Towers of Hanoi

Problem: Move all the rings from pole 1 and pole 2, moving one ring at a time, and never having a larger ring on top of a smaller one.

How do we solve this?

- Think recursively!
- Suppose you could solve it for $n-1$ rings? How could you do it for n?

Solution

- Move top $n-1$ rings from pole 1 to pole 3 (we can do this by assumption)
  - Pretend largest ring isn't there at all
- Move largest ring from pole 1 to pole 2
- Move top $n-1$ rings from pole 3 to pole 2 (we can do this by assumption)
  - Again, pretend largest ring isn't there

This solution translates to a recursive algorithm:

- Suppose robot($r \rightarrow s$) is a command to a robot to move the top ring on pole r to pole s
- Note that if $r, s \in \{1, 2, 3\}$, then $6-r-s$ is the other number in the set

```
procedure H(n, r, s)            [Move n disks from r to s]
    if n = 1 then robot(r → s)
              else H(n − 1, r, 6 − r − s)
                   robot(r → s)
                   H(n − 1, 6 − r − s, s)
    endif
    return
endpro
```

Tree of Calls

Suppose there are initially three rings on pole 1, which we want to move to pole 2:

Analysis of Algorithms

For a particular algorithm, we want to know:

- How much time it takes
- How much space it takes

What does that mean?

- In general, the time/space will depend on the input size
  - The more items you have to sort, the longer it will take
- Therefore want the answer as a function of the input size
  - What is the best/worst/average case as a function of the input size.

Given an algorithm to solve a problem, may want to know if you can do better.

- What is the *intrinsic complexity* of a problem?

This is what *computational complexity* is about.

Euclidean Algorithm: Analysis

Input m, n [m, n natural numbers, $m \geq n$]
  $num \leftarrow m$; $denom \leftarrow n$ [Initialize num and $denom$]
  repeat until $denom = 0$
    $q \leftarrow \lfloor num/denom \rfloor$
    $rem \leftarrow num - (q * denom)$
    $num \leftarrow denom$ [New num]
    $denom \leftarrow rem$ [New $denom$; note $num \geq denom$]
  endrepeat
Output num [$num = \gcd(m, n)$]

How many times do we go through the loop in the Euclidean algorithm:

- Best case: Easy. Never!
- Average case: Too hard
- Worst case: Can't answer this exactly, but we can get a good upper bound.
  - See how fast *denom* goes down in each iteration.

Claim: After two iterations, *denom* is halved:

- Recall $num = q * denom + rem$. Use $denom'$ and $denom''$ to denote value of *denom* after 1 and 2 iterations. Two cases:
  1. $rem \leq denom/2 \Rightarrow denom' \leq denom/2$ and $denom'' < denom/2$.
  2. $rem > denom/2$. But then $num' = denom$, $denom' = rem$. At next iteration, $q = 1$, and $rem' = num' - denom' < denom/2$
- How long until *denom* is ≤ 1?
  - $< 2\log_2(m)$ steps!
- After at most $2\log_2(m)$ steps, $denom = 0$.

Towers of Hanoi: Analysis

procedure H(n, r, s) [Move n disks from r to s]
  if $n = 1$ **then** robot$(r \to s)$
    else $H(n-1, r, 6-r-s)$
      robot$(r \to s)$
      $H(n-1, 6-r-s, s)$
  endif
  return
endpro

Let $h_n = \#$ moves to move n rings from pole r to pole s.

- Clearly $h_1 = 1$
- Algorithm shows that $h_n = 2h_{n-1} + 1$
  - $h_2 = 3$; $h_3 = 7$; $h_4 = 15$; $\ldots$
  - $h_n = 2^n - 1$

We'll prove this formally later, when we also show that this is optimal.

Sequential Search: Analysis

Suppose we have a linked list — a sequence of words in alphabetical order. Given a new word, we want to determine if it's on the list, and where.

