CMPT 383: Vitamin #7

Anders Miltner
miltner@cs.sfu.ca

Due Nov 2

Introduction

This Vitamin is to help you practice basic coding in Rust. The test suite is provided in `src/lib.rs`. You should fill out the function definitions in `src/functions.rs`.

This submission will be autograded. There are some portions of the assignment that are ungraded, and some that will be graded. We provide a (partial) test suite for partial validation. You can run these tests by opening a terminal in the v7 directory, and running `cargo test`.

We have omitted all imports. If you import additional functions, you may get a zero on the assignment.

1 Collatz Conjecture

The Collatz Conjecture is an unsolved conjecture about *hailstone sequences*. In this exercise, you will write functions that generate hailstone sequences. A hailstone sequence is generated by basic operations. If the number n is in a hailstone sequence, then the number after n should be $n/2$ if n is even, and it should be $3 * n + 1$ if n is odd.

First, you will write a function, `next_hailstone` that calculates the next number in a hailstone sequence. So, `next_hailstone(17) = 52` and `next_hailstone(18) = 9`. A useful operation will be the "mod" operation. In Rust, to calculate $n \text{mod} k$, you write `n % k`.

The Collatz Conjecture is that every hailstone sequence eventually ends with 1. You will now build hailstone sequences until they hit 1.

Given a number `n`, `hailstone_sequence(n)` will calculate the hailstone sequence starting from `n`. For example, `hailstone_sequence(5) = vec!([5,16,8,4,2,1])`.

Recall some of the Vector operations are creating an empty Vector with `Vec::new()`, adding elements to a Vector `v` with `v.push` (remember to declare `v` as mutable with `let mut`), and creating a vector elements equal to a list with `vec![x1,...,xn]`. Also, it may be useful to use a `while` loop, which has nearly the same syntax as languages like C and C++.

2 Finding Indices Of an Element

In this part we will find all indices that an element appears at in a Vector. First we will try to find the first index that the element appears at with `find_elt`. Note that `find_elt` is generic over `T`, and that `T` must have the trait `Eq`. This means we can call `t1 == t2` if `t1` and `t2` have type `T`.

Some useful functions may be the length function over Vectors (`v.len()`) when `v` is a Vector. Also, one can range over numbers (for example, from `0` to `k` exclusive) with the syntax `1..k`. It may be useful to use a for loop, though it is also possible with a while loop.

Also of note is the return type, `Option<usize>`. Option is essentially the Maybe monad in Haskell, but in Rust. `Option<T>` has two constructors, `None` and `Some(t)` where `t has type T`. The `usize` is just an uninterpreted integer. If you are on a 32 bit architecture, `usize=u32`. If you are on a 64 bit architecture, `usize=u64`.

Next, it is time to not only compute the first index that an element appears at, but rather all indices. This is done with `all_indices`.