Algorithm Efficiency

Information Flow

Instructor's Guide

Contents

Intro

Coursework

Resources

Table of Contents

Developed by the Teaching and Learning Laboratory at MIT
for the Singapore University of Technology and Design

Introduction

When to Use this Video

- In an introductory programming course, at home or in recitation.
- Prior knowledge: basic computer programming, recursion, and algorithms

Key Information

Duration: 15:04
Narrator: Prof. Charles Leiserson
Materials Needed:
• paper
• pencil

Learning Objectives

After watching this video students will be able to:

- Identify common resource limitations when programming.
- Understand speed and space and how they may be related.
- Understand how efficiency affects modern problem solving.

Motivation

- Modern computers are continually offering increasing computing speed, dynamic memory, and storage space. As a result, students may fail to grasp the importance of designing efficient algorithms.
- Programming demands are also continually growing, in many cases at a faster pace than computing resources. This video helps acquaint students with the scale of modern programming problems and convince them of the perpetual need for efficient design.

Student Experience

It is highly recommended that the video is paused when prompted so that students are able to attempt the activities on their own and then check their solutions against the video.

During the video, students will:

- Consider resources other than speed and space that must be utilized efficiently in programming.
- Calculate the number of function calls needed for a given inefficient solution.
- Brainstorm ways to improve the efficiency of a solution.
- Consider the efficiency bottlenecks of real-world programming problems.

Video Highlights

This table outlines a collection of activities and important ideas from the video.

Time	Feature	Comments
1:15	Introduction and motivation	Dr. Leiserson introduces the concept of efficiency and discusses why it's important to think about speed and space in particular.
4:08	Classic algorithm example: finding Fibonacci numbers three ways	A function for finding the *n*th number of the Fibonacci sequence is implemented three ways, with each subsequent implementation improving on one aspect of efficiency.
10:00	A look at how computer resources have changes over the decades	Processor speeds and RAM have increased by many orders of magnitude since the first computers from the 1970s.
11:18	Motivating example: electronic trading	This is an opportunity for students to understand the type of timescale that can matter in certain applications.
12:30	Motivating example: genome alignment	This is an opportunity for students to understand the large size of data that may be involved with certain applications.
13:52	Motivating example: smartphone applications	This example taps into many students' first-hand experience with the bottleneck of speed and space. Students may better appreciate the need for efficiently operating applications if they are dealing with them daily on their personal smartphone devices.

Video Summary

In this video, MIT professor of Computer Science and Engineering Charles Leiserson explains the importance of speed and space efficiency in programming. He guides students through different methods of computing the Fibonacci sequence and discusses the differences in efficiency of each version. Next, students are presented with several modern programming scenarios. Opportunities for pausing the video are provided so that students may independently consider and appreciate the different factors of efficiency in each example.

Course Materials

Pre-Video Materials

Students should be able to write simple programs before watching this video. The following pre-video activities could be used to reinforce this skill.

When appropriate, this guide is accompanied by additional materials to aid in the delivery of some of the following activities and discussions.

1. Basic programming refresher: Factorial

In mathematics, the factorial of a non-negative integer n is equal to the product of all positive integers less than or equal to n. For example, 4! is equal to 4 x 3 x 2 x 1 = 24.

(a) Solve 5!

(b) Solve 6!

(c) Using pseudocode or a programming language you are comfortable with, write a function for *factorial(n)*.

2. Fibonacci numbers

The Fibonacci Sequence is defined as *Fib(0)=0*, *Fib(1)=1*, and *Fib(n) = Fib(n-1) + Fib(n-2)* for *n>=2*. Write out the Fibonacci sequence from n=0 through n=10.

Post-Video Materials

Use the following activities to reinforce and extend the concepts in the video.