Input n [number of words in list]
$w_1, \ldots, w_n$ [alphabetized list]
w [search word]

Algorithm SeqSearch
$i \leftarrow 1$
repeat until $i > n$ or $w \le w_i$
$i \leftarrow i + 1$
end repeat

if $w = w_i$ **then** print i **else** print 'failure' **endif**

How many times do we go through the loop?

- Best case: 0
- Worst case: n
- Average case: roughly $n/2$ if w is on the list.

Binary Search: Analysis

Sequential search is terrible for finding a word in a dictionary. Can do much better with random access.

- it's like playing 20 questions — cut the search space in half with each question!

Input n [number of words in list]
$w_1, \ldots, w_n$ [alphabetized list]
w [search word]

Algorithm BinSearch
$F \leftarrow 1$; $L \leftarrow n$ [Initialize range]
$i \leftarrow \lfloor (F+L)/2 \rfloor$
repeat until $w = w_i$ or $F > L$
  if $w < w_i$ **then** $L \leftarrow i-1$ **else** $F \leftarrow i+1$ **endif**
  $i \leftarrow \lfloor (F+L)/2 \rfloor$
end repeat
if $w = w_i$ **then** print i **else** print 'failure' **endif**

How many times do we go through the loop?

- Best case: 0
- Average case: too hard for us
- Worst case: $\lfloor \log_2(n) \rfloor + 1$
  - After each loop iteration, $F - L$ is halved.

Methods of Proof

One way of proving things is by induction.

- That's coming next.

What if you can't use induction?

Typically you're trying to prove a statement like "Given X, prove (or show that) Y". This means you have to prove

$$X \Rightarrow Y$$

In the proof, you're allowed to assume X, and then show that Y is true, using X.

- A special case: if there is no X, you just have to prove Y or $\mathit{true} \Rightarrow Y$.

Alternatively, you can do a *proof by contradiction*: Assume that Y is false, and show that X is false.

- This amounts to proving

$$\neg Y \Rightarrow \neg X$$

Example

Theorem n is odd iff n^2 is odd, for $n \in N^+$.

Proof: We have to show

1. n odd $\Rightarrow n^2$ odd
2. n^2 odd $\Rightarrow n$ odd

For (1), if n is odd, it is of the form $2k+1$. Hence,

$$n^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$$

Thus, n^2 is odd.

For (2), we proceed by contradiction. Suppose n^2 is odd and n is even. Then $n = 2k$ for some k, and $n^2 = 4k^2$. Thus, n^2 is even. This is a contradiction. Thus, n must be odd.

A Proof By Contradiction

Theorem: $\sqrt{2}$ is irrational.

Proof: By contradiction. Suppose $\sqrt{2}$ is rational. Then $\sqrt{2} = a/b$ for some $a, b \in N^+$. We can assume that a/b is in lowest terms.

- Therefore, a and b can't both be even.

Squaring both sides, we get

$$2 = a^2/b^2$$

Thus, $a^2 = 2b^2$, so a^2 is even. This means that a must be even.

Suppose $a = 2c$. Then $a^2 = 4c^2$.

Thus, $4c^2 = 2b^2$, so $b^2 = 2c^2$. This means that b^2 is even, and hence so is b.

Contradiction!

Thus, $\sqrt{2}$ must be irrational.

Induction

This is perhaps the most important technique we'll learn for proving things.

Idea: To prove that a statement is true for all natural numbers, show that it is true for 1 (*base case* or *basis step*) and show that if it is true for n, it is also true for $n+1$ (*inductive step*).

- The base case does not have to be 1; it could be 0, 2, 3, ...
- If the base case is k, then you are proving the statement for all $n \geq k$.

It is sometimes quite difficult to formulate the statement to prove.

IN THIS COURSE, I WILL BE VERY FUSSY ABOUT THE FORMULATION OF THE STATEMENT TO PROVE. YOU MUST STATE IT VERY CLEARLY. I WILL ALSO BE PICKY ABOUT THE FORM OF THE INDUCTIVE PROOF.

