CS215 DISCRETE MATHEMATICS FOR COMPUTER SCIENCE

Dr. QI WANG

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Zero knowledge proofs

- S. Goldwasser, S. Micali, C. Rackoff, STOC'85

The Knowledge Complexity of Interactive Proof-Systems

(Extended Abstract)

Shafi Goldwasser, MIT
Silvio Micali, MIT
Charles Rackoff, University of Toronto

Zero knowledge proofs

- S. Goldwasser, S. Micali, C. Rackoff, STOC'85

The Knowledge Complexity of Interactive Proof-Systems

(Extended Abstract)

Shafi Goldwasser
MIT

Silvio Micali
MIT

Charles Rackoff
University of Toronto

Shafi, with Micali (and later Rackoff) [6], had been thinking for a while about expanding the traditional notion of "proof" to an interactive process in which a "prover" can convince a probabilistic "verifier" of the correctness of a mathematical proposition with overwhelming probability if and only if the proposition is correct. They called this interactive process an "interactive proof" (a name suggested by Mike Sipser). They wondered if one could prove some non-trivial statement (for example, membership of a string in a hard language) without giving away any knowledge whatsoever about why it was true. They defined that the verifier receives no knowledge from the prover if the verifier could simulate on his own the probability distribution that he obtains in interacting with the prover.The idea that "no knowledge" means simulatability was a very important contribution. They also gave the first example of these "zero knowledge interactive proofs" using quadratic residuosity. This paper won the first ACM SIGACT Gödel Prize. This zero-knowledge work led to a huge research program in the community that continues to this day, including results showing that (subject to an assumption such as the existence of one-way functions) a group of distrusting parties can compute a function of all their inputs without learning any knowledge about other people's inputs beyond that which follows from the value of the function.

`https://amturing.acm.org/award_winners/goldwasser_8627889.cfm`

Applications of ZKPs

- *Protocol design*. A *protocol* is an algorithm for interactive parties to achieve a certain goal. However, in crypto, we often want to design protocols that should achieve security even when one of the parties is "cheating". Alice can prove in *zero knowledge* that she followed the instructions.

Applications of ZKPs

- *Protocol design*. A *protocol* is an algorithm for interactive parties to achieve a certain goal. However, in crypto, we often want to design protocols that should achieve security even when one of the parties is "cheating". Alice can prove in *zero knowledge* that she followed the instructions.

Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design

(Extended Abstract)

Oded Goldreich
Dept. of Computer Sc.
Technion
Haifa, Israel

Silvio Micali
Lab. for Computer Sc.
MIT
Cambridge, MA 02139

Avi Wigderson
Inst. of Math. and CS
Hebrew University
Jerusalem, Israel

Applications of ZKPs

- *Identification scheme*. How should Alice prove to Bob that she is who she claimed to be? For example, how to design a control access system to the CSE dept.?

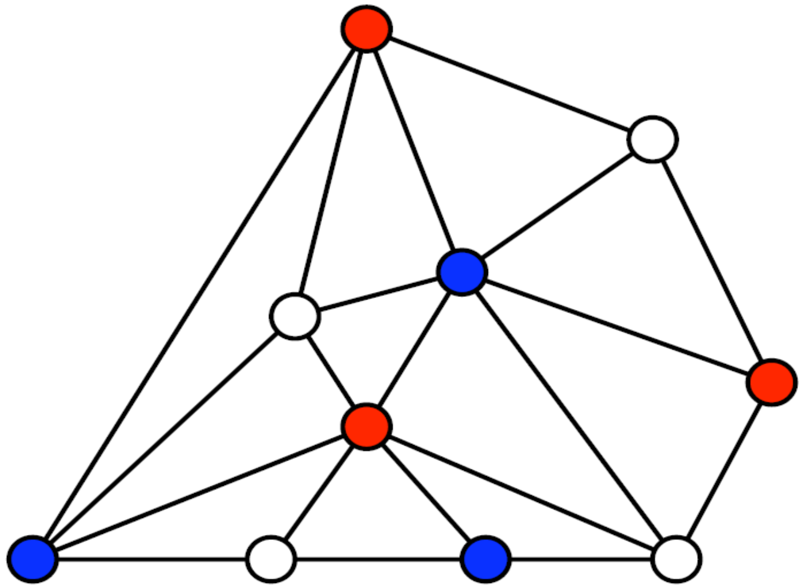
Applications of ZKPs

- *Identification scheme*. How should Alice prove to Bob that she is who she claimed to be? For example, how to design a control access system to the CSE dept.?
- A direct solution is to have a box on the door and give authorized people a secret PIN number. However, a drawback is that the box remains outside all the time and if someone could examine the box, they would perhaps be able to view its memory and extract the secrets keys of all people.

Applications of ZKPs

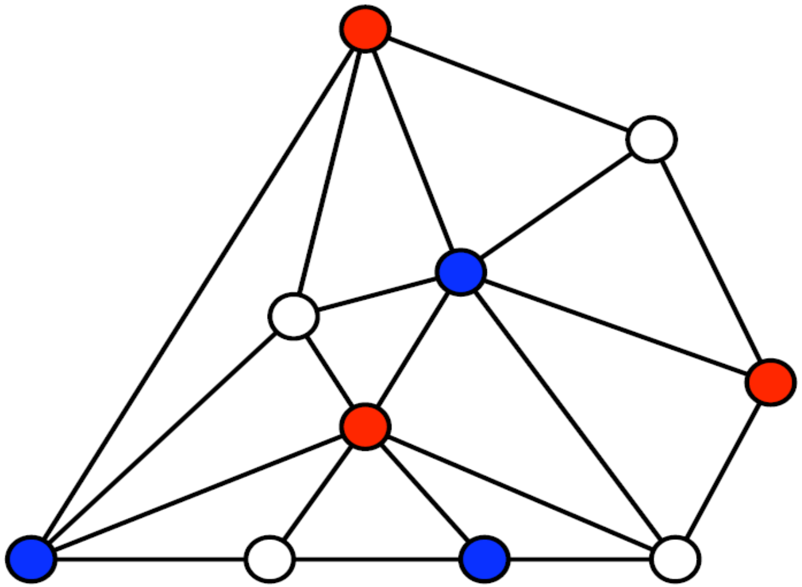
- *Identification scheme*. How should Alice prove to Bob that she is who she claimed to be? For example, how to design a control access system to the CSE dept.?
- A direct solution is to have a box on the door and give authorized people a secret PIN number. However, a drawback is that the box remains outside all the time and if someone could examine the box, they would perhaps be able to view its memory and extract the secrets keys of all people.
- **Ideas using ZKPs**:
  - Let the box contain an *instance* of a hard problem.
  - Give the authorized people the *solution* to the instance.
  - The authorized people will *prove* to the box that they know the solution in zero knowledge.

An example



- Alice knows how to 3-color a graph: no two adjacent vertices have the same color; this is an NPC problem.

An example



- Alice knows how to 3-color a graph: no two adjacent vertices have the same color; this is an NPC problem.
  - can impress your friends
  - useful for identification

An example

- How can Alice convince Bob that she can 3-color the graph without
  - letting him steal her work?
  - letting him impersonate her?

An example

- How can Alice convince Bob that she can 3-color the graph without
  - letting him steal her work?
  - letting him impersonate her?

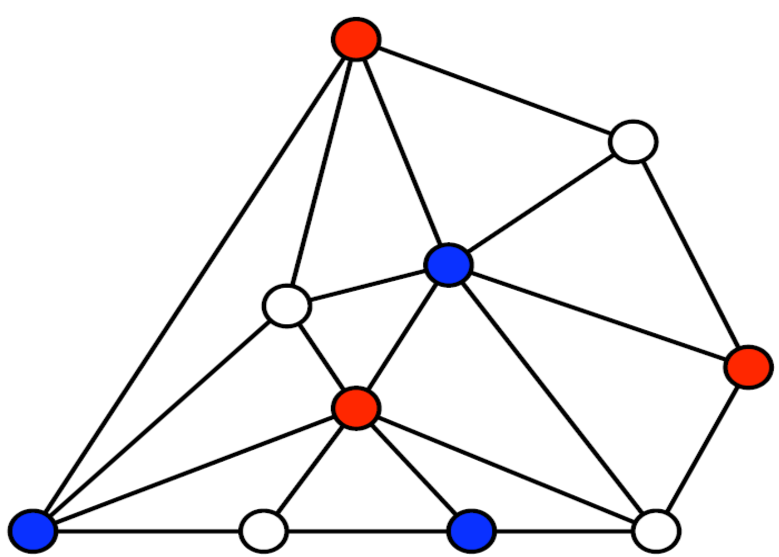
  - Bob is convinced that Alice can do this.
  - Bob has no idea how to do it himself.

