COM S/ENGRD 211 (“CS211”) Spring 2007

Lecture 1: Overview

http://www.cs.cornell.edu/courses/cs211

Announcements

- Did you get this lecture handout? (from now on, posted on-line)
- Assignment 1 (of 5) posted today on website, due 2/7
- Java Bootcamp, Upson B7, Tue 1/23 and Wed 1/24, 7:30-10:30pm (same content both nights)
- ACSU bowling at Helen Newman lanes this Thursday, Jan 25, 5-7pm. Free pizza. Come and meet current members, find out more about the organization and enjoy a few games of bowling.

Course Staff

- Instructors:
  - Professor Dexter Kozen kozen@cs.cornell.edu
  - Professor David I. Schwartz dis@cs.cornell.edu
- Administrative Assistant:
  - Kelly Patwell patwell@cs.cornell.edu
- More contact info?
  - See Staff on website

Student Course Staff

- Teaching Assistants:
  - Lead sections (“recitations”) starting next week
  - Act as your main contact point
- Consultants:
  - In Upson 360, hours online
  - “Front line” for answering questions
- More info?
  - See Staff on website

Lectures

- TR 10:10-11am, Olin 155
- Attendance is mandatory
- ENGRD 211 or COM S 211?
  - **Same course**! We call it CS211
  - Non-engineers sign up for COM S 211
  - Engineers sign up for ENGRD 211, but Engineering college doesn’t care which you sign up for
- Lecture notes will be online
- We will occasionally make small last minute changes to the notes
- Readings and examples will be posted online together with lecture notes

Sections

SEC 01 T 1220-0110P HO 320
SEC 02 T 0125-0215P UP 215
SEC 03 T 0230-0320P UP 211
SEC 04 W 1220-0110P PH 307
SEC 05 W 0125-0215P UP 109
~~SEC 06 R 0230-0320P HO 320~~ Canceled!
SEC 07 T 1220-0110P UP 211
SEC 08 T 0125-0215P HO 306
SEC 09 W 1220-0110P UP 211
SEC 10 W 0125-0215P OH 218

Sections

- Like lecture, attendance is mandatory
- Usually review, help on homework
- Sometimes new material
- Section numbers are different for Com S and ENGRD
- Each section will be led by a member of the teaching staff
- No permission needed to switch sections
- You may attend more than one section if you wish

CS212

- ***CS 212: Java Practicum***
- 1 credit project course
- Substantial project
- 1 lecture per week
- Required for CS majors; recommended for others
- Best to take 211 and 212 in the same semester

Online Resources

- Course web site
  http://www.cs.cornell.edu/courses/cs211
  - Watch for announcements
- Course newsgroups
  cornell.class.cs211, cornell.class.cs211.talk
  - Good place to ask questions (carefully)
- Textbook: Frank M. Carrano, *Data Structures and Abstractions with Java*, 2nd ed., Prentice Hall (1st edition is obsolete)
- Additional material on the Prentice Hall website

Obtaining Java

- We do not require an IDE
  - But we generally use Eclipse
- See **Help & Software** under **Java Resources** on website
- Use Java 5 (aka 1.5.0_10)
- Do not use Java 1.6!
  - Still in beta

Java Help

- CS 211 assumes basic Java knowledge:
  - classes, objects, fields, methods, constructors, static and instance variables, control structures, arrays, strings, exposure to inheritance
- Need review?
  - ***Java Refresher/Bootcamp***
    - self-guided tutorial—material (including solutions) on website (Help & Software)
    - Live help in Upson B7, Tue 1/23 and Wed 1/24, 7:30-10:30pm
    - Same material both days

Academic Excellence Workshops

- Two-hour labs in which students work together in cooperative setting
- One credit S/U course based on attendance
- Time and Location TBA
- See the website for more info

Course Work

- 5 assignments involving both programming and written answers (44%)
  - We A.I. check each homework assignment
  - The software is extremely accurate!
- Two prelims (15% each)
- Final exam (20%)
- Course evaluation (1%)
- Occasional quizzes in class (5%)

Assignments

- Assignments may be done by teams of two students (except for A1)
  - A1 will be posted today
- You may choose to do them by yourself
- Finding a partner: choose your own or contact your TA. Newsgroup may be helpful.
- Monogamy encouraged
- Mandatory reading: partner info and Code of Academic Integrity on website

Course Objectives

An introduction to computer science and software engineering

- Concepts in modern programming languages
  - recursive algorithms and data structures
  - data abstraction, subtyping, generic programming
  - frameworks and event-driven programming
- Algorithm analysis and designing for efficiency
  - asymptotic complexity, induction
- Concrete data structures and algorithms
  - arrays, lists, stacks, queues, trees, hashtables, graphs
- Organizing large programs

Using Java, but not a course on Java!

