# CS1004: Intro to CS in Java, Spring 2005

Lecture #27: Computation theory, AI,
The End…

Janak J Parekh
janak@cs.columbia.edu

## Administrivia

- Forgot to return EC on Tuesday, I'll return today
- Solutions for 4, 5 up by tomorrow
- Review session scheduling, take II
  - I'll email everyone the date as soon as I have it

## Computation theory

- How do we determine, theoretically, if a problem *has* an algorithmic solution?
- Develop a theoretical model of a *computing agent* that enables us to prove one way or another
  - Must capture the fundamental properties of a computing agent
  - Must enable the exploration of the capabilities and limitations of computation in the most general sense

## Properties of a Computing Agent

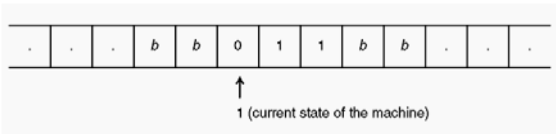
- A computing agent must be able to:
  - Accept input
  - Store information and retrieve it from memory
  - Take actions according to algorithm instructions
    - Choice of action depends on the present state of the computing agent and input item
  - Produce output
- Alan Turing invented the *Turing machine* in 1936, well before electronic computers
  - Considered the father of Computer Science; also largely responsible for cracking Enigma in WWII
  - Don't confuse a computing agent with computer architecture

## The Turing Machine

- A Turing machine includes
  - A (conceptual) tape that extends infinitely in both directions
    - Holds the input to the Turing machine
    - Serves as memory
    - The tape is divided into *cells*, which each contain one symbol from an *alphabet*
  - A unit that reads one cell of the tape at a time and writes a symbol in that cell

## The Turing Machine (continued)

- Alphabet for a given Turing machine
  - Contains a special symbol b (for "blank")
  - Usually contains the symbols 0 and 1
  - Sometimes contains additional symbols
- Input: A finite string of nonblank symbols from the alphabet
- Output: Written on tape using the alphabet
- At any time, the unit is in one of k *states*




S/G Figure 11.2
A Turing Machine Configuration

## The Turing Machine (continued)

- Each operation involves:
  - Write a symbol in the cell (replacing the symbol already there)
  - Go into a new state (could be same state)
  - Move one cell left or right
- Each instruction says something like:

  if (you are in state i) and (you are reading symbol j) then
    write symbol k onto the tape
    go into state s
    move in direction d

## The Turing Machine (continued)

- A shorthand notation for instructions
  - Five components
    - Current state
    - Current symbol
    - Next symbol
    - Next state
    - Direction of move
  - Form

    (current state, current symbol, next symbol, next state, direction of move)

## The Turing Machine (continued)

- A clock governs the action of the machine
- Conventions regarding the initial configuration when the clock begins
  - The start-up state will always be state 1
  - The machine will always be reading the leftmost nonblank cell on the tape
- The Turing machine has the required features for a computing agent

## A Model of an Algorithm

- Instructions for a Turing machine are a model of an algorithm
  - Are a well-ordered collection
  - Consist of unambiguous and effectively computable operations
  - Halt in a finite amount of time
  - Produce a result

## A Bit Inverter

- A bit inverter Turing machine
  - Begins in state 1 on the leftmost nonblank cell
  - Inverts whatever the current symbol is by printing its opposite
  - Moves right while remaining in state 1
- Program for a bit inverter machine

  (1,0,1,1,R)

  (1,1,0,1,R)

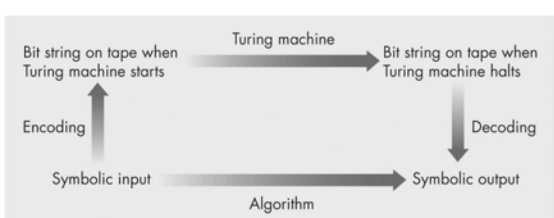
## A Unary Addition Machine

- A Turing machine can be written to add two numbers, using unary representation
  - Uses only one symbol: 1
  - Any unsigned whole number n is encoded by a sequence of n+1 ‘1’s
- Trick: “concatenate” the two numbers – need just to erase two ‘1’ digits and fill in the blanks between the two numbers
- The Turing machine program

| | |
|---|---|
| (1,1,b,2,R) | State 1 deals with removing the first 1 |
| (2,1,b,3,R) | State 2 deals with removing the second 1 |
| (3,1,1,3,R) | State 3 deals with filling in the blank |
| (3,b,1,4,R) | |

