Algorithms - Day 1

Instructor: Pat Devlin — prd41@math.rutgers.edu

Summer, 2016

Overview

An *algorithm* is a step-by-step process for accomplishing something. Consider for example the following very simple (and silly) algorithm.

Example 1 Pat has a canvas bag with items in it. What does he want you to do with these items?

Or as perhaps a more interesting example, consider the following problem.

Example 2 (Search problem) Pat has a secret list of 31 distinct real numbers. Each number is covered up so that you can't see it, but Pat promises that the list is sorted by size (increasing from left to right). You know that the number 100 is *somewhere* in that list, and your goal is to find it. You are allowed to uncover any number you want, but every time you do this, you need to give Pat $1.

What do you think you should do to find the number while paying as little as possible? If you used your strategy, how much would it cost you?

Here is a very silly algorithm for the search problem.

```
Silly_Search:

begin
  Uncover all of the numbers.
  Find the number we want.
end
```

Although `Silly_Search` seems to solve the problem, we may be concerned that it is too expensive. A slightly better algorithm might be the following.

```
Better_Search:

begin
  Uncover the numbers one at a time.
  If we find the number we want, then stop. Otherwise, uncover the next number.
end
```

Whenever we come up with an algorithm, we should always ask ourselves three questions.

(1) Does the algorithm work?

(2) How good is it?

(3) Can we do better?

Question 3 Does `Silly_Search` work (i.e., does it always make sense, and will it always solve the search problem)? How much does it cost in the best-case scenario? How much in the worst-case scenario? What about `Better_Search`? What about your algorithm?

An algorithm can be written down and described in many ways. For us, it doesn't matter how you do it as long as it is unambiguous and easy to understand. Here some ways you could do this.

- in words (see previous page)
- with a flow chart

- with a *decision tree* [follow from top to bottom, and the leaves have output]

Example 4 (Guess the thing) The game "twenty questions" is a guessing game for two people. They agree on a set of objects S, and the first player secretly selects an element of S. Then the second player asks "yes/no" questions until she is able to identify which object the first player selected.

Make and analyze an algorithm to guess which of the following objects I'm thinking of

$$S = \{\text{cat, dog, pencil, pizza, vuvuzela}\}.$$

A *quick* review of logarithms: When someone says $\log_b(x) = y$ is true, they mean $b^y = x$. In computer science, we often write $\log_2(x)$ as simply $\lg(x)$. For example,

x	$\lg(x)$
1	0
2	1
3	1.5849625...
4	2
5	2.321928...
6	2.5849625...
7	2.8073549...
8	3

A few important properties of logarithms are

$$\lg(xy) = \lg(x) + \lg(y), \qquad \lg(2^x) = x, \qquad \log_b(x) = \frac{\lg(x)}{\lg(b)}.$$

For us, the important things to know about $\lg(x)$ are that (a) $\lg(x)$ is a very small number relative to x, and (b) $\lg(x)$ is essentially the number of times you need to cut x in half until you get a number less than 2.

Lower bounds: answering 'can we do better?'

Often when we want to understand how well an algorithm does, we are concerned about how long it takes in the worst-case scenario. The most useful result about this is the following:

Theorem 5 (Information theory lower bound) *Suppose an algorithm needs to figure out which of N possible options is correct, and it can only ask "yes/no" questions. Then there must be a scenario where it has to ask at least $\lg(N)$ questions.*

Proof:

Question 6 What does the information theory lower bound say about the 'guess the thing' problem?

Theorem 7 (Information theory lower bound: generalized) *Suppose an algorithm needs to figure out which of N possible options is correct, and it can only ask multiple-choice questions that have k different options. Then there must be a scenario where it has to ask at least* $\log_k(N) = \lg(N)/\lg(k)$ *questions.*

[Proof is left as an exercise]

Question 8 What does this information theory lower bound say about the search problem? Why can't we use $k = 2$ for this? How does this bound compare to your algorithm?

Question 9 Suppose we want an algorithm that can distinguish between $N = n!$ different possibilities, and at each step, it can only ask a "yes/no" question. Find a lower bound on how many questions it must ask in the worst-case scenario. (Note: I'm asking you to approximate $\lg(n!)$. Try to get a lower estimate as well as an upper estimate on this.)