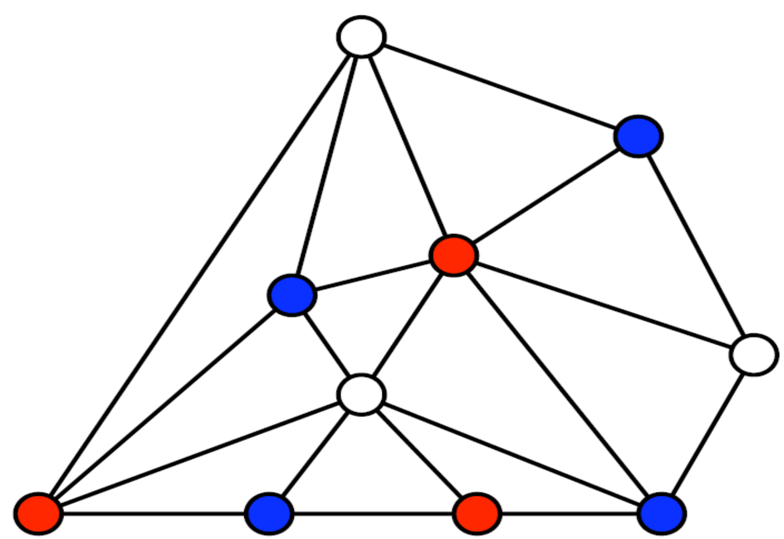
An example

- How can Alice convince Bob that she can 3-color the graph without
  - letting him steal her work?
  - letting him impersonate her?

  - Bob is convinced that Alice can do this.
  - Bob has no idea how to do it himself.

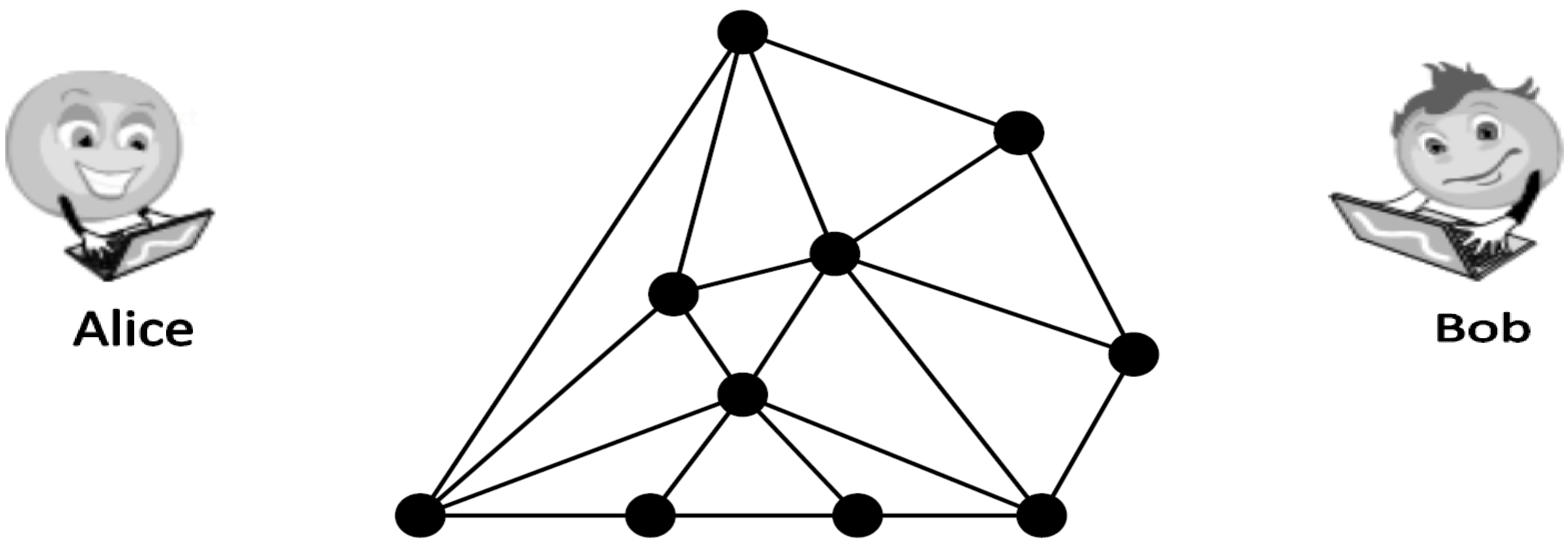
Alice may permute the vertex colors.





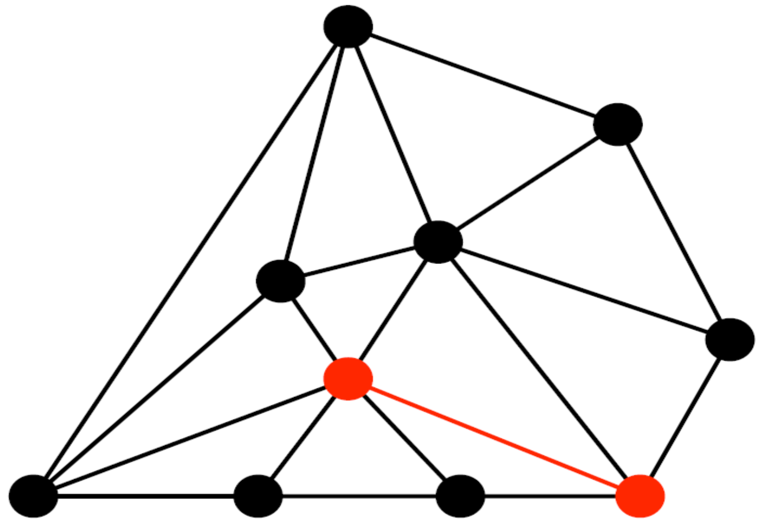
An example

- Alice then encrypts all vertex colors (one key per vertex), and sends the graph to Bob.



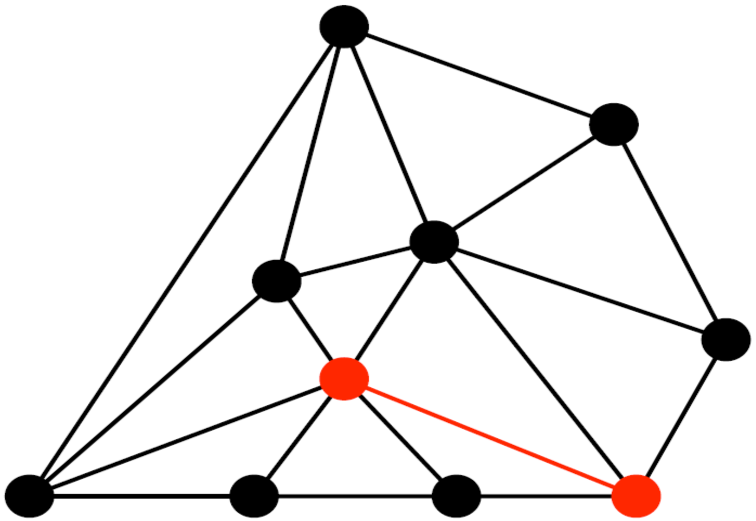
An example

- Bob picks an edge at random.

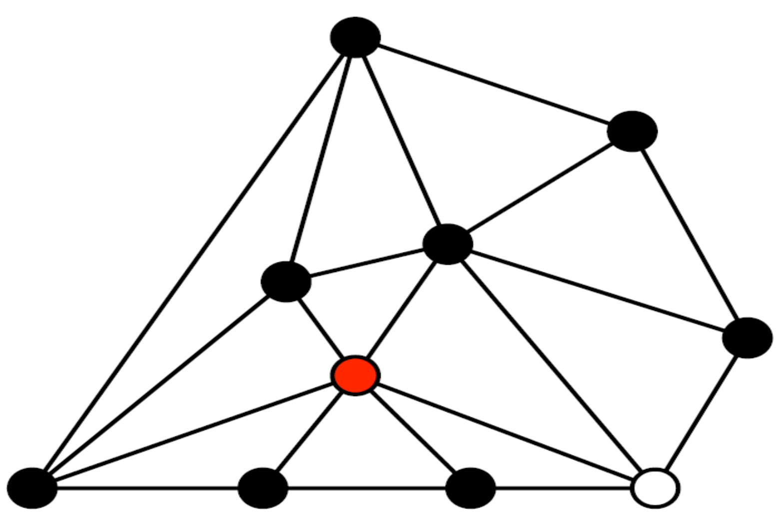


An example

- Bob picks an edge at random.



Alice reveals colors of those two keys.