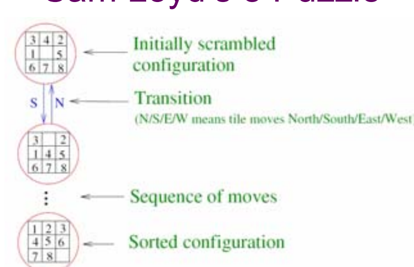
Lecture Sequence

- Introduction and Review
- Recursion and induction
- Object-oriented concepts: data abstraction, subtyping
- Data structures: Lists and trees
- Grammars and parsing
- Inheritance and frameworks
- Algorithm analysis, Asymptotic Complexity
- Searching and Sorting

More Lecture Topics

- Generic Programming
- Data Structures
  - Sequence Structures: stacks, queues, heaps, priority queues
  - Search Structures: binary search trees, hashing
  - Graphs and graph algorithms
- Graphical user interface frameworks
  - Event-driven programming
  - Concurrency and simple synchronization

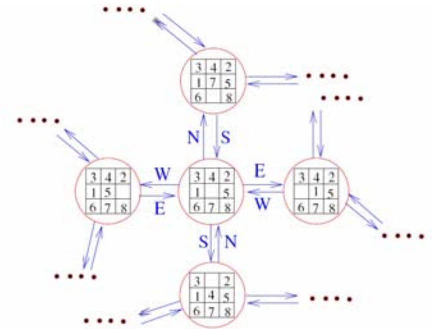
Sam Loyd's 8 Puzzle



Goal: Given an initial configuration of tiles, find a sequence of moves that will lead to the sorted configuration.

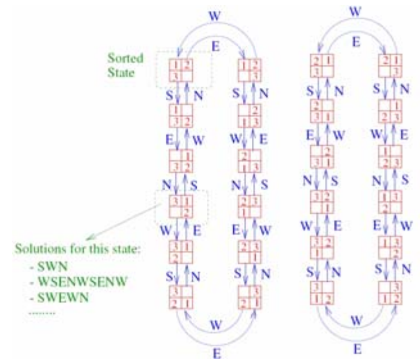
A particular configuration is called a ***state*** of the puzzle.

State Transition Diagram of 8-Puzzle



State Transition Diagram: picture of adjacent states.
A state Y is ***adjacent*** to state X if Y can be reached from X in one move.

State Transition Diagram for a 2x2 Puzzle



Graphs

- State Transition Diagram in previous slide is an example of a ***graph:*** a mathematical abstraction
  - ***vertices*** (or ***nodes***): (e.g., the puzzle states)
  - ***edges*** (or ***arcs***): connections between pairs of vertices
  - vertices and edges may be labeled with some information (name, direction, weight, cost, …)
- Other examples of graphs: airline routes, roadmaps, . . .
  - A common vocabulary for problems

Path Problems in Graphs

- Is there a path from node A to node B?
  - Solve the 8-puzzle
- What is the shortest path from A to B?
  - 8-puzzle (efficiently)
  - Mapquest
- Traveling salesman problem
- Hamiltonian cycles

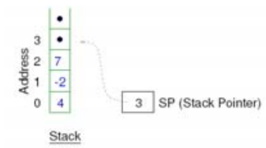
Simulating the 8-puzzle

- What operations should puzzle objects support?
- How do we represent states?
- How do we specify an initial state?
- What algorithm do we use to solve a given initial configuration?
- What kind of GUI should we design?
- How to structure the program so it can be understood, maintained, upgraded?

SaM

- ***SaM*** is a simple StAck Machine:
  - Similar to the Java Virtual Machine (JVM)
  - and to the machine code understood by processor hardware
  - Use it to understand how compilers work
- Download it from course homepage
- Used extensively in CS212

SaM's Stack



Note: For now, assume only integers can be pushed on stack. SaM actually allows floats, characters, etc. to be pushed, and it tracks type of data. GUI displays type (I:integer,F:float,. . .), but ignore this for now.

