is empty by definition) and $t = n$. Thus $n = 2^0 t + 0$. For the third point we note that $t = 0$ and thus $n = 2^k 0 + m = m$. Finally for the second point we consider two cases for the start of the k^{th} loop:

- If t is even, then t mod $2 = 0$ and thus there is no change to the array, t gets divided by 2 and k is incremented, i.e., $n = \frac{t}{2}2^{k+1} + m = 2^k t + m$.
- If t is odd, then $b[k+1]$ is set to 1, which contributes 2^k to m, t is changed to $\frac{t-1}{2}$ and k is incremented. So the expression is $\frac{t-1}{2}2^{k+1} + m + 2^k = (t-1)2^k + m + 2^k = 2^k t + m = n$ as desired.

■

Read about the **common errors** is Section 2.13 of the book [1] to avoid them.

3 Design of Algorithms using Induction

So far we have seen the use of induction in the proof of theorems and showing correctness of algorithms. In a sense, the induction idea gives us **recursive** algorithms.

3.1 Finding one-to-one mappings

Let f be a function that maps a finite set $A = \{1, 2, \ldots, n\}$ to itself. Assume f is represented by an array $f[12\ldots n]$ such that $f[i]$ holds the value of $f(i)$ which is an integer between 1 and n. We call f a **one-to-one** function if for every element j, there is at most one element i that is mapped to j. Consider an example of a function in Figure 1.

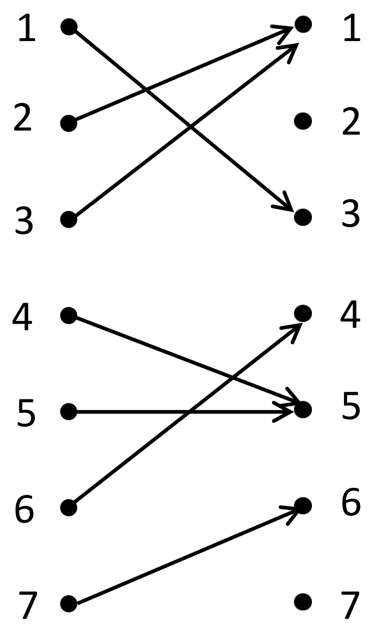

Figure 1:

The Problem: Given a finite set A and a mapping $f : A \to A$ find a subset $S \subseteq A$ with maximum number of elements such that

1. The function f maps every element of S to some other element within S itself.
2. No two elements of S are mapped to the same element, i.e., f is one-to-one when restricted to S.

Algorithm: If f is originally one-to-one then taking $S = A$ we are done. If on the other hand $f(i) = f(j)$ for some $i \neq j$ then S cannot contain both i and j. We can try all subsets $S \subseteq A$, but the running time is $2^n n$ which is exponential. We want a more efficient algorithm. For example, in Figure 1 we have $f(2) = 1 = f(3)$. So S cannot contain both 2 and 3. The choice between 2 and 3 is important: if we eliminate 3, then 1 is eliminated and then 2 is eliminated. However if we instead eliminate 2 only then we obtain $\{1, 3\}$ as a good set.

So reducing the problem to a smaller one is important and non-trivial. We can use induction hypothesis: We know how to solve the problem for sets with $n - 1$ elements. The base case is trivial: if there is only one element in the set then it must be mapped to itself. But the induction hypothesis is non-trivial when we have a choice (e.g. 2 and 3 in Figure 1). The key idea is that the element i that has no element mapped to it cannot belong to S and thus we must remove it. This is because $i \in S$ and $|S| = k$ implies that the k elements of S must be mapped to at most $k - 1$ elements of S and hence f cannot be one-to-one when restricted to S. So we remove $\{i\}$ from A and iterate. Note that the reduced problem is exactly the same (except the size) as the original problem. We have to be careful though: the only condition we had previously was that f maps A to itself. This condition is still maintained for the set $A \setminus \{i\}$. By this algorithm we need to consider only n elements instead of 2^n choices for S. Thus this algorithm is much more efficient.

See Figure 5.3 in the book [1] for the algorithm described above.

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*