1 Searching

A fundamental operation in computer science in search.

Linear Search

Suppose we have a list of strings


and we want to be able to find a string in the list that begins with a certain prefix. Call this function

find_startswith(last,searchstr)

and consider it’s natural definition below:

```python
1 def find_startswith(lst,searchstr):
2     for s in lst:
3         if s.startswith(searchstr):
4             return s
5     return None
```

Question 1. In the worst case, if lst has n elements, how many elements will find_startswith examine?

Binary Search

Now suppose that the list of bands were sorted lexicographically by name. Lexicographically just means alphabetically. We can search through this sorted list much more efficiently. Consider the following version of find_startswith that performs what computer scientists call a binary search on the list.

```python
1 def find_startswith(lst, searchstr):
2     low = 0
3     high = len(lst)−1
4     while (low < high):
5         mid = (high + low) // 2
6         if lst[mid].startswith(searchstr):
7             return lst[mid]
8         elif lst[mid] < searchstr:
9             low = mid+1
10         else:
11             high = mid−1
12     return None
```

Question 2. In the worst case, if lst has n elements, how many elements will find_startswith examine?
Approximating Roots

How do we calculate the square root of a number $x$? In this section we’ll develop two algorithms to calculate $\sqrt{x}$—one based on binary search and another based on tangent approximations, which is often called Newton’s Method.

Bisection Method

Let $x$ be a non-negative real number. Searching for $\sqrt{x}$ can be viewed as a search on the real number line between $(0, x)$ where, with each iteration, we can eliminate half of the remaining candidate square roots. Here we start with low=0 and high=x and choose the midpoint $m=\frac{\text{low}+\text{high}}{2}$ as our candidate square root value. If this value is larger than the actual square root, we can use it as our new high value; if it’s smaller, we can use it as our lower value.

```
def sqrt_bisect(x, error=0.00001):
    low = 0
    high = x
    m = (low + high)/2
    while (abs(m**2 - x) > error):
        if (m**2 < x):
            low = m
        else:
            high = m
    m = (low + high)/2
    return m
```

Newton’s Method

Let $f(x)$ be a well-defined function (think continuous) with root $f$. This means $f(r) = 0$. Newton’s method finds successive approximations of the root by using a linear approximation. Here’s the idea. Suppose that $x_0$ is an estimate of $r$. This means that $r = x_0 + \epsilon$ where $\epsilon$ is some small error.

$$0 = f(r) = f(x_0 + \epsilon) \approx f(x_0) + \epsilon f'(x_0)$$

by the first order terms of the Taylor expansion. The equation $y = f(x_0) + \epsilon f'(x_0)$ is also the tangent line to $f(x)$ at $(x_0, f(x_0))$. Setting this equal to 0 and solving for $\epsilon$ gives

$$\epsilon = -\frac{f(x_0)}{f'(x_0)}.$$ 

But $r = x_0 - \epsilon$ so

$$r \approx x_0 - \frac{f(x_0)}{f'(x_0)}.$$ 

This estimate of $r$ becomes our new estimate $x_1$ and we repeat the process resulting in a (hopefully) better and better approximation so that at iteration $n+1$ we have

$$x_{n+1} \approx x_n - \frac{f(x_n)}{f'(x_n)}.$$ 

This method works perfectly well for complex numbers too.

Square Root

If we want to find the square root of $y$ then using Newton’s method to solve $f(x) = x^2 - y$ leads to an appropriate solution.