Writing Up a Proof by Induction

1. State the hypothesis very clearly:
    - Let $P(n)$ be the statement … [some statement involving n]
2. The basis step
    - $P(k)$ holds because … [where k is the base case, usually 0 or 1]
3. Inductive step
    - Assume $P(n)$. We prove $P(n+1)$ holds as follows … Thus, $P(n) \Rightarrow P(n+1)$.
4. Conclusion
    - Thus, we have shown by induction that $P(n)$ holds for all $n \geq k$ (where k was what you used for your basis step). [It's not necessary to always write the conclusion explicitly.]

A Simple Example

Theorem: For all positive integers n,

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

Proof: By induction. Let $P(n)$ be the statement

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

Basis: $P(1)$ asserts that $\Sigma_{k=1}^{1} k = \frac{1(1+1)}{2}$. Since the LHS and RHS are both 1, this is true.

Inductive step: Assume $P(n)$. We prove $P(n+1)$.

$$\begin{aligned} \Sigma_{k=1}^{n+1} k &= \Sigma_{k=1}^{n} k + (n+1) \\ &= \tfrac{n(n+1)}{2} + (n+1) \text{[Induction hypothesis]} \\ &= \tfrac{n(n+1)+2(n+1)}{2} \\ &= \tfrac{(n+1)(n+2)}{2} \end{aligned}$$

Thus, $P(n)$ implies $P(n+1)$, so the result is true by induction.

Notes:

- You can write $\overset{P(n)}{=}$ instead of writing "Induction hypothesis" at the end of the line, or you can write "$P(n)$" at the end of the line.
  - Whatever you write, make sure it's clear when you're applying the induction hypothesis
- Notice how we rewrite $\Sigma_{k=1}^{n+1} k$ so as to be able to appeal to the induction hypothesis. This is standard operating procedure.

Another example

Theorem: $(1+x)^n \geq 1+nx$ for all nonnegative integers n and all $x \geq 0$.

Proof: By induction on n. Let $P(n)$ be the statement $(1+x)^n \geq 1+nx$.

Basis: $P(0)$ says $(1+x)^0 \geq 1$. This is clearly true.

Inductive Step: Assume $P(n)$. We prove $P(n+1)$.

$$
\begin{aligned}
(1+x)^{n+1} &= (1+x)^n(1+x) \\
&\geq (1+nx)(1+x) \text{[Induction hypothesis]} \\
&= 1+nx+x+nx^2 \\
&= 1+(n+1)x+nx^2 \\
&\geq 1+(n+1)x
\end{aligned}
$$

Euclidean Algorithm: Worst Case

This time, we'll do a formal analysis. Suppose we start the algorithm with inputs m and n. Let $denom_0$, $denom_1$, ... be the values of $denom$ on successive iterations of the loop. Similary num_0, num_1, ...

Lemma 1: For all natural numbers n, $denom_{n+2} \leq denom_n/2$.

Proof: We don't need induction here. The proof we did before works. (We actually proved $<$, but it's easier to use $\leq$.)

Lemma 2: For all natural numbers n, $denom_{2n} \leq denom_0/2^n$.

By induction. Let $P(n)$ be the statement $denom_{2n} \leq denom_0/2^n$.

Basis: $denom_0 \leq denom_0/1$.

Inductive Step: Assume $P(n)$.

$$
\begin{aligned}
denom_{2(n+1)} &= denom_{2n+2} \\
&\leq denom_{2n}/2 && \text{[Lemma 1]} \\
&\leq denom_0/(2^n \times 2) && \text{[Induction Hypothesis]} \\
&= denom_0/2^{n+1}
\end{aligned}
$$

Lemma 3: On input (m, n), we go through the loop at most $2\lceil \log_2(n) \rceil + 1$ times.

