CS 42—Analysis

Tuesday, October 16, 2018

Summary

Today, we'll review **branching recursion**, also known as **"use it or lose it"**. This technique helps us solve difficult problems, using a systematic approach.

We'll also pick up our discussion from last time about **asymptotic analysis** and **Big O**—ideas from Math that help us talk about the scalability of a problem (or a solution to that problem). In other words, Big O lets us talk about what happens when the inputs to our problems get very, very big.

Finally, we'll talk about **recurrence relations**—a mathematical technique that we can use to reason about the behavior of recursive functions.

Recap: use it or lose it

Use it or lose it is a recursive problem-solving technique, also known as "branching recursion" because the technique relies on (at least) *two* recursive calls. Each call is a separate "branch" of the recursion, which we use to

Before we write code for a use-it-or-lose-it solution, we should fill out the following template:

> What is / are the base case(s)
>
> For the recursive case(s):
>
> > What is "it"?: a piece of the problem
> > Smaller version: What does the input look like without "it"?
> > Lose-it solution: How could we solve a smaller version with*out* "it"?
> > Use-it solution: How could we solve a smaller version *with* "it"?
> > How would we combine the solutions to solve the full problem?

Recap: How can we tell if a solution is good?

First, we should clarify: what do we mean by "good"? Then, we should ask: how will we measure "goodness"?

Empirical data (i.e., observations), but it can be messy and incomplete. A **theoretical model** (i.e., math) abstracts over the messiness, but it is lossy.

In CS 42, we'll call this abstraction a **cost metric**. A cost metric:

- corresponds to one "step": one unit of work
- highlights the essence of the work (e.g., multiplications, comparisons, function calls…)
- serves as a proxy for an empirical measurement

Good science requires good empirical data and good theoretical models. With both in hand, we can make predictions and communicate with other scientists. In CS 42, we will learn two techniques for building and communicating theoretical models: asymptotic analysis (e.g., Big O) and exact formulae (e.g., recurrence relations).

Recap: Asymptotic analysis

Key question: How does the cost of a problem scale (when we give larger and larger inputs to the problem)?

Informally, we say that the cost of a problem is "in **Big O** of *f*" if *f* is a function that describes a *reasonable upper bound* on (an abstraction of) a problem's difficulty or a solution's performance, for *reasonably large input sizes*.

In other words, Big O is all about scalability: the performance in the limit. If we care only about the limit, then differences between n and $n + 1$ don't matter. We say that "both n and $n + 1$ are in $O(n)$." That is, in the limit, they behave the same: computing an answer for an input of size $2n$ takes a constant factor longer to compute that it does to compute an answer for an input of size n.

Recurrence relation

A **recurrence relation** is a mathematical function that is defined in terms of itself.

In Computer Science, we use recurrence relations to describe a theoretical model of the cost of a recursive function.

There is a straightforward procedure for translating recursive functions to recurrence relations:

1. Choose a cost metric.
2. Define what we mean by "the size of the input" (e.g., the value of the input or the size of a list)
3. For each base case size, define the cost of computing the result.
4. For each recursive call, define the cost of computing the result of *this iteration* of the recursive call plus the cost the recursive call on a smaller input size.

Here is an example of a recurrence relation:

$$T(0) = 1$$
$$T(N) = 1 + T(N\text{-}1)$$

T is our **cost function**, and it is recursively defined. The value that we pass to the cost function is the size of an input. The result of the cost function is measured according to our cost metric. For example, if our cost metric is "number of additions", then this recurrence relation says: "An input of size 0 costs 1 addition. An input of size N costs 1 addition, plus the cost of an input of size N-1."