Welcome to CSE 326: Data Structures & Algorithms

- Instructor: Rajesh Rao (rao@cs.washington.edu)
- TAs:
  - Vassily Litvinov (vass@cs)
  - Christophe Bisciglia (chrisrb@cs)
- Class web page for syllabus and course information:
  - http://www.cs.washington.edu/education/courses/326/
- Add yourself to the mailing list: cse326@cs
  - see class web page for instructions
- Textbook
  - *Data Structures and Algorithm Analysis in Java/C++* by Mark Allen Weiss

Today's Lecture

- Course Trivia and Logistics
- Course Topics
- Course Goals
- Overview of Selected Topics from Chapter 1
  - Recursion
  - Proof by Induction
- Class Survey

Grading, Homework, and other logistics

- Weekly homework assignments (50%)
  - Approximately 7 written/programming assignments
  - See website for policies on collaboration, etc.
- Midterm exam (20%)
  - Wednesday, February 12, 2003 in class
- Final (30%)
  - 8:30-10:20 a.m. Thursday, Mar. 20, 2003

Course Policies

- Written homeworks: Due at the start of class on due date
- Programming projects: Turn in electronically before 11pm on due date to graduate TA Vass (vass@cs)
- No late homeworks accepted
- Course labs are Sieg 232 and Sieg 329
  - Labs have NT machines with X servers to access UNIX
- All programming projects graded on UNIX
  - OK to develop using other tools (e.g. J++) but make sure you test under UNIX
- Work in teams only on explicit team projects
  - See policies on collaboration and cheating on website

Course Topics

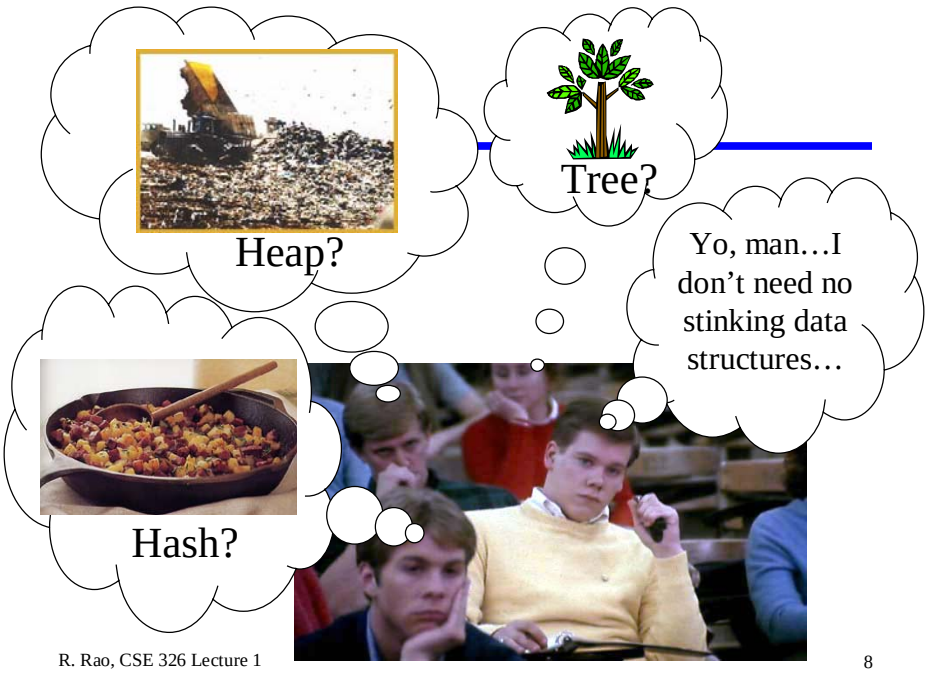
- Mathematical Preliminaries (Chap. 1)
- Introduction to Algorithm Analysis (Chap. 2)
- Review of Lists, Stacks, and Queues (Chap. 3)
- Trees and Search Algorithms (Chap. 4)
- Hashing and Heaps (Chaps. 5 & 6)
- Sorting out Sorting Algorithms (Chap. 7)
- Disjoint Sets and Union-Find (Chap. 8)
- Graph Algorithms (Chap. 9)
- Other Topics (Chaps. 10-12) as time permits

What is this Course About?