Proof:

$$denom_{2(\lceil \log_2 n \rceil)} \leq denom_0/2^{\lceil \log_2 n \rceil} \leq n/n = 1$$

(Recall $2^{\log_2 n} = n$, so $2^{\lceil \log_2 n \rceil} \geq n$.)

Thus, $denom_{2\lceil \log_2 n \rceil + 1} = 0$.

Towers of Hanoi

Theorem: It takes $2^n - 1$ moves to perform $H(n, r, s)$, for all positive n, and all $r, s \in \{1, 2, 3\}$.

Proof: Let $P(n)$ be the statement "It takes 2^n-1 moves to perform $H(n, r, s)$ and all $r, s \in \{1, 2, 3\}$."

- Note that "for all positive n" is not part of $P(n)$!
- $P(n)$ is a statement about a particular n.
- If it were part of $P(n)$, what would $P(1)$ be?

Basis: $P(1)$ is immediate: robot($r \leftarrow s$) is the only move in $H(1, r, s)$, and $2^1 - 1 = 1$.

Inductive step: Assume $P(n)$. To perform $H(n+1, r, s)$, we first do $H(n, r, 6 - r - s)$, then robot($r \leftarrow s$), then $H(n, 6 - r - s, s)$. Altogether, this takes $2^n - 1 + 1 + 2^n - 1 = 2^{n+1} - 1$ steps.

A Matching Lower Bound

Theorem: Any algorithm to move n rings from pole r to pole s requires at least $2^n - 1$ steps.

Proof: By induction, taking the statement of the theorem to be $P(n)$.

Basis: Easy: Clearly it requires (at least) 1 step to move 1 ring from pole r to pole s.

Inductive step: Assume $P(n)$. If we want to move $n+1$ rings from r to s, at some point we have to move the largest ring. At this point, the pole we want to move the largest ring to must be clear, and all the other n rings must be on the third pole. Thus, by the induction hypothesis, $2^n - 1$ moves were used to get them there.

Now we're going to need at least $2^n - 1$ moves to move the n rings back on top of the largest ring. This means we need at least

$$2^n - 1 + 1 + 2^n - 1 = 2^{n+1} - 1 \text{ steps.}$$

Strong Induction

Sometimes when you're proving $P(n+1)$, you want to be able to use $P(j)$ for $j < n$, not just $P(n)$. You can do this with *strong induction*.

1. Let $P(n)$ be the statement ... [some statement involving n]
2. The basis step
   - $P(k)$ holds because ... [where k is the base case, usually 0 or 1]
3. Inductive step
   - Assume $P(k), \ldots, P(n)$ holds. We show $P(n+1)$ holds as follows ...

Although strong induction looks stronger than induction, it's not. Anything you can do with strong induction, you can also do with regular induction, by appropriately modifying the induction hypothesis.

- If $P(n)$ is the statement you're trying to prove by strong induction, let $P'(n)$ be the statement $P(1), \ldots, P(n)$ hold. Proving $P'(n)$ by regular induction is the same as proving $P(n)$ by strong induction.

An example using strong induction

Theorem: Any item costing $n > 7$ kopecks can be bought using only 3-kopeck and 5-kopeck coins.

Proof: Using strong induction. Let $P(n)$ be the statement that n kopecks can be paid using 3-kopeck and 5-kopeck coins, for $n \geq 8$.

Basis: $P(8)$ is clearly true since $8 = 3 + 5$.

Inductive step: Assume $P(8), \ldots, P(n)$ is true. We want to show that $P(n+1)$. If $n+1$ is 9 or 10, then it's easy to see that there's no problem ($P(9)$ is true since $9 = 3 + 3 + 3$, and $P(10)$ is true since $10 = 5 + 5$). Otherwise, note that $(n + 1) - 3 = n - 2 \geq 8$. Thus, $P(n - 2)$ is true, using the induction hypothesis. This means we can use 3- and 5-kopeck coins to pay for something costing $n-2$ kopecks. One more 3-kopeck coin pays for something costing $n + 1$ kopecks.