## The Church–Turing Thesis

- Key insight as to Turing machines
  - If there exists an algorithm to do a symbol manipulation task, then there exists a Turing machine to do that task
- Two parts to writing a Turing machine for a symbol manipulation task
  - Encoding symbolic information as strings of 0s and 1s
  - Writing the Turing machine instructions to produce the encoded form of the output
- Based on the Church–Turing thesis
  - The Turing machine can be accepted as an ultimate model of a computing agent
  - A Turing machine program can be accepted as an ultimate model of an algorithm




S/G Figure 11.9
Emulating an Algorithm by a Turing Machine

## The Church–Turing Thesis (continued)

- Turing machines define the limits of computability
- An uncomputable or unsolvable problem
  - A problem for which we can prove that no Turing machine exists to solve it

## Unsolvable Problems

- The halting problem
  - Decide, given any collection of Turing machine instructions together with any initial tape contents, whether that Turing machine will ever halt if started on that tape
  - If we could find such a program, we'd be able to actively avoid infinite loops and related crashes
- Traditionally, one uses a *proof by contradiction*
  - Assume that a Turing machine exists that solves this problem
  - Show that this assumption leads to an impossible situation

## The halting problem, part I

- Let there exist a Turing machine P that can take, *as input*, a program T (composed of Turing machine instructions)
- We want to know if T halts or not given some input t.
  - Ideally, P will write a 1 on the tape if it halts, and a 0 if it doesn't
- Now, write a Turing machine Q that runs P, and then:
  - *Doesn't halt* if P writes a 1 (how do we do this?);
  - *Does halt* if P writes a 0

## Key insight to the halting problem

- Finally, make a copy of Q (called Q') and use it *as input to Q itself*
  - If P finds that Q' halts, then Q won't halt
  - If P finds that Q' doesn't halt, then Q will halt
- But… Q' is equivalent to Q, so if P claims Q' halts – and Q doesn't – it's **wrong**
  - This is a contradiction
  - Yes, we "backed ourselves" into it, but believe it or not, the formal proof is airtight

## Yikes!

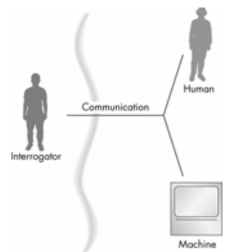
- OK, so the proof is kind of mind-bending and warped
  - Sadly, the book is a little *more* sophisticated about it
- Here's the compromise: just accept that the halting problem is unsolvable and understand its consequences, I won't ask you about the proof
- When you do get it, though, it's pretty neat

## Consequences of the halting problem

- No program can be written to decide whether any given program always stops eventually, no matter what the input
- No program can be written to decide whether any two programs are equivalent (will produce the same output for all inputs)
- No program can be written to decide whether any given program run on any given input will ever produce some specific output

# AI

- Artificial Intelligence explores techniques for incorporating aspects of intelligence into computer systems
- Simplest example of AI: the *Turing test*
  - A test for intelligent behavior of machines
  - Emacs has *M-x doctor*




# Classifying human tasks

- Computational tasks
  - Tasks for which algorithmic solutions exist by definition
  - We already know computers are better than humans
- Recognition tasks
  - Sensory/recognition/motor-skills tasks
  - Humans are better than computers
- Reasoning tasks
  - Require a large amount of knowledge
  - Humans are *far better* than computers
- AI seeks to bridge the gap by using algorithms

# Knowledge Representation

- In order to apply algorithms, we must first store *knowledge* (a body of facts or truths) and represent it
- For a computer to make use of knowledge, it must be stored within the computer in some form
  - Natural language: use *natural-language processing* (NLP)
  - Formal language: use *formal logic*; most common
  - Pictorial: use vision technologies
  - Graphical: use graph algorithms
- Goal: be adequate, efficient, extendable, and appropriate

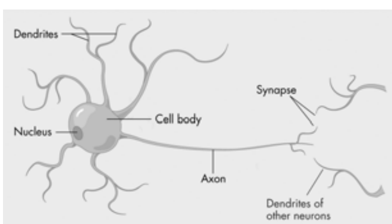
## Formal language

- From page 633
- Such encodings make it easier to process information
- Use of if-then like logic constructs

| | |
|---|---|
| Spot is a dog | dog(S) |
| Spot is brown. | brown(S) |
| Every dog has 4 legs. | $(\forall x)(dog(x) \rightarrow four\text{-}legged(x))$ |

## Recognition Tasks

- A neuron is a cell in the human brain, capable of:
  - Receiving stimuli from other neurons through its dendrites
  - Sending stimuli to other neurons through its axon