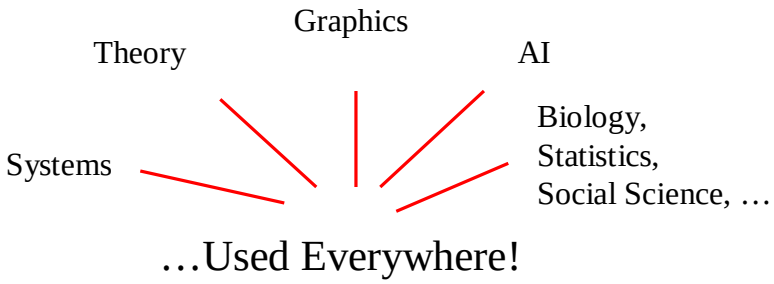
- Clever ways to organize information in order to enable efficient computation
  - What do we mean by clever?
  - What do we mean by efficient?

Clever? Efficient?

Data Structures	Algorithms
Lists, Stacks, Queues	Insert
Heaps	Delete
Binary Search Trees	Find
AVL Trees	Merge
Hash Tables	Shortest Paths
Graphs	Union (etc.)
Disjoint Sets	



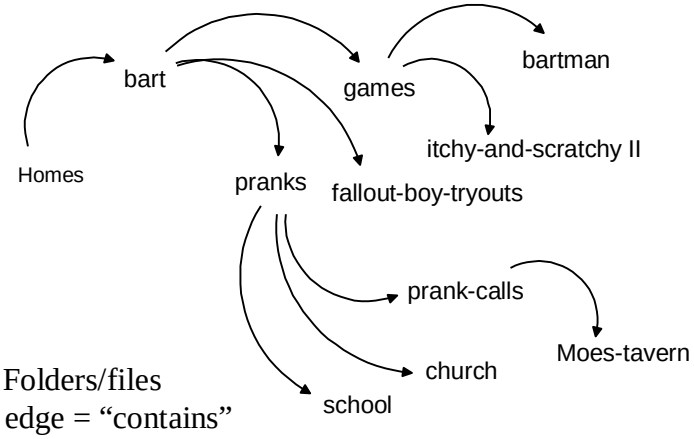
Data Structures with their Algorithms are…



…Used Everywhere!

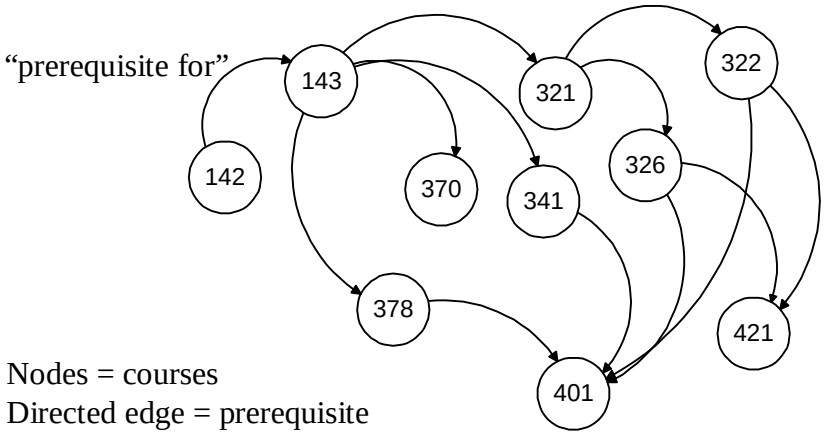
- ***Perhaps the most important course in your CS curriculum?***
- ***Guaranteed non-obsolescence!***

E.g. 0: Tree of Folders and Files



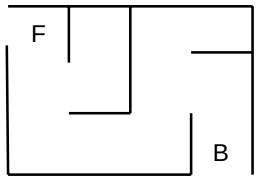
Nodes = Folders/files
Directed edge = "contains"