1. Consider the following version of the Fibonacci function using memoization from the video:

```
#variable for memoization
declare table

Function Fib-memo(n)
{
        if( entry for n exists in table ){ return entry; }
        elseif( n==0 ) return 0;
        elseif( n==1 ) return 1;
        else
        {
                f = Fib_memo(n-2) * Fib_memo(n-1);
                store [n,f] in table;
                return f;
        }
}
```

(a) Recall that Fib-memo of n=8 utilized 15 function calls and 8M space. Draw the corresponding function call tree for Fib-memo of n=9. How many function calls occur? How much space is used?

(b) Using pseudocode or a language you are comfortable with, write the code for the iterative version of Fibonacci presented in the video.

(c) How many function calls and how much space is used for the iterative version of Fibonacci of n=9?

2. Recall the genome alignment example from the video, where 5GB of the RAM was utilized by the raw data. Keep in mind that in reality, it is not uncommon for the amount of data to even exceed 8GB of space! Let's continue on working with 30 million reads, which require 2GB of space. Say we want to sort the reads using various basic sort algorithms you may have already been introduced to. The following table summarizes the average time (where n=the number of reads to be sorted) and average space (where m=the size of the data set) required in addition to the space already used by the data itself.

(a) How much time is required for each algorithm? Which algorithm(s) perform the best?

(b) How much space is required for each algorithm? Which algorithm(s) perform the best with respect to both speed and space?

(c) In the worst-case scenario, Quick Sort requires n2 time and n space. How much more time and space is this than the average case? What happens to time and space if you are sorting 60 million reads?

Algorithm	Average Time	Average Space
Quick Sort	n*log(n)	log(m)
Heap Sort	n*log(n)	1
Insertion Sort	n^2	1

3. In the video, we only touched briefly on how speed and space efficiency can have consequences on energy usage. One way to measure how much work a program requires is by using CPU hours, or the amount of time a processor needs to operate at full capacity in order to finish the task. Consider a modern programming task that uses 1000 CPU hours to run.

(a) If the job is run on a computer cluster rated at 1400 Watts (W), how much energy (in kilowatt-hours, or kWh) is used? Hint: use the estimate that a 40W appliance constantly running uses 1kWh/day from the electricity grid.

(b) Consider that 1 gallon of gasoline contains 36.6kWh of energy, only 15% of which is actually utilized to move a car down the road. If an average midsize car can travel 24.7 miles on a gallon of gas, how much energy (in kWh) is used to travel 380 miles, or roughly the distance from San Francisco, CA to Los Angeles, CA? How many times can you travel this distance using 1000 CPU hours-worth of energy?

(c) How many gallons of gasoline are needed to run the cluster per hour?

(d) Suppose you can parallelize your program to use four processor cores, effectively reducing the number of CPU hours by one fourth to 250. What is the corresponding distance you can travel by car?

Additional Resources

References

Algorithmic efficiency is an extremely complex topic with many rich sub-fields of study. A good starting point for beginning students is MIT Open CourseWare's Introduction to Algorithms class. Course materials, including lectures, reading, and assignments are available online:

- Leiserson, Charles, and Erik Demaine. 6.046J Introduction to Algorithms (SMA 5503), Fall 2005. (MIT OpenCourseWare: Massachusetts Institute of Technology), http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005 (Accessed 4 Dec, 2013). License: Creative Commons BY-NC-SA

An in depth look at programming the Fibonacci in python and stepping through function calls is available:

- Khan Academy: "Recursive Fibonacci Example". Retrieved July 28, 2013, from https://www.khanacademy.org/science/computer-science/v/recursive-fibonacci-example

Information on gasoline and engine efficiency were obtained from the following:

- Energy available in gasoline: HowStuffWorks "How Gasoline Works." Retrieved July 28, 2013, from http://science.howstuffworks.com/gasoline2.htm
- Energy usage from gasoline: Energy Fuel Economy: Where the Energy Goes. Retrieved July 28, 2013, from http://www.fueleconomy.gov/feg/atv.shtml
- Average miles per gallon for a light-duty vehicle: Sustainable Worldwide Transportation - Eco-Driving - sales-weighted fuel economy. Retrieved July 28, 2013, from http://www.umich.edu/~umtriswt/EDI_sales-weighted-mpg.html