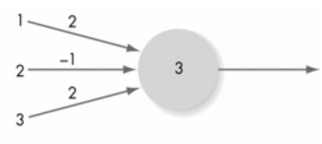



## Recognition Tasks (continued)

- If the sum of activating and inhibiting stimuli received by a neuron equals or exceeds its "threshold" value, the neuron sends out its own signal
- Each neuron can be thought of as an extremely simple computational device with a single on/off output
- Compare the human brain to a computer
  - Human brain: large number of simple "processors" with multiple interconnections
  - Computer: A small number (maybe only one) of very powerful processors with a limited number of interconnections between them

## Recognition Tasks (continued)

- Artificial neural networks (neural networks)
  - Simulate individual neurons
  - Connect them in a massively parallel network of simple devices that act somewhat like biological neurons
- *Neural network*
  - Each neuron has a threshold value
  - Incoming lines carry weights that represent stimuli
  - The neuron fires when the sum of the incoming weights equals or exceeds its threshold value
- Like hardware logic operators, but allows for "shades of grey"




S/G Figure 14.5

One Neuron with Three Inputs

## Recognition Tasks (continued)

- Both the knowledge representation and "programming" are stored as weights of the connections and thresholds of the neurons
- The network can learn from experience by modifying the weights on its connections
  - The algorithm tweaks the weights so that for a given input (say, picture or voice), we get the correct output
  - Surprisingly useful for image and voice recognition

## Reasoning Tasks

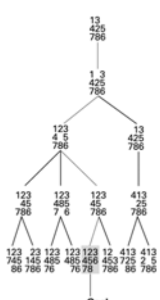
- Human reasoning requires the ability to draw on a large body of facts and past experience to come to a conclusion
- Artificial intelligence specialists try to get computers to emulate this characteristic, most commonly via *searching a state space*
- State-space graph:
  - After any one node has been searched, there are a huge number of next choices to try
  - There is often no complete algorithm to dictate the next choice
  - Finds a solution path through a state-space graph

## Intelligent Searching (continued)

- Each node represents a problem state
- *Goal state:* the state we are trying to reach
- Intelligent searching applies some heuristic (or an educated guess) to:
  - Evaluate the differences between the present state and the goal state
  - Move to a new state that minimizes those differences
- Example: games!

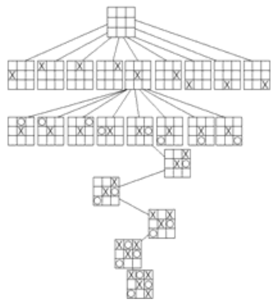
## Search tree for 9-puzzle

- This is just a partial search tree
- Represents one initial configuration
- Goal: to traverse the tree quickly enough and find the correct state
- Problem: tree can be very "wide"



## Search tree for Tic-Tac-Toe

- Again, partial search tree
- User might be the first move, followed by a computer move, etc.
- Goal: find a *winning* state
- Problem reduced to a data structure and a set of *search algorithms*
- Still many choices…



## Expert Systems

- Alternative: reason based on rule-based systems
  - Also called expert systems or knowledge-based systems
  - Attempt to mimic the human ability to engage pertinent facts and combine them in a logical way to reach some conclusion
- Must contain
  - A *knowledge base*: set of facts about subject matter
  - An *inference engine*: mechanism for selecting relevant facts and for reasoning from them in a logical way
- Many rule-based systems also contain
  - An explanation facility: allows user to see assertions and rules used in arriving at a conclusion

## Expert Systems (continued)

- A fact can be
  - A simple assertion
  - A rule: a statement of the form if . . . then . . .
- Inference engines can proceed through
  - Forward chaining: start with assertions and match if clauses/rules, which form new assertions
  - Backward chaining: given a conclusion, work backwards towards the initial set of assertions

## Conclusion

- Computing theory defines what's a computer, what's not, and what we can compute
- Artificial intelligence defines how computers are processing the information flows of the future
- Both boil down to the same thing: computers take information and work with them
- You've learned the basics of how to do this; everything else in CS just builds on the core algorithm skillset you learned here
- Hang on… two more slides…

## Final

- Structure: very similar to midterm, maybe about 50% longer – you shouldn't need all three hours, but you will have them
- The last two classes are technically fair game, but I'll "go light" on the material (i.e., factual)
  - Feel free to post in "exam discussion" if you're unsure if a particular topic is covered
- Review sessions next week: they'll be open-ended, so bring questions!

## Thank you!

- You guys have been a great audience
- I hope you found this class rewarding
  - Believe it or not, you guys are real programmers now
- Good luck with the rest of your Computer Science mini-careers!
  - And with finals
- Don't forget review sessions next week
- And fill out those evaluations