Review

A selection of slides from throughout the term
Study guide for the final exam
The final exam will be 3:15 Wednesday June 14
two hour exam
emphasis on the second half of the course

Course Goals

- This course was intended to be an introduction to the field of computer science
  - what sorts of problems computer scientists study
  - some of the methods we use to solve those problems
- Although it wasn’t a “programming course” I wanted to have some lab projects that would give people some experience with a programming language
  - one of the best ways to understand a problem is to write a program that explores different ways of solving it
- The emphasis was (or should have been) on problems and their solutions
  - computer science is all about solving problems
  - it’s more than just “programming”

Course Topics

- Algorithms
- Automata
- "Big O" Notation
- String Search I: Exact Match
- String Search II: Indexed Search
- Relational Databases
- String Search III: Matching with Regular Expressions
- String Search IV: Inexact Match
- String Search V: BLAST
- Huffman Trees
- Hamming Codes
- Genetic Algorithms

Elements of an Algorithm

- Review: An algorithm is a description of a procedure to solve a problem
  - algorithms have well-defined inputs (starting conditions) and outputs (goals)
  - steps of the procedure must be
    - precise (simple and unambiguous)
    - effective (make progress toward the solution)
    - practical (can be completed)
  - the process must eventually terminate, otherwise the outputs are not defined

The idea that problems can be solved by algorithms is the central concept in computer science
Example: Insertion Sort

Here is a different way to sort cards, known as "insertion sort":
1. start with the card one space over from the left edge (the queen in this example)
2. use your right hand to pick up the current card
3. keep looking to the left until you find the first card lower than the one in your right hand, or the end of the poker hand, whichever comes first
4. insert the card in your right hand back into the poker hand at this location
5. move one place to the right of the original spot and go back to step 2

This description of a sorting process is more explicit about the "find" and "move" steps of the previous algorithm
But it's pretty verbose, and is still not very precise in places

Algorithms
Algorithms can be described in English....

Example: Insertion Sort

Here is a pseudo-code version of insertion sort:
input: a list A₀, A₁, ... Aₙ-1
for j = 1 to n-1
  key ← Aⱼ
  i ← j-1
  while i >= 0 and Aᵢ > key:
    Aᵢ+1 ← Aᵢ
    i ← i-1
  end
  Aᵢ+1 ← key
end
return A

This specification is much more concise
In the next set of slides we'll see how to represent lists in Ruby and how to write loops like these
Note this algorithm has two loops, one inside the other
these are called nested loops

Finite State Automata

Our simple state machine can be given a precise mathematical description

A finite state automaton is a system defined by
- S a set of states
- L a finite set of input symbols (an alphabet)
- δ a state transition function
- F a set of final states

One of the states (S₀) is the start state, and one or more states F ⊂ S are designated final states

See Chapter 2 of NTO

An automaton is a simple abstract machine
State Diagram

- We can use a state diagram to show the states and how the machine moves from one state to another.
- Circles represent states:
  - green: start state
  - red: final ("accepting") state
- Lines represent state transitions.
- Labels on lines are inputs.

Candy machine that sells packs of gum for 25¢

N = nickel, D = dime, Q = quarter

Automata

A state diagram is a useful visual representation of an automaton.

Failure Transitions (Finally)

- The final solution has links that remember the machine has seen "an" if a state was reached because the machine read an "a" and the next letter is an "n" go back to state 2.
- There are two places where this can happen in our FSA.
- The new transitions are shown in green.

Simple String Search

- The simplest algorithm to search for a substring scans from left to right:
  - Think of a "window" the size of the pattern sliding over the text.
  - At each step of the algorithm compare the two strings in the window.

```
p" = "p";
check next

"i" != "e";
moved window

"p" != "e";
moved window
```

Informal Definition of O

- Order of magnitude conveys some information about scalability.
- The blue line shows the execution time of an algorithm that is $O(n^2)$.
- The green line is one that is $O(n)$.
- For small problems the quadratic algorithm might be more efficient.
- But eventually at some problem size the linear algorithm will be faster.

```
f(n) = n^2/4
```

"Big O" notation describes the "complexity" of an algorithm.

```
0 10 20 30 40 50
0 100 200 300 400 500 600
```

f(n) = n^2/4

"Big O" notation describes the "complexity" of an algorithm.

We used automata as descriptions of several different algorithms this term (string matching, parity checks, ...)

We looked at three different algorithms for finding a pattern $P$ in a text $T$.
The Boyer-Moore Algorithm

- An algorithm defined by R. Boyer and J. S. Moore in 1977 use the “sliding window” approach of the simple algorithm, but with a small difference: 
  - scan the window from right to left

```
peter piper picked a peck of pickled peppers
```

The window starts at the left edge of the text, as in the simple algorithm.