An example

- Repeat as much as needed:
  - Alice permutes graph coloring
  - Alice enrypts all vertices with distinct keys
  - Alice sends permuted encrypted colors to Bob
  - Bob picks an edge
  - Alice sends keys for two vertices
  - Bob checks whether these two colors are distinct

An example

- Repeat as much as needed:
  - Alice permutes graph coloring
  - Alice enrypts all vertices with distinct keys
  - Alice sends permuted encrypted colors to Bob
  - Bob picks an edge
  - Alice sends keys for two vertices
  - Bob checks whether these two colors are distinct

If Alice is lying, with probability $\frac{1}{|E|}$ she will be caught.
If she is telling the truth, she will never be caught.

An example

- Repeat as much as needed:
  - Alice permutes graph coloring
  - Alice enrypts all vertices with distinct keys
  - Alice sends permuted encrypted colors to Bob
  - Bob picks an edge
  - Alice sends keys for two vertices
  - Bob checks whether these two colors are distinct

If Alice is lying, with probability $\frac{1}{|E|}$ she will be caught.
If she is telling the truth, she will never be caught.

After k repetitions, the probability she fools Bob is $(1 - \frac{1}{|E|})^k$.

An example

- What does Bob see?
  - randomly-generated keys
  - randomly-generated colors

An example

- What does Bob see?
  - randomly-generated keys
  - randomly-generated colors

  Because Bob could have generated those keys and colors by himself, he learns nothing from the graph coloring.

The Principle of Mathematical Induction

- The *well-ordering* principle permits us to assume that every set of nonnegative integers has a smallest element, allowing us to use the smallest counterexample.

The Principle of Mathematical Induction

- The *well-ordering* principle permits us to assume that every set of nonnegative integers has a smallest element, allowing us to use the smallest counterexample.

  This is actually **equivalent** to the *principle of mathematical induction*.

The Principle of Mathematical Induction

- The *well-ordering* principle permits us to assume that every set of nonnegative integers has a smallest element, allowing us to use the smallest counterexample.

  This is actually **equivalent** to the *principle of mathematical induction*.

  Principle. (*Weak Principle of Mathematical Induction*)

  (a) If the statement $P(b)$ is true

  (b) the statement $P(n-1) \to P(n)$ is true for all $n > b$,

  then $P(n)$ is true for all integers $n \geq b$

The Principle of Mathematical Induction

- The *well-ordering* principle permits us to assume that every set of nonnegative integers has a smallest element, allowing us to use the smallest counterexample.

  This is actually **equivalent** to the *principle of mathematical induction*.

  Principle. (*Weak Principle of Mathematical Induction*)
  (a) If the statement $P(b)$ is true
  (b) the statement $P(n-1) \rightarrow P(n)$ is true for all $n > b$,
  then $P(n)$ is true for all integers $n \geq b$

  (a) – *Basic Step* *Inductive Hypothesis*

  (b) – *Inductive Step* *Inductive Conclusion*

Another Form of Induction

- We may have another form of *direct proof* as follows.

Another Form of Induction

- We may have another form of *direct proof* as follows.
  - ◇ First suppose that we have proof of $P(0)$

Another Form of Induction

- We may have another form of *direct proof* as follows.
  - ◇ First suppose that we have proof of $P(0)$
  - ◇ Next suppose that we have a proof that, $\forall k > 0$,

$$P(0) \wedge P(1) \wedge P(2) \wedge \cdots \wedge P(k-1) \rightarrow P(k)$$

Another Form of Induction

- We may have another form of *direct proof* as follows.
  - ◇ First suppose that we have proof of $P(0)$
  - ◇ Next suppose that we have a proof that, $\forall k > 0$,

$$P(0) \wedge P(1) \wedge P(2) \wedge \cdots \wedge P(k-1) \rightarrow P(k)$$

  - ◇ Then, $P(0)$ implies $P(1)$

$P(0) \wedge P(1)$ implies $P(2)$

$P(0) \wedge P(1) \wedge P(2)$ implies $P(3)$ ...

Another Form of Induction

- We may have another form of *direct proof* as follows.
  - ◇ First suppose that we have proof of $P(0)$
  - ◇ Next suppose that we have a proof that, $\forall k > 0$,

$$P(0) \wedge P(1) \wedge P(2) \wedge \cdots \wedge P(k-1) \rightarrow P(k)$$

  - ◇ Then, $P(0)$ implies $P(1)$

$P(0) \wedge P(1)$ implies $P(2)$

$P(0) \wedge P(1) \wedge P(2)$ implies $P(3)$ ...

  - ◇ Iterating gives us a proof of $P(n)$ for all n

Strong Induction

- **Principle** (*Strong Principle of Mathematical Induction*)

  (a) If the statement $P(b)$ is true

  (b) for all $n > b$, the statement
  $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \to P(n)$ is true.

  then $P(n)$ is true for all integers $n \geq b$.

Example

- Prove that every positive integer is a power of a prime or the product of powers of primes.

Example

- Prove that every positive integer is a power of a prime or the product of powers of primes.
  - Base Step: 1 is a power of a prime number, $1 = 2^0$

Example

- Prove that every positive integer is a power of a prime or the product of powers of primes.

  ◇ Base Step: 1 is a power of a prime number, $1 = 2^0$

  ◇ Inductive Hypothesis: Suppose that every number less than n is a power of a prime or a product of powers of primes.

Example

- Prove that every positive integer is a power of a prime or the product of powers of primes.

  ◇ Base Step: 1 is a power of a prime number, $1 = 2^0$

  ◇ Inductive Hypothesis: Suppose that every number less than n is a power of a prime or a product of powers of primes.

  ◇ Then, if n is not a prime power, it is a product of two smaller numbers, each of which is, by the inductive hypothesis, a power of a prime or a product of powers of primes.

Example

- Prove that every positive integer is a power of a prime or the product of powers of primes.

  ◇ Base Step: 1 is a power of a prime number, $1 = 2^0$

  ◇ Inductive Hypothesis: Suppose that every number less than n is a power of a prime or a product of powers of primes.

  ◇ Then, if n is not a prime power, it is a product of two smaller numbers, each of which is, by the inductive hypothesis, a power of a prime or a product of powers of primes.

  ◇ Thus, by the strong principle of mathematical induction, every positive integer is a power of a prime or a product of powers of primes.

Mathematical Induction

- In practice, we do not usually explicitly distinguish between the weak and strong forms.

Mathematical Induction

- In practice, we do not usually explicitly distinguish between the weak and strong forms.

- In reality, they are equivalent to each other in that the weak form is a special case of the strong form, and the strong form can be derived from the weak form.

Summary

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

Summary

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step

Summary

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step
  2. We then, $\forall n > b$, show either

$$(*) \qquad P(n-1) \to P(n)$$

or

$$(**) \qquad P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \to P(n)$$

Summary

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step
  2. We then, $\forall n > b$, show either

  $$(*) \qquad P(n-1) \to P(n)$$

  or

  $$(**) \qquad P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \to P(n)$$

  We need to make the inductive hypothesis of either $P(n-1)$ or $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1)$.We then use $(*)$ or $(**)$ to derive $P(n)$.

Summary

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step
  2. We then, $\forall n > b$, show either

     $$(*) \qquad P(n-1) \rightarrow P(n)$$

     or

     $$(**) \qquad P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \rightarrow P(n)$$

     We need to make the inductive hypothesis of either $P(n-1)$ or $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1)$.We then use $(*)$ or $(**)$ to derive $P(n)$.
  3. We conclude on the basis of the principle of mathematical induction that $P(n)$ is true for all $n \geq b$.

Recursion

- Recursive computer programs or algorithms often lead to inductive analysis.

Recursion

- Recursive computer programs or algorithms often lead to inductive analysis.

- A classical example of *recursion* is the **Towers of Hanoi** Problem.