E.g. 1: Representing Course Prerequisites

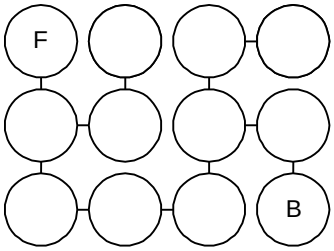


Nodes = courses
Directed edge = prerequisite

E.g. 2: Representing the Floor Plan of a House



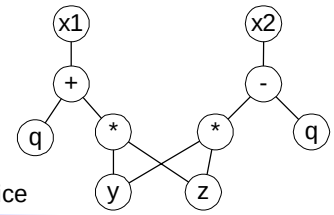
Nodes = rooms
Edge = door or passage



E.g. 3: Representing Expressions in Compilers

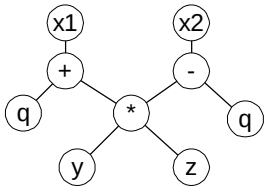
```
x1=q+y*z
x2=y*z-q
```

Naive:



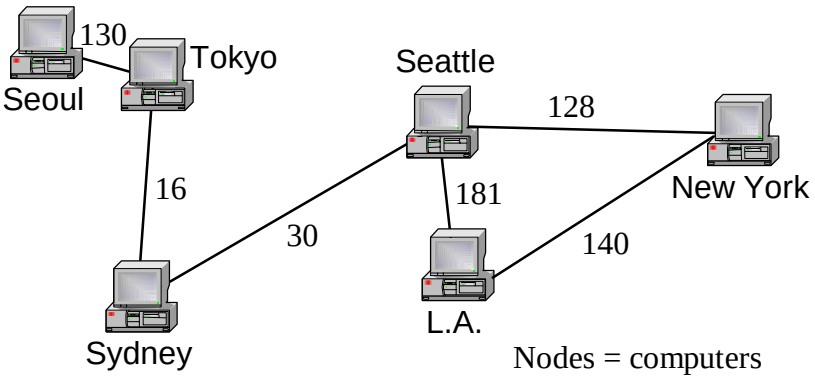
y*z calculated twice

common subexpression eliminated:



Nodes = symbols/operators
Edges = relationships

E.g. 4: Information Transmission in a Network

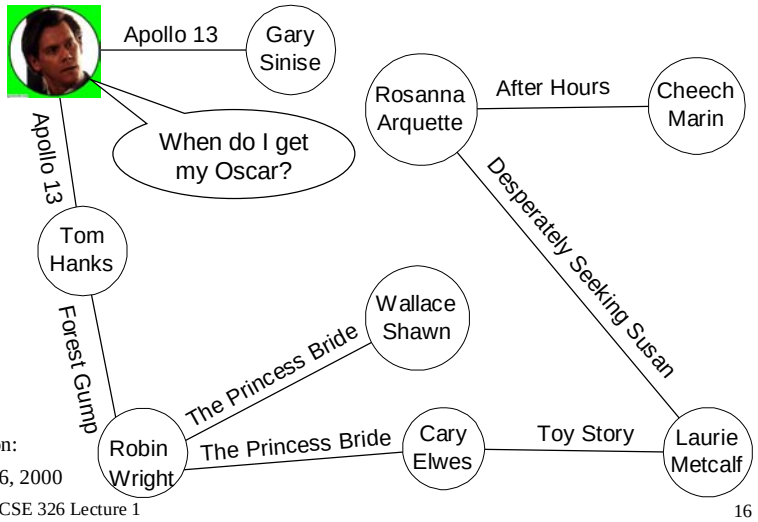


Nodes = computers
Edges = transmission rates

E.g. 5: Traffic Flow/Congestion on Highways

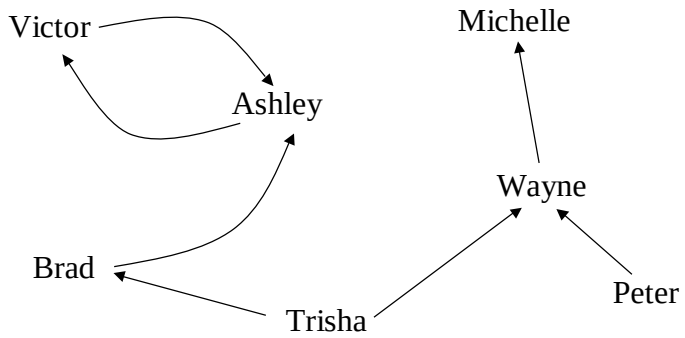
Nodes = cities
Edges = # vehicles on connecting highway

E.g. 6: Six Degrees of Separation from Kevin Bacon



Based on:
CSE 326, 2000

E.g. 7: Soap Opera Relationships



Data Structures (DS): What, How, and Why?

