CS50 Algorithms

Overview

Recall that computing involves taking some form of input, and then processing that input in order to produce some form of output. Processing input will often require the use of an **algorithm**, which are sequences of instructions that can be executed by a computer. In computer science, these algorithms are usually written in code. Computers depend upon such algorithms in order to perform tasks. In many cases, multiple different algorithms can be used to achieve the same result. However, in some cases, one algorithm will be faster than another at arriving at the correct result.

Key Terms

- algorithm
- correctness
- efficiency
- loops
- conditions

```
 1 pick up phone book
 2 open to first page of phone book
 3 look at names
 4 if “Smith” is among names
 5       call Mike
 6 else if not at end of book
 7       flip to next page
 8       go to line 3
 9 else
10       give up
```

A Correct Algorithm

Algorithms are just sequences of steps that a computer can follow in order to translate input into output. Algorithms can be expressed in English as a detailed list of steps.

Take, for example, the task of finding a name (e.g. “Mike Smith”) in a phone book. One possible algorithm (represented to the left) involves picking up the phone book, opening to the first page, and checking to see if Mike Smith is on the page. If he’s not, flip to the next page, and check that page. Keep repeating this until you either find Mike Smith or get to the end of the book.

This algorithm is **correct** — if Mike Smith is in the phone book, then this algorithm will succesfully allow someone to find him. However, algorithms can be evaluated not only on their correctness but also on their **efficiency**: a measure of how well an algorithm minimizes the time and effort needed to complete it. The one-page-at-a-time algorithm is correct, but not the most efficient.

We could improve the algorithm by flipping two pages at a time instead of one — though we’d have to be careful of the case where we might skip over the page with Mike Smith’s name, at which point we’d have to go back a page. But even this algorithm is not the most efficient.

```
 1 pick up phone book
 2 open to middle of phone book
 3 look at names
 4 if “Smith” is among names
 5     call Mike
 6 else if “Smith” is earlier in book
 7     open to middle of left half of book
 8     go to line 3
 9 else if “Smith” is later in book
10     open to middle of right half of book
11     go to line 3
12 else
13     give up
```

An Efficient Algorithm

Consider instead what might be a more intuitive, and efficient, algorithm. First, open to the middle of the phone book. If Mike Smith is on the page, then our algorithm is done. Otherwise, taking advantage of the fact that the phone book is sorted, we know which half of the book Mike Smith will be in, if he’s in the book at all. Thus, we can eliminate half of the book, and now repeat our algorithm using a book that’s effectively half the size. We can repeat this process until we arrive at a single page, which either does or does not have Mike Smith’s name on it. This algorithm is pictured above (below our original algorithm).

This algorithm is more efficient than the original algorithm. Consider what would happen if a 500 page phone book were to double in size the next year. Using the original algorithm, it might take 500 more steps to go through 500 more pages. However, using the second algorithm, it would only take 1 additional step when searching through a phone book that’s twice the size.

Note that our algorithms have made use of several programming constructs, including **loops**, which repeat steps multiple times, and **conditions**, which only executes certain steps if some statement is true.