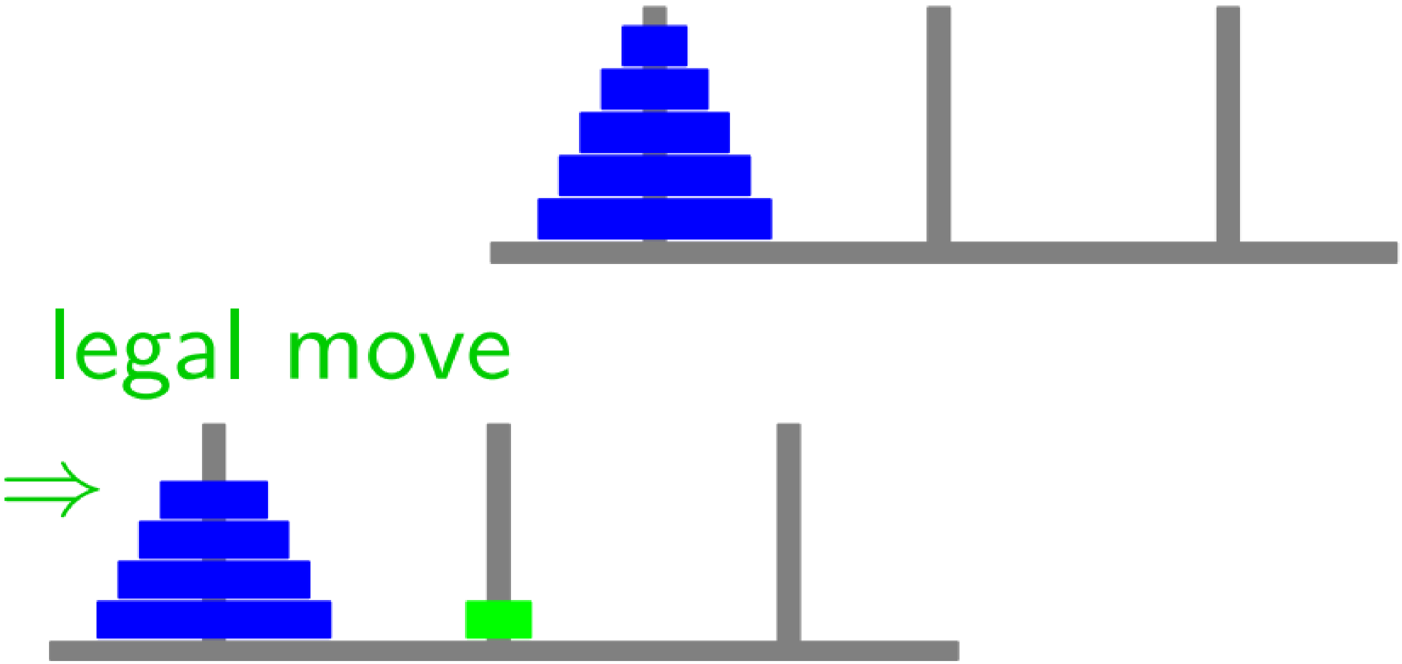
Towers of Hanoi

Towers of Hanoi

- 3 pegs; n disks of different sizes
- A *legal move* takes a disk from one peg and moves it onto another peg so that it is not on top of a smaller disk
- **Problem**: Find a (efficient) way to move all of the disks from one peg to another

