CSCB63

Design and Analysis of Data Structures

Topics

- Worst Case Complexity
- Balanced Search Trees
- Graphs and Graph Traversals
- Priority Queues and Heaps
- Disjoint Sets
- Amortized Complexity
- Average Case Analysis
- Hashing

Term Work

Assignments

- 4 each worth 5% for a total of 20%
- Basic understanding questions
- More challenging questions
- Only some of the questions will be graded
- Applied question (programming in C)

Term tests (3)

- 60min each held in Wednesday lecture
- Worth 15%, 15%, 15%
- Test 1 covers A1, Test 2 covers A2, Test 3 covers A3

Final Exam

- worth 35%

Textbook Options

Algorithm Design: Foundations, Analysis, and Internet Examples.

Michael Goodrich and Roberto Tamassia, John Wiley & Sons (2002), ISBN:0471383651.

Introduction to Algorithms (2nd edition).

Cormen, Leiserson, Rivest, Stein McGraw-Hill (2001),

ISBN:0070131511.

Free online access for U of T students:

http://main.library.utoronto.ca/eir/resources.cfm

Course Design

- Each week there may be *pre-lecture preparation* and *pre-lecture exercises*.

- In lectures we will have a mixture of *slides* and *practice*.

- Completing the *pre-lecture work* will maximize your learning in this course.

- *It will enable you to make the best use of lecture time.*

Resources

Communication

- Piazza: you will get an invite - all updates posted here.
- Website: mathlab.utsc.utoronto.ca/bretscher/b63
- Office Hours: Anna, TAs - on calendar

- Calendar: google calendar has all dates
- U of T email: I will send you important updates to your email.

ADTs vs Data Structures

- What is a data structure? ADT?
- Worst Case Complexity.
- Asymptotic Notation
  - Big “Oh”
  - Big “Omega”
  - Big “Theta”

ADTs and Data Structures

What does *ADT* stand for?

Abstract Data Type

What is an *ADT*?

A set of *objects* together with a set of *operations*.

Examples

Objects: integers

Operations ADD(x,y), SUBTRACT(x,y), MULTIPLY(x,y), QUOTIENT(x,y), REMAINDER(x,y)

Objects: stacks

Operations: PUSH(S,x) , POP(S), EMPTY(S)

Data Structures

What is a *data structure*?

A data structure is an *implementation* of an *ADT*.

This includes

- a way to *represent* objects and
- algorithms for the operations

Example

Objects: A *stack* could be *implemented* by either of

- a singly-linked list
- an array with a counter to keep track of the “top.”

Operations: How would you implement PUSH, POP and EMPTY in each of these implementations?

Motivation

Why are *ADTs* important?

- Important for *specification*.
- Provides *modularity*:
  - Usage *depends* only on the *definition*, not on the *implementation*
  - implementation of the ADT can be *changed* (corrected or improved) without changing the rest of the program
- Reusability
  - an abstract data type can be *implemented* once, and used in many *different programs*

Key Points

An *ADT* is a way to describe *WHAT* the data is and *WHAT* you can do with it.

A *data structure* is a way to describe *HOW* the objects are implemented and *HOW* the operations are performed.

Analysis of Data Structures and Algorithms

The *complexity* of an algorithm is the *amount of resources* it uses.

How do we *quantify* this?

We express the resources as a function of the *size* of the input.

Types of resources:

- Running time
- Space (memory)
- Number of logic gates (in a circuit)
- Area (in a VLSI) chip
- Messages or bits communicated (in a network)

Running Time

Input size will depend on the *type*

Examples

Strings: number of characters

Lists: number of elements

Graphs: number of vertices and edges

The *running time $T(n)$* of an algorithm for input of size n is the number of primitive operations or "*steps*" executed.

What are some examples of steps?

The notion of "*step*" should be machine *independent*.

Worst-Case Complexity

How do we *measure* the running time of an algorithm in terms of *input size* when there may be *many different inputs* of the same size?

1. Average-Case Complexity

   We'll explore this later…

2. Worst-Case complexity

For an *algorithm A*, let *t(x)* be the number of steps *A* takes on input *x*.

Then, the *worst-case time complexity* of *A* on input of size *n* is

$$T_{wc}(n) = max_{|x|=n}\{t(x)\}$$

In words:

Look at all inputs of size n and take the time of the one that is the worst (slowest).

Calculating the Worst Case

How can we determine the number of steps of an algorithm?

We could count every computational step…

- tiresome and machine dependent

What do we really care about?

- We care about when the *input size is very big*.
- We would like a measure that is *proportional to the input size* and not the machine.

Food for thought…

- Given an unsorted list of length n, would you rather use an algorithm that takes a multiple of

n steps, or

n^2 steps, or

$n \log n$ steps, or

$n \log n^2$ steps?

For Tomorrow

Remind yourself how *insertion sort* works.