Stack: an array of integers
- Stack grows when integer is "pushed" on top.
- Stack shrinks when integer is "popped" from top.
- Stack starts at address 0 and grows to larger addresses.

Stack pointer (SP):
- first "free" address in stack
- stores integer address
- initialized to 0

Some SaM Commands

- All arithmetic/logical operations pop values from stack, perform operation, push result, and move SP to first free address
- Some commands:

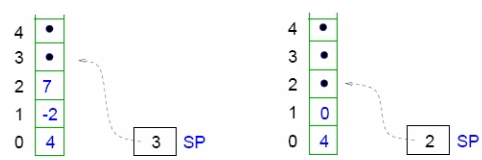
```
PUSHIMM int
    // push integer int onto top of stack
ADD
    // pops two values from top of stack
    // adds them and pushes result
SUB
    // pops two values (say top and below)
    // and pushes result of doing (below - top)
TIMES
    // works like ADD
GREATER
    // Boolean values are simulated using 0/1 (false/true)
AND
    // logical AND
STOP
    // terminate execution of program
```


Demonstrate SaM Commands

PUSHIMM 16

ADD:
- Pop 7
- Pop -2
- Determine 7 + (-2)
- Push result

Booleans and SaM



Booleans are simulated in SaM with integers:
- False → 0
- True → any int except 0 (usually 1)

GREATER:
- Pop two values (V_{top} and V_{below}) from stack (V).
- So, V_{top} = 7 and V_{below} = -2.
- If $V_{below} > V_{top}$ push 1; else push 0.
- In example, we would push 0.

SaM Programs

- Example 1:

```
PUSHIMM 5
PUSHIMM 4
PUSHIMM 3
PUSHIMM 2
TIMES
TIMES
TIMES
STOP // should leave 120 on top of stack
```

- Example 2:

```
PUSHIMM 5
PUSHIMM 4
GREATER
STOP //should leave 1 on top of stack
```


SaM Simulator

- What operations must SaM objects support?
- How do we represent the internal state of SaM?
- How do we load programs from a file?
- How do we write code to interpret each of the opcodes?
- How do we translate a high-level language like Java into SaM code?

Why you need CS 211

You will be able to design and write moderately large, well-structured programs to simulate such systems.

Computer systems are complex. Need CS to make them work; can't just hack it

- Selected software disasters:
  - CTAS air traffic control system 1991-present
  - Ariane 5 ex-rocket
  - Denver airport automated baggage handling

Why you need CS211, cont'd

Fun and intellectually interesting: cool math ideas meet engineering and make a difference.

- Recursion, induction, logic, discrete structures, …

Crucial to any engineering or science career

- Good programmers are >10x more productive
- Leverages knowledge in other fields, makes new possibilities
- Where will you be in 10 years?

Why you need CS211, cont'd

Real systems are large, complex, buggy, bloated, unmaintainable, incomprehensible.

Year	*Operating System*	*Millions of lines of code**
1993	Windows NT 3.1	6
1994	Windows NT 3.5	10
1996	Windows NT 4.0	16
2000	Windows 2000	29
2001	Windows XP	40
2005	Windows Vista Beta 2	50

Commercial software typically has 20 to 30 bugs for every 1,000 lines of code†

---

*source: Wikipedia

†source: CMU CyLab Sustainable Computing Consortium

Moore's Law

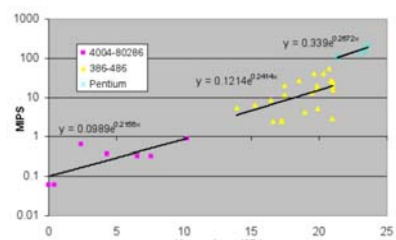


Figure 5: Processor performance in millions of instructions per second (MIPS) for Intel processors, 1971-1995.

From *Lives and death of Moore's Law*, Ilkka Tuomi, 2002

Grandmother's Law

- Brain takes about 0.1 second to recognize your grandmother
  - About 1 second to add two integers (e.g. 3+4=7)
  - About 10 seconds to think/write statement of code
- Your brain is not getting any faster!

Motivation

- Computers double in speed every 18 months
  - Software doubles in size every M Years
  - Data doubles in size every N Years
  - Your brain never doubles in speed
  - But we do get smarter, and can work in teams
- Computer science is increasingly important
  - Better algorithms
  - Better data structures
  - Better programming languages
  - Better understanding of what is (and is not) possible