Chapter 3 Notes

Exhaustive Enumeration

The algorithmic technique used in the a “brute force” checking program is a variant of guess and check called exhaustive enumeration. We enumerate all possibilities until we get to the right answer or exhaust the space of possibilities.

While and For loops

**While loops**: When writing while loops, we should think about an appropriate decrementing function (or incrementing) that can act as a counter.

The while loops can be used to iterate over a sequence of integers. Python provides the **for loop**, a loop structure that can be used to simplify containing this kind of iteration. The variable after “for” is generated using **the function range (start, stop, step)** and is implemented into the loop every time it changes. If the first argument is omitted it defaults to 0, and if the last argument is omitted is defaults to 1.

The for statement can be used in conjunction with **the in operator** to conveniently iterate over characters of a string.

Approximate Solutions and Bisection Search

An approximation is an answer that lies within some constant, call it epsilon, of the actual answer.

Exhaustive enumeration is a search technique that works only if the set of values being searched includes the answer. If you trying to find the square root of a number between 0 and 1, exhaustive enumeration doesn’t work. Bisection search is a method where you divide the search space in half at each step, which can greatly reduce the search time.

Floats

We, as humans, uses the decimal number systems in base 10 (A sequence of length n can represent 10^n numbers). Binary numbers (base 2) works similarly. A binary number is represented by a sequence of digits each of which is either 0 or 1. These digits are often called bits. A sequence of length n can represent 2^n numbers.

All modern computer systems represent numbers in binary. This is because that it is easy to build hardware switches, devices that can be in only one of two states, on or off. In most programming languages non-integer numbers are implemented using floating point.

To represent a float, we humans use significant digits and an exponent to represent. For example, the number 2.718 can be represented as (2718, -3) in base 10, meaning that 2.718 equals to 2718\*10^-3. The number of significant digits determines the precision with twhich numbers can be represented.

However, modern computers use binary, not decimal, representations. We represent the significant digits and exponents in binary rather than decimal and raise 2 raither than 10 to the exponent. Example, the number 0.625 can be represented as the pair (101, -11) because 5/8 is 0.101 in binary and -11 is the binary representation of -3. The pair (101, -11) stands for 5\*2^-3 which result in 0.625. Explanation: 5/8 = ½ + 1/8 which equals to 1\*1/2 + 0\* 1/4 \* 1\*1/8 which results in 0.101.

Because of this, when a number like 0.1 is expressed in terms of significant digits and exponents, it will take an infinite number of digits to satisfy the exact value of 0.1. This is because there do not exist integers “sig” and “exp” such that sig\*2^-exp equals to 0.1. Therefore we use the expression **round(x, numDigits)** to return a floating point number equivalent to rounding the value of x to numDigits decimals digits following the decimal point.

It is almost always more appropriate to ask whether two floating point values are close enough to each other, not whether they are identical. Writing abs(x – y) < 0.0001 is always better than writing x == y.

Newton-Raphson

Newton’s Method is a method that is used to find the root of many functions. It can be applied to polynomials. A polynomial with one variable is either 0 or sum of a finite number of nonzero terms, consists of a constant (coefficient of the term), multiply by the variable, raised to a non-negative integer exponent. The highest exponent in a polynomial is the degree of that polynomial.

Newton proved a theorem that implies that if a value, call it x, is an approximation to root of a polynomial, the guess – p(guess)/p’(guess) (where p’ is the 1st derivate of p()), is a better approximation. We can apply Newton Raphson method to find square roots.