Towers of Hanoi

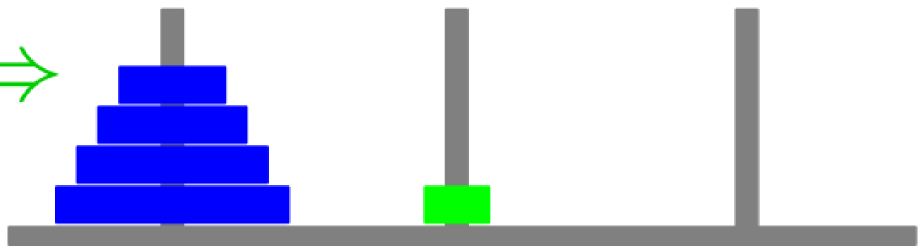
Towers of Hanoi



Towers of Hanoi

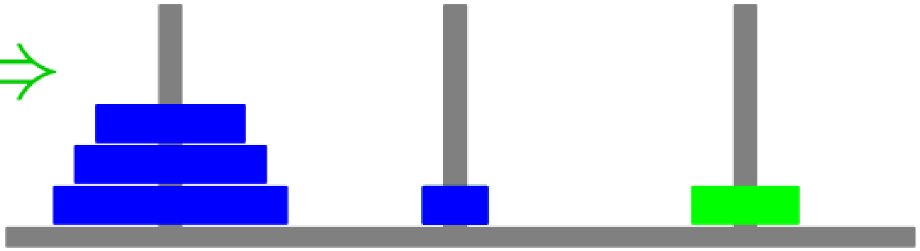
legal move

$\Rightarrow$



legal move

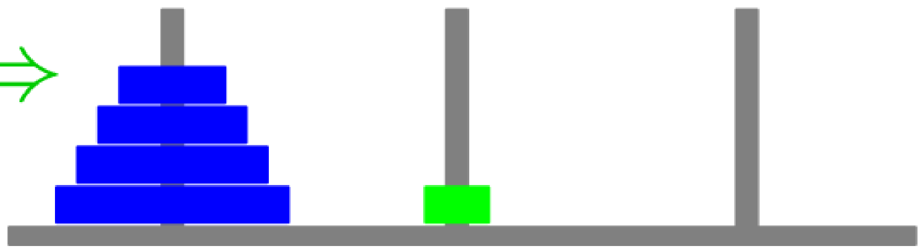
$\Rightarrow$



Towers of Hanoi

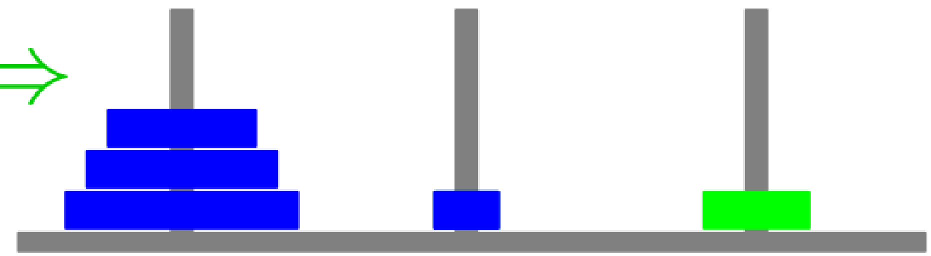
legal move

$\Rightarrow$



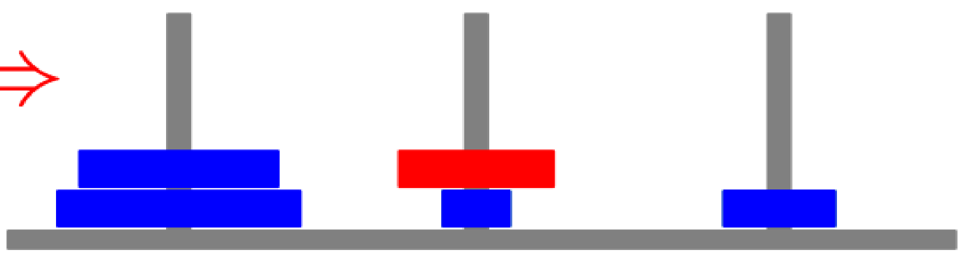
legal move

$\Rightarrow$



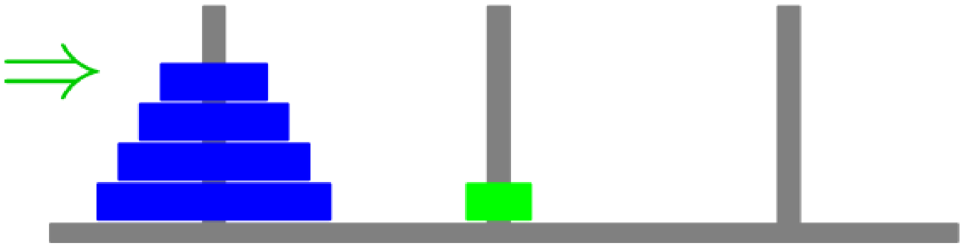
not legal

$\Rightarrow$

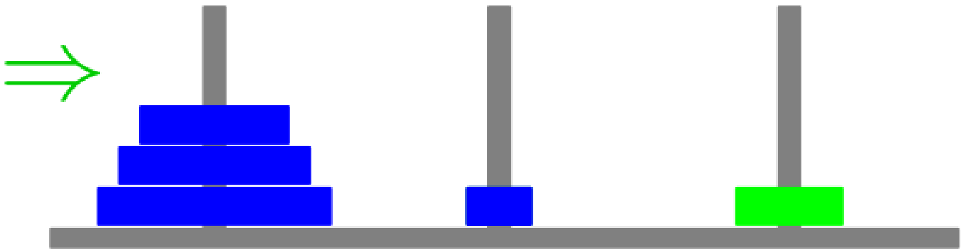


Towers of Hanoi

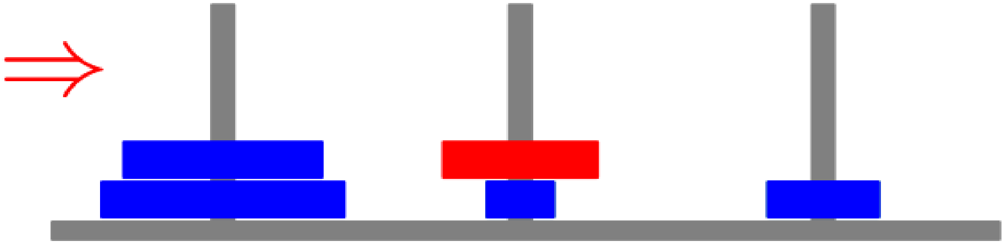
legal move



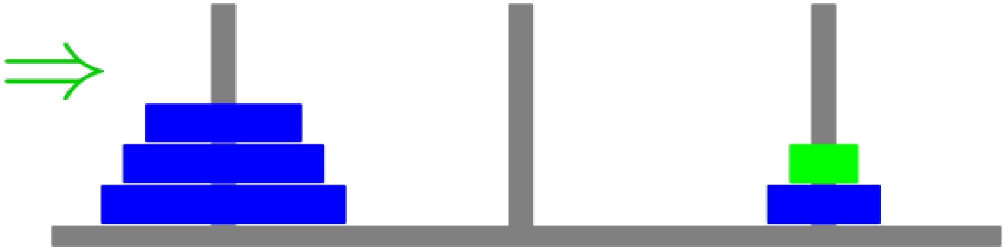
legal move



not legal



legal move



Towers of Hanoi

- **Problem:** Start with n disks on leftmost peg

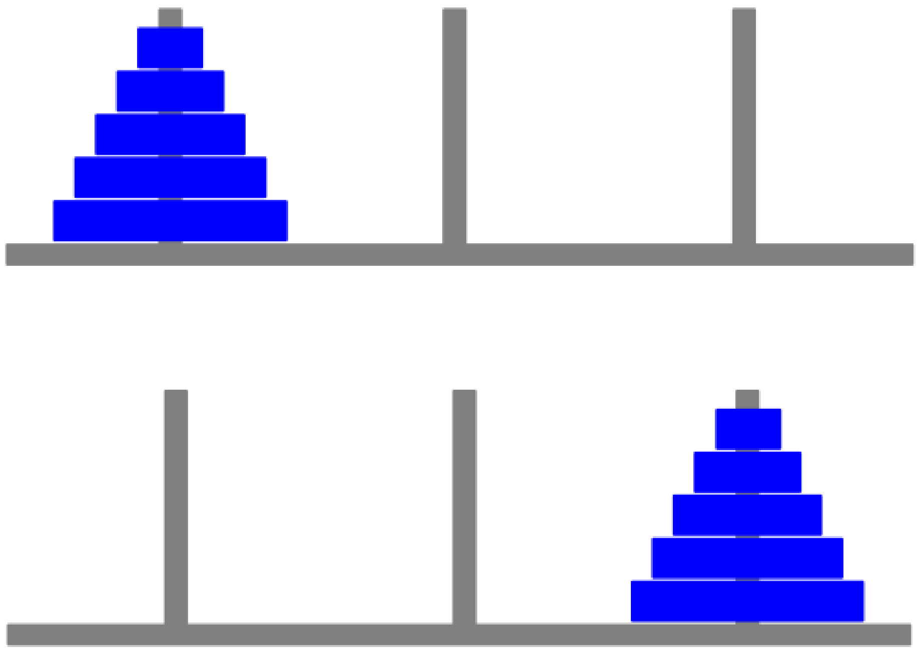
Towers of Hanoi

- **Problem:** Start with n disks on leftmost peg

  using only legal moves

Towers of Hanoi

- **Problem:** Start with n disks on leftmost peg
  using only legal moves
  move all disks to rightmost peg.

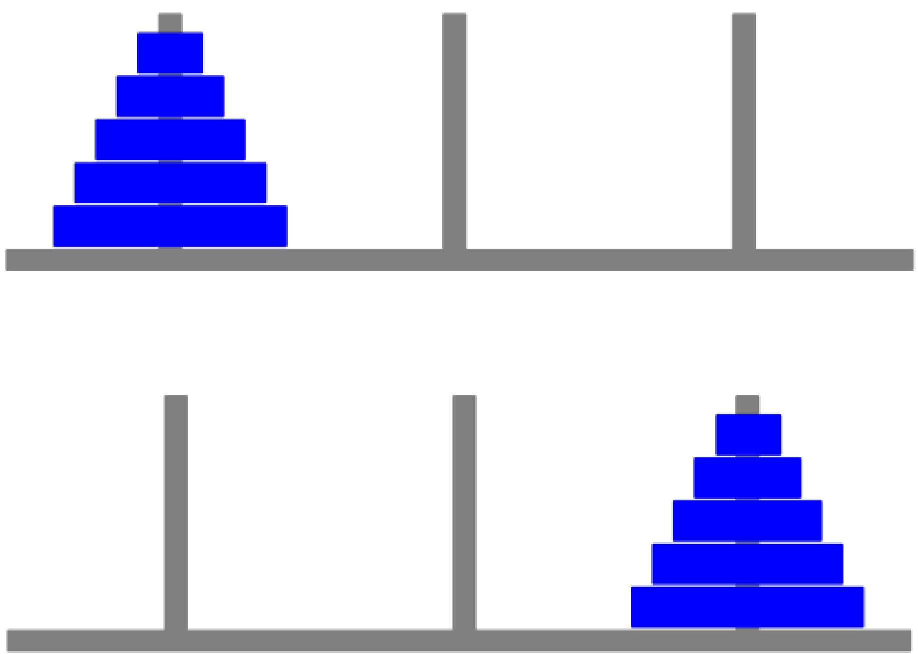


Towers of Hanoi

- **Problem:** Start with n disks on leftmost peg

  using only legal moves

  move all disks to rightmost peg.



Given $i, j \in \{1,2,3\}$, let
$\overline{\{i,j\}} = \{1,2,3\} - \{i\} - \{j\}$,
i.e., $\overline{\{1,2\}} = \{3\}$, $\overline{\{1,3\}} = \{2\}$,
$\overline{\{2,3\}} = \{1\}$.

Towers of Hanoi

- General solution

Towers of Hanoi

- General solution

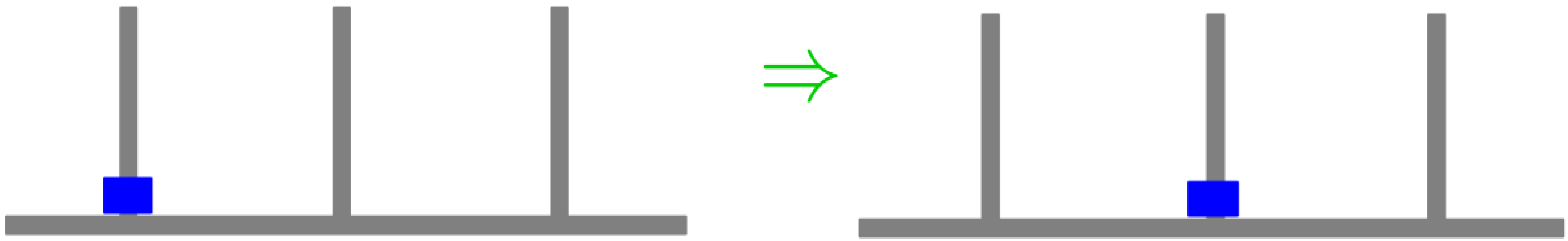
  Recursion Base:

  If $n = 1$, moving one disk from i to j is easy. Just move it.

- General solution

  Recursion Base:

  If $n = 1$, moving one disk from i to j is easy. Just move it.



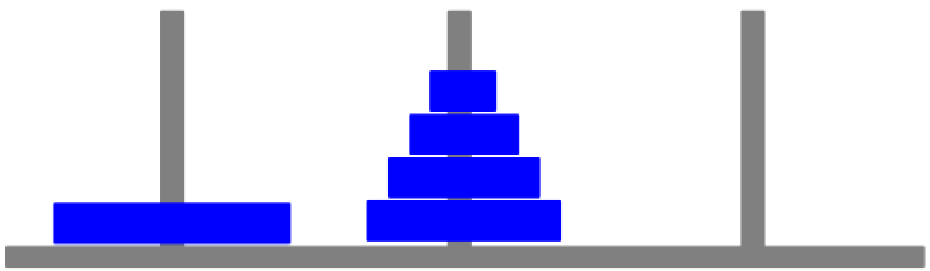
Towers of Hanoi

To move $n > 1$ disks from i to j

Towers of Hanoi

To move $n > 1$ disks from i to j

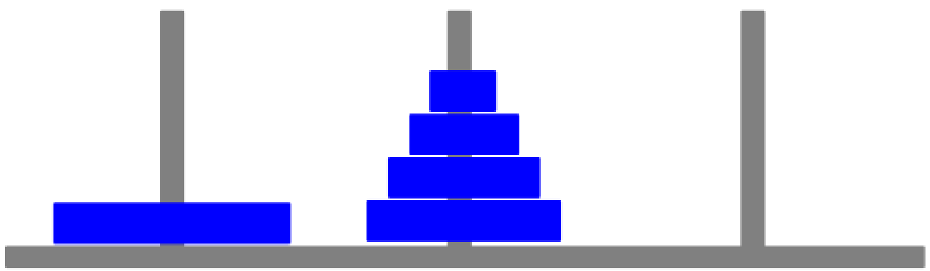
1)