- (1) simple match -- $O(n \times m)$
- (2) automaton -- $O(n)$
- (3) Boyer-Moore -- "sublinear"

Summary

- An index is an array of strings
  - if the array is not sorted you need a linear search to look up a string
  - in a sorted array you can use a binary search
  - a binary search tree is equivalent to a sorted array but may be easier to manage when new strings are added
- A hash table is an array with additional empty slots
  - use a hash function $h(S)$ to find the location for string $S$
  - make sure you have lots of extra slots to minimize the number of collisions

Indexing

- Searching is much more efficient if we can preprocess the text to build an index

Hash Function

- The idea is to store $S$ at an address computed by a hash function $h$
  - $h$ is a function that maps strings to integers
  - the table has room for $n$ items, so
    - $h: S \rightarrow 0..n-1$
  - Example:
    - $h(\text{"literal"}) = 2$
    - $h(\text{"expanding"}) = 5$
  - This table has room for 10 entries, but only 6 are filled

```
<table>
<thead>
<tr>
<th>string</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;literal&quot;</td>
<td>14, 307</td>
</tr>
<tr>
<td>&quot;creating&quot;</td>
<td>40</td>
</tr>
<tr>
<td>&quot;method&quot;</td>
<td>332</td>
</tr>
<tr>
<td>&quot;expanding&quot;</td>
<td>78, 333</td>
</tr>
<tr>
<td>&quot;class&quot;</td>
<td>357, 406</td>
</tr>
<tr>
<td>&quot;indexing&quot;</td>
<td>41</td>
</tr>
</tbody>
</table>
```

A hash table can reduce the search time to $O(1)$

Review

- Know the basic search methods (linear search, binary search)
- Given a sorted list show which items are compared during a lookup
- Given the definition of a hash function be able to show how a hash table is filled
- find the location based on $h(S)$
- Show the sequence of locations scanned when collisions occur

Expect questions about hash tables and indexing on the final exam
Database Management Systems

- A database management system (DBMS) is a set of applications for organizing and accessing information.
  - Client programs provide the interface to the system.
  - Clients can connect to servers via the internet.
  - Similar to the way browsers connect to web servers.
- Users access information by submitting a query.

Database systems make extensive use of indices to organize data.

Relational Operator: \( \sigma \)

- One operator in relational algebra is called “select”.
  - The symbol is \( \sigma \) (lower case sigma = Greek “s”)
- This operator makes a new table by choosing a subset of rows from an existing table.
- Specify selection criteria as a subscript, e.g. \( \sigma_{b > 1 \text{ and } b < 5} \)

A relational database is based on a branch of math known as relational algebra.

Review

- You should be familiar with the basic definitions of relational databases:
  - Table, column, record, key (index).
- Given the description of the contents of one or more tables, know what the results of the basic operations would produce:
  - Select, project, join.
- Be able to use MySQL queries to fetch information from a database:
  - Get a list of tables.
  - Get a description of a table.
  - Select rows from a single table.
  - Extra credit: write a query that does a join.

A regular expression is a string used as a pattern.

- A regular expression search looks for instances of the pattern in a text.
- Patterns are built up from:
  - Letters (which stand for themselves).
  - Character classes (\d, \w, \s).
  - Anchors (\^, $, \b).
  - Sets (\[\ldots\])
  - Other special symbols (+, *, ., {}).
- Ruby provides several opportunities to use regular expressions:
  - Scan, index, split, gsub, and other methods that have string arguments.
- Skills:
  - Give a short description of the kinds of strings a regular expression matches.
  - Write a regular expression given a description of a string.
  - Labs, problem sets, and exams will not use memory or other advanced features.

Regular expressions allow us to search for instances of a pattern instead of an exact string.
Regular Expressions in Ruby

- Ways to use regular expressions in Ruby

```ruby
>> s = "I can no longer shop happily"
>> r = /h[eio]p/
>> s.index(r)
=> 17
>> s[r]
=> "hop"
>> s.scan(r)
=> ["hop"]
```

I will provide a reference sheet (e.g. \w = "word char", \d = "digit")

Review

A useful extension to the string search algorithms we have been studying is to allow searches for text that is similar to the input pattern.

