

Lecture 26: Review

CS442: Great Insights in Computer Science
Michael L. Littman, Spring 2006

1: Introduction

- Computers are everywhere creating new capabilities.
- Computer science is the study of “reduction”: making complex out of simple.
- **Skill:** Babbage’s difference engine computation.

2: Bits and Switches

- Bits: 0/1, simple but can represent anything
- **Skill:** Evaluate an expression using and/or/not.

3: Gates

- Bits can be made out of many things.
- Universal logic gates can simulate and/or/not.
- Gates can be defined in terms of simpler gates (reduction).
- HW1
 - Fill in difference engine table.
 - Evaluate a logical expression.
 - Fill in a truth table (with relay).

4: Binary

- Some simple logic gates.
- **Skill:** Recognize some simple functions of logic gates.
- Representing numbers in binary.
- **Skill:** convert between binary and decimal (base 10).
- **Skill:** adding numbers in binary.

5: Machine Language

- Fundamental circuits:
 - CPU interprets machine instructions.
 - Memory holds data and programs and is addressed in binary.
 - Arithmetic circuits perform mathematical calculations.
- Each clock cycle corresponds to a small instruction being executed.
- Machine-language program is made from bytes.
- Everything in the computer is made from these small instructions.

6: Subroutines

- *The hierarchy:* application programs -> software libraries -> high-level languages -> machine language -> logic blocks -> basic logic gates -> physical bits (transistors).
- Subroutines can package up repeated operations (like song choruses).
- **Skill:** Write a song as a series of subroutines.
- Parameters let you use the same subroutine for related tasks.
- The subroutine stack keeps track of the program's place.

7: Algorithms

- Sock matching.
- Different approaches to a problem can be faster than others!
- HW2:
 - Convert a truth table to a logical expression.
 - Convert and add binary numbers.
 - Count the number of gates in a (multilevel) logic block.

8: Growth Rates

- Computer scientists analyze algorithms by their running time as a function of the size of the input.
- Can sing generalizations of songs with n verses.
- Number of syllables is a pain to figure out as a function of n .
- Big-O notation (asymptotic growth rate) simplifies the process.

8: more

- Major classes of song growth rates:
 - constant-size verses: linear $O(n)$
 - verses grow because includes a number, which grows: linear-logarithmic ($O(n \lg n)$).
 - verses grow by a constant size: quadratic ($O(n^2)$).
 - verses grow by a constant size and include larger numbers: quadratic-logarithmic ($O(n^2 \lg n)$).
- **Skill:** Recognize growth rate of songs.

8: more

- **Skill:** Distinguish proper (growth rate) and improper (“big”) uses of the word “exponential”.
- The growth-rate classes are also very useful for analyzing algorithms like the sock sorter.

9: Compilers

- People write programs in languages that cannot be directly run on the computer.
- Compilers translate the programs into machine language that the computer can run.
- Parser captures the structure of the program, allows for optimizations and code generation.
- Interpreters don’t translate the program, but just simulate it themselves.
- Stack helps in the interpretation.

10: Graph Algorithms

- Google builds a map of the web by visiting web pages it knows about, then looking on those pages for the address of other pages.
- This operation is called “graph search”.
- Graph defined by nodes, links.
- Graphs terms: source, sink, path, cycle, connected components, tour, directed, undirected, tree.
- Node x is “reachable” from y if there’s a path from y to x .

11: Sorting Algorithms

- Sorting speeds up the search for information.
- Insertion Sort: Find the proper sorted place for each successive object. $O(n^2)$.
- Binary search: Ask a question that splits the remaining set of options in half. $O(\lg n)$.
- Merge Sort: Start with lots of small, sorted lists. Combine them pairwise. $O(n \lg n)$.
- Impossible to sort in fewer than $n \lg n$ comparisons.

HW3

- Is the use of “exponential” right?
- Given a generalized n -verse song, what is its syllable growth rate?
- How many comparisons does a sorting algorithm make?