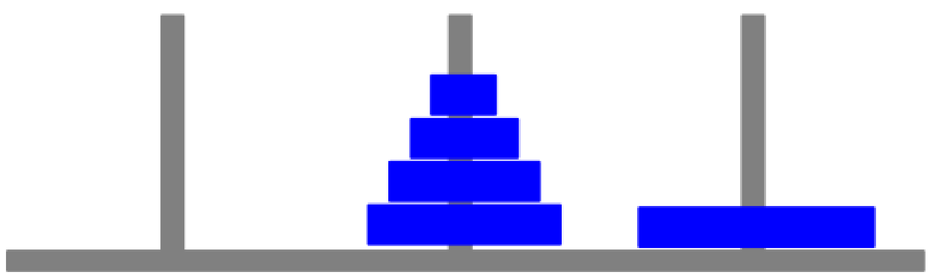
move top $n-1$ disks from i to $\overline{\{i,j\}}$

Towers of Hanoi

To move $n > 1$ disks from i to j

1) 

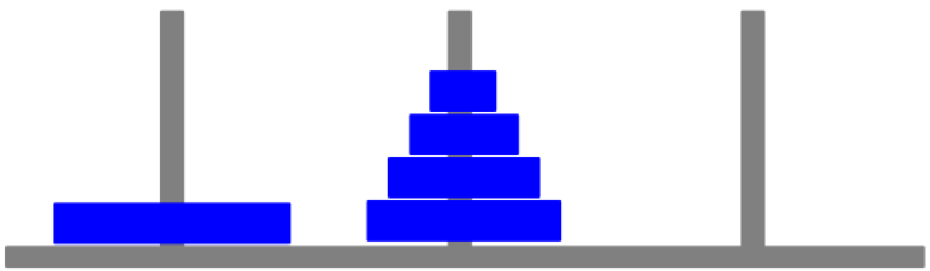
move top $n - 1$ disks from i to $\overline{\{i, j\}}$

2) 

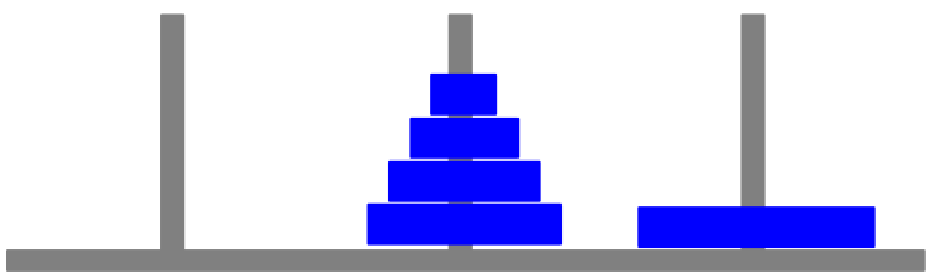
move largest disk from i to j

Towers of Hanoi

To move $n > 1$ disks from i to j

1) 

move top $n-1$ disks from i to $\overline{\{i,j\}}$

2) 

move largest disk from i to j

3) 

move top $n-1$ disks from $\overline{\{i,j\}}$ to j

Towers of Hanoi

```java
public class Hanoi
{

    public void move(int n, char a, char b, char c)
    {
        if (n == 1)
            System.out.println("plate " + n + " from " + a + " to " + c);
        else
        {
            move(n-1,a,c,b);
            System.out.println("plate " + n + " from " + a + " to " + c);
            move(n-1,b,a,c);
        }

    }

```

Towers of Hanoi

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

Towers of Hanoi

- To prove Correctness of solution, we are implicitly using induction

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

Towers of Hanoi

- To prove Correctness of solution, we are implicitly using induction

- $p(n)$ is statement that algorithm is correct for n

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

Towers of Hanoi

- To prove Correctness of solution, we are implicitly using induction
- $p(n)$ is statement that algorithm is correct for n

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

- $p(1)$ is statement that algorithm works for $n = 1$ disks, which is obviously true

Towers of Hanoi

- To prove Correctness of solution, we are implicitly using induction

- $p(n)$ is statement that algorithm is correct for n

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

- $p(1)$ is statement that algorithm works for $n = 1$ disks, which is obviously true

- $p(n-1) \rightarrow p(n)$ is *recursion* statement that

  if our algorithm works for $n-1$ disks, then we can build a correct solution for n disks

Towers of Hanoi

- Running time

  $M(n)$ is number of disk moves needed for n disks

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

Towers of Hanoi

- Running time

  $M(n)$ is number of disk moves needed for n disks

  $M(1) = 1$

  if $n > 1$, then $M(n) = 2M(n-1) + 1$

To move n disks from i to j

i) move top $n-1$ disks from i to $\overline{\{i,j\}}$

ii) move largest disk from i to j

iii) move top $n-1$ disks from $\overline{\{i,j\}}$ to j

Towers of Hanoi

- We saw that $M(1) = 1$ and that
- $M(n) = 2M(n-1) + 1$ for $n > 1$

Towers of Hanoi

- We saw that $M(1) = 1$ and that
- $M(n) = 2M(n-1) + 1$ for $n > 1$

- Iterating the recurrence gives

  $M(1) = 1$, $M(2) = 3$, $M(3) = 7$,

  $M(4) = 15$, $M(5) = 31$, ...

Towers of Hanoi

- We saw that $M(1) = 1$ and that
- $M(n) = 2M(n-1) + 1$ for $n > 1$

- Iterating the recurrence gives

  $$M(1) = 1,\ M(2) = 3,\ M(3) = 7,$$
  $$M(4) = 15,\ M(5) = 31,\ \ldots$$

- We *guess* that $M(n) = 2^n - 1$

Towers of Hanoi

- We saw that $M(1) = 1$ and that
- $M(n) = 2M(n-1) + 1$ for $n > 1$

- Iterating the recurrence gives

  $M(1) = 1$, $M(2) = 3$, $M(3) = 7$,

  $M(4) = 15$, $M(5) = 31$, $\ldots$

- We *guess* that $M(n) = 2^n - 1$

  We'll prove this by induction

Towers of Hanoi

- We saw that $M(1) = 1$ and that
- $M(n) = 2M(n-1) + 1$ for $n > 1$

- Iterating the recurrence gives

  $M(1) = 1$, $M(2) = 3$, $M(3) = 7$,

  $M(4) = 15$, $M(5) = 31$, $\ldots$

- We *guess* that $M(n) = 2^n - 1$

  We'll prove this by induction

  Later, we'll also see how to solve without guessing

Towers of Hanoi

- Formally, given

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1) + 1 & \text{otherwise} \end{cases}$$

We show that $M(n) = 2^n - 1$.

Towers of Hanoi

- Formally, given

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1)+1 & \text{otherwise} \end{cases}$$

We show that $M(n) = 2^n - 1$.

Proof. (by induction)

The base case $n = 1$ is true, since $2^1 - 1 = 1$.

For the inductive step, assume that $M(n-1) = 2^{n-1} - 1$ for $n > 1$.

- Formally, given

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1)+1 & \text{otherwise} \end{cases}$$

We show that $M(n) = 2^n - 1$.

Proof. (by induction)

The base case $n = 1$ is true, since $2^1 - 1 = 1$.

For the inductive step, assume that $M(n-1) = 2^{n-1} - 1$ for $n > 1$.

Then $M(n) = 2M(n-1) + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$

Towers of Hanoi

- Note that we used induction twice.

Towers of Hanoi

- Note that we used induction twice.

- The first time was to derive correctness of algorithm and the recurrence

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1) + 1 & \text{otherwise} \end{cases}$$

Towers of Hanoi

- Note that we used induction twice.
- The first time was to derive correctness of algorithm and the recurrence

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1)+1 & \text{otherwise} \end{cases}$$

- The second time was to derive the closed form solution $M(n) = 2^n - 1$ of the recurrence.

Recurrences

- A *recurrence equation* or *recurrence* for a function defined on the set of integers $\geq b$ is one that tells us how to compute the nth value from some or all the first $n-1$ values.

Recurrences

- A *recurrence equation* or *recurrence* for a function defined on the set of integers $\geq b$ is one that tells us how to compute the nth value from some or all the first $n-1$ values.

  To completely specify a function on the basis of a recurrence, we have to give the *initial condition(s)* (a.k.a. the *base case(s)*) for the recurrence.

