Introduction

CSE 373
Data Structures

Staff

• Instructor
  › Steven L. Tanimoto,
    tanimoto@cs.washington.edu
• TA’s
  › Erik Vee, env@cs.washington.edu
  › Artem Zhurid, artem@cs.washington.edu

Steven L. Tanimoto

• Professor of Computer Science and Engineering
• Research: Applications of Visual Information Processing and Artificial Intelligence in Learning and Teaching

Web Page

• All info is on the web page for CSE 373
  › http://www.cs.washington.edu/373
  › also known as
    • http://www.cs.washington.edu/education/courses/373/04sp

Office Hours

• Steve Tanimoto – 638 CSE (Allen Center)
  › MF 2:30-3:20 or by appointment
• Erik Vee – to be announced
• Artem Zhurid – to be announced

CSE 373 E-mail List

• Subscribe by going to the class web page.
• E-mail list is used for posting announcements by instructor and TAs.
• It is your responsibility to subscribe. It will turn out to be very helpful for assignments hints, corrections etc.
Computer Lab

- Math Sciences Computer Center
  - [http://www.ms.washington.edu](http://www.ms.washington.edu)
- We’ll be using Java for the programming assignments.
  › Supports sharing on the web (with applets),
  › Makes it easy to display data structures graphically.

Textbook

- *Data Structures and Algorithms in Java, by Goodrich and Tamassia, 2nd (or 3rd) edition.*

Grading

*Estimated Breakdown:*
- Assignments 25%
- Project 20%
- Midterm 20%
  › Approximately May 3 (not definite yet)
- Final 30%
  › 2:30-4:20 p.m. Wednesday, June 9, 2004

Class Overview

- Introduction to many of the basic data structures used in computer software
  › Understand the data structures
  › Analyze the algorithms that use them
  › Know when to apply them
- Practice design and analysis of data structures.
- Practice using these data structures by writing programs.
- Data structures are the plumbing and wiring of programs.

Goal

- You will understand
  › what the tools are for storing and processing common data types
  › which tools are appropriate for which need
- So that you will be able to
  › make good design choices as a developer, project manager, or system customer

Course Topics

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues
- Search Algorithms and Trees
- Hashing and Heaps
- Sorting
- Disjoint Sets
- Graph Algorithms
Reading

- Chapters 1, 2, and 3, *Data Structures and Algorithms in Java*, by Goodrich and Tamassia
  - Very important chapter:
    - 3 on Analysis Tools

Data Structures: What?

- Need to organize program data according to problem being solved
- Abstract Data Type (ADT) - A data object and a set of operations for manipulating it
  - List ADT with operations insert and delete
  - Stack ADT with operations push and pop
- Note similarity to Java classes
  - private data structure and public methods

Data Structures: Why?

- Program design depends crucially on how data is structured for use by the program
  - Implementation of some operations may become easier or harder
  - Speed of program may dramatically decrease or increase
  - Memory used may increase or decrease
  - Debugging may become easier or harder

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of an object with set of operations on the object. Useful building block.
- Algorithm
  - A high level, language independent, description of a step-by-step process
- Data structure
  - A specific family of algorithms for implementing an abstract data type.
- Implementation of data structure
  - A specific implementation in a specific language

Algorithm Analysis: Why?

- Correctness:
  - Does the algorithm do what is intended.
- Performance:
  - What is the running time of the algorithm.
  - How much storage does it consume.
- Different algorithms may correctly solve a given task
  - Which should I use?

Iterative Algorithm for Sum

- Find the sum of the first `num` integers stored in an array `v`.

```plaintext
sum(v[ ], num: integer): integer{
  temp_sum: integer;
  temp_sum := 0;
  for i = 0 to num - 1 do
    temp_sum := v[i] + temp_sum;
  return temp_sum;
}
```

Note the use of pseudocode.
Programming via Recursion

- Write a recursive function to find the sum of the first num integers stored in array v.

  ```java
  sum (v[], num: integer): integer {
      if num = 0 then
          return 0
      else
          return v[num-1] + sum(v, num-1);
  }
  ```

Proof by Induction

- **Basis Step:** The algorithm is correct for the base case (e.g. n=0) by inspection.
- **Inductive Hypothesis (n=k):** Assume that the algorithm works correctly for the first k cases, for any k.
- **Inductive Step (n=k+1):** Given the hypothesis above, show that the k+1 case will be calculated correctly.

Program Correctness by Induction

- **Basis Step:** sum(v,0) = 0.
- **Inductive Hypothesis (n=k):** Assume sum(v,k) correctly returns sum of first k elements of v, i.e. v[0]+v[1]+...+v[k-1]
- **Inductive Step (n=k+1):** sum(v,n) returns v[k]+sum(v,k) which is the sum of first k+1 elements of v.

Algorithms vs Programs

- Proving correctness of an algorithm is very important
  - a well designed algorithm is guaranteed to work correctly and its performance can be estimated
- Proving correctness of a program (an implementation) is fraught with weird bugs
  - Abstract Data Types are a way to bridge the gap between mathematical algorithms and programs

Pseudocode

- In the lectures algorithms will be presented in pseudocode.
  - This is very common in the computer science literature
  - Pseudocode is usually easily translated to real code.
  - This is programming language independent