- Programs receive, manipulate, and output data
- Need to organize data according to problem being solved
  - Data structures are methods for organizing data
- Formal definition of DS: Abstract Data Type (ADT)
  - A "toolkit" of operations for manipulating data
  - E.g. A list with operations insert and delete

Data Structures (DS): What, How, and Why?

- Program design depends crucially on data organization i.e. how data is structured for use by the program
  - Implementation of some operations becomes easier or harder
  - Speed of program may dramatically decrease or increase
  - Memory used may increase or decrease
  - Debugging may be become easier or harder
- We will see examples of these throughout the course

Course Goals for Data Structures

- Study different implementation techniques for some fundamental ADTs
- Learn how to choose the “best” one
- Learn how to modify standard ADTs for specific problems, and create new ADTs

Algorithms and their Analysis

- What is an algorithm?
  - A sequence of steps (a “program”) that accomplishes a task
  - Independent of Programming Language
- Many different algorithms may correctly solve a given task
- But choice of a particular algorithm may have enormous impact on time and memory used
  - Time versus space tradeoffs are very common
- Choice of algorithm and choice of data structure for a task are often interdependent

Course Goals for Algorithm Analysis

- Understand the mathematical fundamentals needed to analyze algorithms
- Learn how to compare the efficiency of different algorithms in terms of running time and memory usage
- Study a number of standard algorithms for data manipulation and learn to use them for solving new problems

A Simple Example for Today's Class

Problem: Find the sum of the first num integers stored in an array v. (Assume num ≤ number of elements in v)

```
public static int sum ( int [ ] v, int num)
  {
  int temp_sum = 0;

  for (              ?              )
       temp_sum =            ?              ;
  return temp_sum;
  }
```


A Simple Example (cont.)

Problem: Find the sum of the first num integers stored in an array v. (Assume num ≤ number of elements in v)

```
public static int sum ( int [ ] v, int num)
  {
  int temp_sum = 0;

  for (  int i = 0; i < num; i++  )
       temp_sum += v[i] ;
  return temp_sum;
  }
```

Programming via Recursion

New Twist: Write a *recursive* function to find the sum of the first num integers stored in array v.

```
public static int sum ( int [ ] v, int num)
  {
                        ?

  }
```


Programming via Recursion

New Twist: Write a *recursive* function to find the sum of the first num integers stored in array v.

```
public static int sum ( int [ ] v, int num)
  {
    if (num == 0) return 0;
     else return sum(v,num-1) + v[num-1];
  }
```

- Simplifies the code dramatically
- Is the program correct?
  - *Proof by induction*

Proof of Program Correctness by Induction

```
public static int sum ( int [ ] v, int num)
  {
    if (num == 0) return 0;
     else return sum(v,num-1) + v[num-1];  }
```

Need to prove: sum(v,n) correctly returns the sum of the first n elements of array v, for any n≥0.

Basis Step: Program is correct for n=0: Returns 0 ✓

Inductive Hypothesis (n=k): Assume sum(v,k) returns sum of first k elements of v, i.e. v[0]+v[1]…+v[k-1]

Inductive Step (n=k+1): sum(v,k+1) returns: sum(v,k) + v[k] = v[0]+v[1]…+v[k-1]+v[k] = sum of first k+1 elements of v ✓

Next Class: Analysis of Algorithms

- How do we judge whether one algorithm for a problem is faster (more efficient) than another?
  - The "Big Oh" notation (next time)
- **Things to do this week**:
  - Visit course website
  - Sign up for mailing list (instructions on class website)
  - Do Homework #1 (on-line) – Due next Mon., Jan 13!
  - Read Chapters 1 and 2
- Please complete and hand-in the class survey before you leave!