Recurrences

- A *recurrence equation* or *recurrence* for a function defined on the set of integers $\geq b$ is one that tells us how to compute the nth value from some or all the first $n-1$ values.

  To completely specify a function on the basis of a recurrence, we have to give the *initial condition(s)* (a.k.a. the *base case(s)*) for the recurrence.

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2M(n-1)+1 & \text{otherwise} \end{cases}$$

Towers of Hanoi

Fibonacci Sequence

$$F(n) = \begin{cases} 1 & \text{if } n = \\ F(n-1)+F(n-2) & \text{other} \end{cases}$$

- **Example 2**: Let $S(n)$ be the number of subsets of a set of size n. What is the formula for $S(n)$?

  The empty set, of size $n = 0$ has only one subset (itself), so $S(0) = 1$.

  It is not difficult to see that

  $S(1) = 2$, $S(2) = 4$, $S(3) = 8$

- **Example 2**: Let $S(n)$ be the number of subsets of a set of size n. What is the formula for $S(n)$?

  The empty set, of size $n = 0$ has only one subset (itself), so $S(0) = 1$.

  It is not difficult to see that

  $S(1) = 2$, $S(2) = 4$, $S(3) = 8$

  We "guess" that $S(n) = 2^n$. But, in order to prove formula, we'll need to think recursively.

Recurrences

- Consider the eight subsets of $\{1, 2, 3\}$:

  $\emptyset, \{1\}, \{2\}, \{1, 2\}, \{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$

- Consider the eight subsets of $\{1,2,3\}$:

$$\emptyset, \{1\}, \{2\}, \{1,2\}, \{3\}, \{1,3\}, \{2,3\}, \{1,2,3\}$$

$$\begin{array}{cccc} \emptyset & \{1\} & \{2\} & \{1,2\} \\ \{3\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array}$$

- Consider the eight subsets of $\{1, 2, 3\}$:

  $\emptyset, \{1\}, \{2\}, \{1, 2\}, \{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$

$$\begin{array}{cccc} \emptyset & \{1\} & \{2\} & \{1,2\} \\ \{3\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array}$$

  First four subsets are exactly the subsets of $\{1, 2\}$, while second four are the subsets of $\{1, 2\}$ with 3 added into each.

Recurrences

- Consider the eight subsets of $\{1, 2, 3\}$:

  $\emptyset, \{1\}, \{2\}, \{1, 2\}, \{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$

$$\begin{array}{cccc} \emptyset & \{1\} & \{2\} & \{1,2\} \\ \{3\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array}$$

  First four subsets are exactly the subsets of $\{1, 2\}$, while second four are the subsets of $\{1, 2\}$ with 3 added into each.

  So, we get a subset of $\{1, 2, 3\}$ either by taking a subset of $\{1, 2\}$ or by adjoining 3 to a subset of $\{1, 2\}$.

Recurrences

- Consider the eight subsets of $\{1,2,3\}$:

  $\emptyset, \{1\}, \{2\}, \{1,2\}, \{3\}, \{1,3\}, \{2,3\}, \{1,2,3\}$

  $$\begin{array}{cccc} \emptyset & \{1\} & \{2\} & \{1,2\} \\ \{3\} & \{1,3\} & \{2,3\} & \{1,2,3\} \end{array}$$

  First four subsets are exactly the subsets of $\{1,2\}$, while second four are the subsets of $\{1,2\}$ with 3 added into each.

  So, we get a subset of $\{1,2,3\}$ either by taking a subset of $\{1,2\}$ or by adjoining 3 to a subset of $\{1,2\}$.

  This suggests that the recurrence for the number of subsets of an n-element set $\{1,2,\ldots,n\}$ is

  $$S(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2S(n-1) & \text{if } n \geq 1 \end{cases}$$

Recurrences

- **Proof.** of correctness of this recurrence

- **Proof.** of correctness of this recurrence

  The subsets of $\{1, 2, \ldots, n\}$ can be partitioned according to whether or not they contain the element n.

Recurrences

- **Proof.** of correctness of this recurrence

  The subsets of $\{1, 2, \ldots, n\}$ can be partitioned according to whether or not they contain the element n.

  Each subset S containing n can be constructed in a unique fashion by adding n to the subset $S - \{n\}$ not containing n.

  Each subset S not containing n can be constructed by removing n from the unique set $S \cup \{n\}$ containing n.

- **Proof.** of correctness of this recurrence

  The subsets of $\{1, 2, \ldots, n\}$ can be partitioned according to whether or not they contain the element n.

  Each subset S containing n can be constructed in a unique fashion by adding n to the subset $S - \{n\}$ not containing n.

  Each subset S not containing n can be constructed by removing n from the unique set $S \cup \{n\}$ containing n.

  So, the number of subsets containing n is exactly the same as the number of subsets not containing n.

Recurrences

- **Proof.** of correctness of this recurrence

  The subsets of $\{1, 2, \ldots, n\}$ can be partitioned according to whether or not they contain the element n.

  Each subset S containing n can be constructed in a unique fashion by adding n to the subset $S - \{n\}$ not containing n.

  Each subset S not containing n can be constructed by removing n from the unique set $S \cup \{n\}$ containing n.

  So, the number of subsets containing n is exactly the same as the number of subsets not containing n.

  Thus, if $n > 1$, then $S(n) = 2S(n-1)$.

Recurrences

- **Proof.** of correctness of this recurrence

  The subsets of $\{1, 2, \ldots, n\}$ can be partitioned according to whether or not they contain the element n.

  Each subset S containing n can be constructed in a unique fashion by adding n to the subset $S - \{n\}$ not containing n.

  Each subset S not containing n can be constructed by removing n from the unique set $S \cup \{n\}$ containing n.

  So, the number of subsets containing n is exactly the same as the number of subsets not containing n.

  Thus, if $n > 1$, then $S(n) = 2S(n-1)$.

  Proof by induction is easy.

Iterating a Recurrence

- Let $T(n) = rT(n-1) + a$,

  where r and a are constants.

Iterating a Recurrence

- Let $T(n) = rT(n-1) + a$,

  where r and a are constants.

  Find a recurrence that expresses

  $T(n)$ in terms of $T(n-2)$
  $T(n)$ in terms of $T(n-3)$
  $T(n)$ in terms of $T(n-4)$
  $\vdots$

Iterating a Recurrence

- Let $T(n) = rT(n-1) + a$,

  where r and a are constants.

  Find a recurrence that expresses

  $T(n)$ in terms of $T(n-2)$
  $T(n)$ in terms of $T(n-3)$
  $T(n)$ in terms of $T(n-4)$
  $\vdots$

  Can we generalize this to find a closed-form solution?

Iterating a Recurrence

- Note that $T(n) = rT(n-1) + a$ implies that

  $$\forall i < n,\ T(n-i) = rT((n-i)-1)) + a$$

Iterating a Recurrence

- Note that $T(n) = rT(n-1) + a$ implies that

  $\forall i < n,\ T(n-i) = rT((n-i)-1)) + a$

  Then, we have

$$
\begin{aligned}
T(n) &= rT(n-1) + a \\
&= r(rT(n-2) + a) + a \\
&= r^2T(n-2) + ra + a \\
&= r^2(rT(n-3) + a) + ra + a \\
&= r^3T(n-3) + r^2a + ra + a \\
&= r^3(rT(n-4) + a) + r^2a + ra + a \\
&= r^4T(n-4) + r^3a + r^2a + ra + a.
\end{aligned}
$$

Iterating a Recurrence

- Note that $T(n) = rT(n-1) + a$ implies that

  $\forall i < n,\ T(n-i) = rT((n-i)-1)) + a$

  Then, we have

$$
\begin{aligned}
T(n) &= rT(n-1) + a \\
&= r(rT(n-2) + a) + a \\
&= r^2 T(n-2) + ra + a \\
&= r^2(rT(n-3) + a) + ra + a \\
&= r^3 T(n-3) + r^2 a + ra + a \\
&= r^3(rT(n-4) + a) + r^2 a + ra + a \\
&= r^4 T(n-4) + r^3 a + r^2 a + ra + a.
\end{aligned}
$$

Guess $T(n) = r^n T(0) + a \sum_{i=0}^{n-1} r^i$

Iterating a Recurrence

- The method we used to guess the solution is called *iterating the recurrence*, because we repeatedly (iteratively) use the recurrence.

Iterating a Recurrence

- The method we used to guess the solution is called *iterating the recurrence*, because we repeatedly (iteratively) use the recurrence.

  Another approach is to iterate from the "bottom-up" instead of "top-down".

Iterating a Recurrence

- The method we used to guess the solution is called *iterating the recurrence*, because we repeatedly (iteratively) use the recurrence.