12: NP-Completeness

- Formal problem: Define input, output, property.
- Decision problem: Output is yes / no (one bit).
- **Skill:** Define / recognize a formal problem.
- If a formal decision has a small (polynomial) “yes proof” it is in NP.
- Hardest problems in NP are “NP-complete”; no polynomial time algorithm is known. A polynomial-time solution to any NP-complete problem solves them all!
- Examples: Hamiltonian Path, Graph Coloring, Satisfiability, Traveling Salesperson, Subset Sum.

13: Heuristics

- Approximate solutions possible.
- Heuristic functions: give an answer without requiring a complete solution, especially useful in game playing.
- Local search: Make small local improvements via “hillclimbing” in hope that the process will end with a solution to the entire problem (not a local max).

14: Computability

- If an assumption leads to self contradiction, the assumption is broken.
 - Barber paradox, Godel’s theorem, Kantor’s diagonalization
- Assumption that we can detect whether a program loops forever leads to a program that loops forever and doesn’t loop forever.
 - The “halting problem” is not solvable.

15: Randomness

- Some problems solved simply and efficiently using random numbers (survey, quicksort).
- Seemingly random numbers can be generated using complex numerical mixing-up functions.
- Random bits useful for sending secret messages.
- Factoring useful for breaking secret messages.

HW 4

- Is a formula satisfiable?
- Is a problem valid or not?
- What's a short "yes" proof for a problem in NP?
- Will a given subroutine halt on all inputs?

16: Image Processing

- Pictures are sets of pixels, pixels are colors, colors are intensity triples (RGB), an intensity triple is 3 bytes, a byte is 8 bits. A picture is just a very long list of zeros and ones.
- Programs can modify these bits to change the picture: clip out a piece, brighten a patch of color, swap colors, flip things around.

17: Parallel Computation

- Moore's Law: Roughly, computers double in speed every year and a half.
- Some problems can be sped up by letting more than one computer work on them at a time.
- Some can't!
- Some others can, but only if you're clever.

18: Compression

- The number of bits that it takes to represent a message varies with the *encoding*.
- Finding a shorter encoding is “compression”.
- Huffman encoding uses few bits for common characters.
- **Skill:** Given a code and frequency counts for each character, what is the average number of bits needed per character?

19: Neural Networks

- Classifiers map feature vectors to yes/no.
- Classifiers can be created automatically (learned) by analyzing a training set of examples.
- Perceptrons use a linear threshold function and can solve linearly separable problems.

20: Reinforcement Learning

- Basic idea: Learn to act (via trial and error) to maximize some computed reward function.
- Some evidence people and animals work this way, too.

21: Genetic Algorithms

- Search algorithm using a population to find good solutions.
- Uses populations of individuals (often bitstrings) that mate if their fitness is sufficiently high to produce new generations of improved individuals.
- **Skill:** Mate two bitstrings and find the result.
- Nice summary of earlier concepts!

HW 5

- Identify how easily parallelized a task is.
- Calculate the number of bits needed to encode a message using a variable-length code.
- What father string could have produced the given offspring string?

23: Simulation & Modeling

- (no notes)
- Computers good at solving differential equations.
- Differential equations models physical systems (object interactions, sounds, appearance, motion).
- So, computers can simulate reality.

24: Bioinformatics

- DNA looks like bit sequences.
- Bases encode amino acids, which fold into proteins.
- Can find genes by searching for start/stop codons.
- Other gene-based activities greatly enhanced via computing technology.

25: Language Games

- Word games: Logic games involving words.
- Language games: Word meanings (and judgment) become important.
- Such programs limited by their *data*, not processing power.
-

26: Polynomiography

- Complex numbers are number pairs with a special rule for multiplication.
- Iterations of polynomial equations lead to interesting dynamics, which can be turned into pictures.
- These pictures are mathematically instructive and often pretty to look at.

HW 6

- What's the difference between a language game and a word game?
- Which program creates which robot behavior?
- How do you decode amino acids?

Next Time

- Final! May 5th, 12-3pm
- This Room.
- Open book.