Definitions of “similar”:
- distance (number of characters)
- edit distance (number of copy/insert/delete operations)

A dot-plot is an informal (but useful) way to display two strings similarities show up as dots arranged on a diagonal.
- use this as a guide to create a sequence of edit operations

A dynamic programming algorithm systematically fills in a matrix with path costs
- guaranteed to find the optimal (lowest cost) path in \(O(n^2)\) steps
- a path corresponds to a sequence of editing operations

Distance

The \(k\)-mismatch algorithm gives us a simple way to define similarity
- recall the goal is to search a database for strings that are “similar” to a pattern \(P\)
- The similarity of strings \(S\) and \(T\) can be defined as the number of mismatches between the strings
- the number of mismatches is often called the distance between the strings
- Examples:

```
pilgrim d = 3
program
biology d = 4
boolean
```

Similarity is defined as number of differences (“Hamming distance”) when strings are the same length

Edit Distance Examples

- “mac” \(\Rightarrow\) “pc”
  - delete \(a\)
  - change \(a\) to \(p\)
  - \(d = 2\)

- “management” \(\Rightarrow\) “menagerie”
  - delete \(t\)
  - delete \(a\)
  - change \(a\) to \(e\)
  - change \(m\) to \(r\)
  - insert \(i\)
  - \(d = 5\)

- “prolog” \(\Rightarrow\) “perl”
- “palatino” \(\Rightarrow\) “alternate”

Similarity can be defined in terms of editing operations for strings of different lengths
Dot Plot Examples

Dot plots provide an informal method for finding the best sequence of editing operations.

Skills

- You should be able to
  - find the matches to a pattern found by k-mismatch
  - apply a sequence of editing operations to a string S
  - given S and T come up with at least one sequence (not necessarily the optimal sequence) that transforms S into T
  - evaluate the similarity of strings S and T if you are given a cost function and a sequence of operations

- Extra credit type opportunity (on the problem set):
  - given S and T and a cost function create the matrix, fill it in with costs, show the optimal path

Expect questions about distance, dot plots, alignment

Another Minimum Cost Example

The matrix for an earlier edit distance example: transform “human” to “chimpanzee”

An algorithm that uses a technique called dynamic programming finds the best solution

BLAST

- BLAST = Basic Local Alignment Search Tool
  - probably the most widely used of all bioinformatics applications
  - so common the name is now a verb: “we blasted our sequence against....”

- A BLAST database is a collection of sequences
- Users provide an input string (a query)
- BLAST will search the database by doing local alignments of the query with strings in the database
- Output is a set of alignments, sorted by match quality

BLAST is a program that uses similarity to search a sequence database

BLAST reports
Huffman Tree

- The codes on the previous slide were defined by a Huffman tree.
- A Huffman tree is a type of binary tree:
  - circles represent nodes
  - connections between nodes are labeled with bits
  - the root is at the top of the diagram (no nodes above it)
  - leaves are at the bottom (no nodes below them)
  - there is one leaf for each letter in the alphabet
  - The path from the root to a leaf defines the code for that letter.

A Huffman code saves space by using shorter codes for common letters.