  Another approach is to iterate from the "bottom-up" instead of "top-down".

  $$\begin{aligned} &T(0) = b \\ &T(1) = rT(0) + a = rb + a \\ &T(2) = rT(1) + a = r(rb + a) + a = r^2b + ra + a \\ &T(3) = rT(2) + a = r^3b + r^2a + ra + a \end{aligned}$$

Iterating a Recurrence

- The method we used to guess the solution is called *iterating the recurrence*, because we repeatedly (iteratively) use the recurrence.

  Another approach is to iterate from the "bottom-up" instead of "top-down".

  $$\begin{aligned} &T(0) = b \\ &T(1) = rT(0) + a = rb + a \\ &T(2) = rT(1) + a = r(rb + a) + a = r^2 b + ra + a \\ &T(3) = rT(2) + a = r^3 b + r^2 a + ra + a \end{aligned}$$

  This would lead to the same guess

  $$T(n) = r^n b + a \sum_{i=0}^{n-1} r^i.$$

Formula of Recurrences

- **Theorem** If $T(n) = rT(n-1) + a$, $T(0) = b$, and $r \neq 1$, then

$$T(n) = r^n b + a\frac{1 - r^n}{1 - r}$$

for all nonnegative integers n.

- **Theorem** If $T(n) = rT(n-1) + a$, $T(0) = b$, and $r \neq 1$, then

$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$

for all nonnegative integers n.

Proof by induction

The base case:

$$T(0) = r^0 b + a\frac{1-r^0}{1-r} = b.$$

So the formula is true when $n = 0$.

Now assume that $n > 0$ and

$$T(n-1) = r^{n-1} b + a\frac{1-r^{n-1}}{1-r}.$$

- **Proof by induction**

$$
\begin{aligned}
T(n) &= rT(n-1) + a \\
&= r\left(r^{n-1}b + a\frac{1-r^{n-1}}{1-r}\right) + a \\
&= r^n b + \frac{ar - ar^n}{1-r} + a \\
&= r^n b + \frac{ar - ar^n + a - ar}{1-r} \\
&= r^n b + a\frac{1-r^n}{1-r}.
\end{aligned}
$$

Formula of Recurrences

- **Theorem** If $T(n) = rT(n-1) + a$, $T(0) = b$, and $r \neq 1$, then

$$T(n) = r^n b + a\frac{1 - r^n}{1 - r}$$

for all nonnegative integers n.

Formula of Recurrences

- **Theorem** If $T(n) = rT(n-1) + a$, $T(0) = b$, and $r \neq 1$, then

$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$

for all nonnegative integers n.

Example:
$T(n) = 3T(n-1) + 2$ with $T(0) = 5$

Formula of Recurrences

- **Theorem** If $T(n) = rT(n-1) + a$, $T(0) = b$, and $r \neq 1$, then

$$T(n) = r^n b + a\frac{1-r^n}{1-r}$$

for all nonnegative integers n.

Example:
$T(n) = 3T(n-1) + 2$ with $T(0) = 5$

Plugging $r = 3, a = 2, b = 5$ in the formula, gives

$$T(n) = 3^n \cdot 5 + 2\frac{1-3^n}{1-3} = 3^n \cdot 6 - 1$$

First-Order Linear Recurrences

- A recurrence of the form $T(n) = f(n)T(n-1) + g(n)$ is called a *first-order linear recurrence*.

First-Order Linear Recurrences

- A recurrence of the form $T(n) = f(n)T(n-1) + g(n)$ is called a *first-order linear recurrence*.
  - ◇ First Order because it only depends upon going back one step, i.e., $T(n-1)$

First-Order Linear Recurrences

- A recurrence of the form $T(n) = f(n)T(n-1) + g(n)$ is called a *first-order linear recurrence*.

    ◇ First Order because it only depends upon going back one step, i.e., $T(n-1)$

    If it depends upon $T(n-2)$, it would be a second-order recurrence, e.g., $T(n) = T(n-1) + 2T(n-2)$.

First-Order Linear Recurrences

- A recurrence of the form $T(n) = f(n)T(n-1) + g(n)$ is called a *first-order linear recurrence*.

  ◇ First Order because it only depends upon going back one step, i.e., $T(n-1)$

  If it depends upon $T(n-2)$, it would be a second-order recurrence, e.g., $T(n) = T(n-1) + 2T(n-2)$.

  ◇ Linear because $T(n-1)$ only appears to the first power.

First-Order Linear Recurrences

- A recurrence of the form $T(n) = f(n)T(n-1) + g(n)$ is called a *first-order linear recurrence*.

  ◇ First Order because it only depends upon going back one step, i.e., $T(n-1)$

  If it depends upon $T(n-2)$, it would be a second-order recurrence, e.g., $T(n) = T(n-1) + 2T(n-2)$.

  ◇ Linear because $T(n-1)$ only appears to the first power.

  Something like $T(n) = (T(n-1))^2 + 3$ would be a non-linear first-order recurrence relation.

First-Order Linear Recurrences

- $T(n) = f(n)T(n-1) + g(n)$

First-Order Linear Recurrences

- $T(n) = f(n)T(n-1) + g(n)$

  When $f(n)$ is a constant, say r, the general solution is almost as easy as we derived before. Iterating the recurrence gives

$$
\begin{aligned}
T(n) &= rT(n-1) + g(n) \\
&= r(rT(n-2) + g(n-1)) + g(n) \\
&= r^2 T(n-2) + rg(n-1) + g(n) \\
&= r^3 T(n-3) + r^2 g(n-2) + rg(n-1) + g(n) \\
&\vdots \\
&= r^n T(0) + \sum_{i=0}^{n-1} r^i g(n-i)
\end{aligned}
$$

- **Theorem** For any positive constants a and r, and any function g defined on nonnegative integers, the solution to the first-order linear recurrence

$$T(n) = \begin{cases} rT(n-1) + g(n) & \text{if } n > 0 \\ a & \text{if } n = 0 \end{cases}$$

is

$$T(n) = r^n a + \sum_{i=1}^{n} r^{n-i} g(i).$$

- **Theorem** For any positive constants a and r, and any function g defined on nonnegative integers, the solution to the first-order linear recurrence

$$T(n) = \begin{cases} rT(n-1) + g(n) & \text{if } n > 0 \\ a & \text{if } n = 0 \end{cases}$$

is

$$T(n) = r^n a + \sum_{i=1}^{n} r^{n-i} g(i).$$

Proof by induction

Examples

- Solve $T(n) = 4T(n-1) + 2^n$ with $T(0) = 6$

Examples

- Solve $T(n) = 4T(n-1) + 2^n$ with $T(0) = 6$

$$
\begin{aligned}
T(n) &= 6 \cdot 4^n + \sum_{i=1}^{n} 4^{n-i} \cdot 2^i \\
&= 6 \cdot 4^n + 4^n \sum_{i=1}^{n} 4^{-i} \cdot 2^i \\
&= 6 \cdot 4^n + 4^n \sum_{i=1}^{n} (\frac{1}{2})^i \\
&= 6 \cdot 4^n + (1 - \frac{1}{2^n}) \cdot 4^n \\
&= 7 \cdot 4^n - 2^n.
\end{aligned}
$$

Examples

- Solve $T(n) = 3T(n-1) + n$ with $T(0) = 10$

Examples

- Solve $T(n) = 3T(n-1) + n$ with $T(0) = 10$

$$
\begin{aligned}
T(n) &= 10 \cdot 3^n + \sum_{i=1}^{n} 3^{n-i} \cdot i \\
&= 10 \cdot 3^n + 3^n \sum_{i=1}^{n} i \cdot 3^{-i}
\end{aligned}
$$

Examples

- Solve $T(n) = 3T(n-1) + n$ with $T(0) = 10$

$$\begin{aligned} T(n) &= 10 \cdot 3^n + \sum_{i=1}^{n} 3^{n-i} \cdot i \\ &= 10 \cdot 3^n + 3^n \sum_{i=1}^{n} i \cdot 3^{-i} \end{aligned}$$

Theorem. For any real number $x \neq 1$,

$$\sum_{i=1}^{n} ix^i = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(1-x)^2}.$$

Examples

- Solve $T(n) = 3T(n-1) + n$ with $T(0) = 10$

$$
\begin{aligned}
T(n) &= 10 \cdot 3^n + \sum_{i=1}^{n} 3^{n-i} \cdot i \\
&= 10 \cdot 3^n + 3^n \sum_{i=1}^{n} i \cdot 3^{-i} \\
&= 10 \cdot 3^n + 3^n \left(-\frac{3}{2}(n+1)3^{-(n+1)} - \frac{3}{4}3^{-(n+1)} + \frac{3}{4} \right. \\
&= \frac{43}{4}3^n - \frac{n+1}{2} - \frac{1}{4}.
\end{aligned}
$$

Next Lecture

- recurrence ...