BLAST Output

- The output from “standalone BLAST” is printed in the terminal window:

  ```
  >45443158 NC_005810
  Length = 496
  Score = 30.4 bits (67), Expect = 0.15
  Identities = 32/149 (21%), Positives = 64/149 (42%), Gaps = 21/149 (14%)
  Query: 77 --------LTYIMNPLGKRKINK--KELFEFLKI--KDFSISDENNPYTDFSLSNKFLYQ 178
  Positivity = 18/149 (12%)
  The general formula for figuring out how many bits you need to represent n symbols is
  \[ b = \lceil \log_2 n \rceil \]
  \( \lceil n \rceil \) means “round up to the next integer.”
  So for \( n = 20 \) letters we need
  \[ b = \lceil \log_2 20 \rceil = \lceil 4.3219 \rceil = 5 \]

  The most common encoding technique: fixed-length code.

Compact Codes (cont’d)

- A special code for protein sequences would need 5 bits per letter:
  - there are 20 amino acid letters
  - \( 2^1 = 16 \), so 4 bits don’t provide enough combinations
  - \( 2^2 = 32 \) combinations

  The codes on the previous slide were defined by a Huffman tree:

  A Huffman tree is a type of binary tree:
  - circles represent nodes
  - connections between nodes are labeled with bits
  - the root is at the top of the diagram (no nodes above it)
  - leaves are at the bottom (no nodes below them)
  - there is one leaf for each letter in the alphabet
  - The path from the root to a leaf defines the code for that letter.

A simple algorithm builds the Huffman tree given an alphabet and letter frequencies:

1. Repeat until there is only one node left in the list:
   - remove the first two items (call them \( n_1 \) and \( n_2 \))
   - make an interior tree node \( n \):
     - \( n_1 \) and \( n_2 \) will be the children of \( n \)
     - the frequency of \( n \) is the sum of the frequencies of \( n_1 \) and \( n_2 \)
     - label the connection to \( n_1 \) with 0 and the connection to \( n_2 \) with 1
   - insert \( n \) back into the list (keeping the list sorted by frequency)

   Note the list shrinks by one node on each step (remove two, insert one).

Generating a Huffman Tree (cont’d)

- After studying ways of searching text
- we looked at issues related to storing and transmitting text

The “hits” it finds are described by how well they align with the query sequence
Review

The main topics for today:
- **encoding** translates letters into sequences of bits
- **decoding** recovers letters from bit sequences
- ASCII codes (8 bits per character) are the default choice for text files
- Text can be compressed by using alternative codings
- Special-purpose codes can be designed for an application (e.g. 2-bit code for DNA)
- Variable-length codes are based on letter frequencies
- The Huffman tree algorithm can be used to generate variable-length codes

You should be able to:
- Encode or decode a string using ASCII
- Encode or decode a string given a drawing of a Huffman tree
- Create a Huffman tree for a small alphabet given a table of letter frequencies

Checksums vs. Parity Bits

 Adding the codes for the letters is the same as having 8 individual parity bits
 think of 8 wires transmitting the bits of each character in parallel
 the receiver has 8 FSAs, one for each column
 Using 8 parity bits (one per column) gives the receiver more information, helps it detect more errors

Distance Codes

With more bits (larger cubes) we can “scatter” the valid code patterns around the cube
- When the receiver reads an invalid pattern it can guess the correct pattern by moving to the closest valid code pattern
- In the general case, if $d$ is the minimum distance between two codes:
  - a receiver can detect up to $d - 1$ errors
  - the receiver can correct $\lfloor d/2 \rfloor$ errors by choosing the “closest” correct code
- Example: 5-bit code with only two valid words (00000, 11111) and $d = 5$
  - Suppose transmitter sends 00000
  - If 1 or 2 errors (any pattern with up to 2 1 bits)

### Calculating Parity: FSA (cont’d)

- The receiver will have its own machine
  - it also starts in state 0
- As the message is being read, this machine changes state according to the bits it receives
- When the message ends:
  - if the machine is in state 1 an error occurred
  - if the machine is in state 0 the message had an even number of bits; assume no

### Distance Codes and Error Correction (cont’d)

With more bits (larger cubes) we can “scatter” the valid code patterns around the cube
- When the receiver reads an invalid pattern it can guess the correct pattern by moving to the closest valid code pattern
- In the general case, if $d$ is the minimum distance between two codes:
  - A receiver can detect up to $d - 1$ errors
  - The receiver can correct $\lfloor d/2 \rfloor$ errors by choosing the “closest” correct code
- Example: 5-bit code with only two valid words (00000, 11111) and $d = 5$
  - Suppose transmitter sends 00000
  - If 1 or 2 errors (any pattern with up to 2 1 bits)
Review (cont’d)

- Skills: you should be able to
  - add a parity bit to a word
  - identify whether an error occurred by checking a parity bit
  - add a checksum to a message
  - given a distance code find the closest code word to an erroneous bit pattern
    - e.g. given a 3D or 4D hypercube drawing understand the links and find a path from an error code to a valid code

Expect questions about parity bits, checksums, distance codes

Genetic Algorithm

- Today’s talk is about a novel approach called a **genetic algorithm**
  - The basic idea:
    - create a “population” of possible solutions
    - the “fitness” of a solution depends on the cost
      - lower cost = better fitness
    - use “natural selection” to keep good solutions
    - replace bad solutions with new ones derived from the survivors

A genetic algorithm is a different kind of search — search an abstract space for solutions to a problem

Genetic Algorithm (cont’d)

- The key idea in a GA is that “individuals” represent problem solutions
- Generation of new solutions happens by:
  - mutation: make a copy of an existing solution and make a small change
  - cross-over: select two existing solutions, combine elements at random to produce a new solution
- In both cases the result must a complete solution

A solution “evolves” from random starting points

Solutions (cont’d)

- A simple way to represent the tour is to use a string
  - if there are \( n \) cities there are \( n \) letters in the string
  - tours of more than 26 cities would use arrays of integers, but strings are useful for small demos (easy to understand, easy to display)
  - for the small graph shown below strings would have the letters “A” through “G”

- Any string that is a permutation of these letters is a valid solution

For the traveling salesman problem we start with random permutations and evolve better ones
What is “computer science”?

- Computer science is
  - engineering
  - math
  - cognitive science
  - linguistics
  - business

Hopefully the topics we looked at this term help give you a good overview of some of the problems computer scientists study.

The Science of Computing

- The “science” in computer science includes
  - algorithms: what are the most efficient methods for solving problems?
  - languages: what are the best ways to express algorithms?
  - software engineering: how can we build useful and reliable programs?
  - computer engineering: how can we build cost-effective computer systems?

- Computer science helps people solve problems
  - science
  - engineering
  - medicine

- Computer science helps people be more effective or creative
  - architecture
  - communications
  - music and the arts

Viewpoint

Jeannette M. Wing

Jeanette M. Wing (wing@cs.cmu.edu) is the President’s Professor of Computer Science, in and head of the Computer Science Department at Carnegie Mellon University, Pittsburgh, PA.

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

Communications of the ACM, Mar. 2006