CS2030S Recitation

Week 10: Problem Set 7

brian

2026-03-26

National University of Singapore

Ammendment

- If a variable is assigned to a field of a local class, that variable will also be captured
  - could be a factory method that calls the constructor of `B` so still need the value of `x`

```
1  interface I {}
2  class A {
3    int z = 0;
4    I f() {
5      int x = 2;
6      class B implements I {
7        int y = x; // causes x to be captured
8        int a = z; // z still not captured
9      }
10     return new B();
11   }
12 }
```

- But a field of `A` would not be captured by `B`

Recap: Infinite Lists

- What is `head`?
  - A `Producer` that produces the value of that element of the infinite list
  - Think of it as the instructions to create the current value
- What is `tail`?
  - A `Producer` that produces the rest of the infinite list
  - Think of it as the instructions to create the next `InfiniteList`

Recap: InfiniteList APIs

- `iterate`: `init::T` is your initial value, `next::T → T` transforms a value to the next one
- `head`: gets the value of the first element
- `tail`: gets the rest of the `InfiniteList`
- `get`: get the value of the element that is n elements away from our head (offset by n)

Fibonacci Sequence

- First described by Indian mathematician Pingala
- Popularised by Fibonacci
- He realised that rabbits are loving and thus breed a lot
- He wanted to model the size of the rabbit population

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n-1) + F(n-2)$$

Q1(a)

Write a method `fib` that creates an infinite list of Fibonacci numbers. The list should be of type `InfiniteList<BigInteger>`.

```
1 // a is the first BigInteger in the sequence
2 // b is the second BigInteger in the sequence
3 InfiniteList<BigInteger> fib(BigInteger a, BigInteger b) {
4   :
5 }
```

Q1(a): Idea

- a and b are your current number and your next number.
- If `head` is called, return a (your current number)

Q1(a): Idea

- a and b are your current number and your next number.
- If `head` is called, return a (your current number)
- `Fib` is an encoding of 2 Fibonacci numbers at a particular point
- What should `tail` do then?
  - Hint: tail is the instructions for the rest of the list

Q1(a): Idea

- a and b are your current number and your next number.
- If `head` is called, return a (your current number)
- `Fib` is an encoding of 2 Fibonacci numbers at a particular point
- What should `tail` do then?
  - Hint: tail is the instructions for the rest of the list
- `tail` should create an `Fib` encoding where b is the current number, and $a + b$ is the next number

Q1(a): Reminder for brian

- Remind brian to pretend to be able to code live
- Answer on the next slide

Q1(a): Solution

```
1 InfiniteList<BigInteger> fib(BigInteger a, BigInteger b) {
2   return new InfiniteList<>(
3       () → a,
4       () → fib(b, a.add(b))
5   );
6 }
```

Q1(b)

Implement the method `InfiniteList::zipWith`, to combine two lists, element-wise, with a curried lambda expression.

For example,

$$l_1 = [a_1, a_2, a_3]$$

$$l_2 = [b_1, b_2, b_3]$$

$$l_1.\text{zipWith}(l_2) = [f(a_1)(b_1), f(a_2)(b_2), f(a_3)(b_3)]$$

```
1 public <S, R> InfiniteList<R> zipWith(
2   InfiniteList<? extends S> list,
      Transformer<? super T, ? extends Transformer<? super S, ? extends R>>
3     mapper) {
4       :
5   }
6 }
```

Q1(b): Idea

- We have l_1 as `this`, l_2 as `list`, f as `mapper`
- What's the instructions to make the current element?

Q1(b): Idea

- We have l_1 as `this`, l_2 as `list`, f as `mapper`
- What's the instructions to make the current element?
  - take current element of `this` and current element of `list` and combine with mapper.

Q1(b): Idea

- We have l_1 as `this`, l_2 as `list`, f as `mapper`
- What's the instructions to make the current element?
  - take current element of `this` and current element of `list` and combine with mapper.
- What's the instructions to make the rest of the list?

Q1(b): Idea

- We have l_1 as `this`, l_2 as `list`, f as `mapper`
- What's the instructions to make the current element?
  - take current element of `this` and current element of `list` and combine with mapper.
- What's the instructions to make the rest of the list?
  - Recurse on the rest of `this` and `list`.

Q1(b): Reminder for brian

- Remind brian to code live
- Answer on the next slide

Q1(b): Solution

```
1 public <S, R> InfiniteList<R> zipWith(
2   InfiniteList<? extends S> list,
      Transformer<? super T, ? extends Transformer<? super S, ? extends R>>
3
      mapper) {
4       return new InfiniteList<>(
5           () → mapper.transform(this.head()).transform(list.head()),
6           () → this.tail().zipWith(list.tail(), mapper)
7       );
8   }
9 }
```

Q2

- Write `fib` again, such that it returns the first n Fibonacci numbers as a `Stream<BigInteger>`.
- Can use `iterate`, `map`, and `limit` from `Stream` and the `Pair` class

Q2

- Write `fib` again, such that it returns the first n Fibonacci numbers as a `Stream<BigInteger>`.
- Can use `iterate`, `map`, and `limit` from `Stream` and the `Pair` class
- Previously, `fib` itself represented 2 Fibonacci numbers, now we can use `Pair` to do that

Q2: Idea

- Iterate to create pairs of Fibonacci numbers
  - For every pair, first is the “current” number second is the “next” number

Q2: Idea

- Iterate to create pairs of Fibonacci numbers
  - For every pair, first is the “current” number second is the “next” number
- Limit at n

Q2: Idea

- Iterate to create pairs of Fibonacci numbers
  - For every pair, first is the "current" number second is the "next" number
- Limit at n
- map to get the first of each pair

Q2: Reminder for brian

- Remind brian to code live
- Answer on the next slide

Q2: Solution

```
1 Stream<BigInteger> fib(int n) {
2   return Stream.iterate(new Pair<>(BigInteger.ONE, BigInteger.ONE),
3       pair → new Pair<>(pair.second, pair.first.add(pair.second)))
4       .limit(n).map(pair → pair.first);
5 }
```

Q3

- Write `product` that takes in 2 lists `list1` and `list2` and produces a `Stream` containing elements combining elements from `list1` with every element from `list2`

$$l_1 = [a, b]$$

$$l_2 = [1, 2]$$

$$\text{product}(l_1, l_2) = [f(a, 1), f(a, 2), f(b, 1), f(b, 2)]$$

```
1 <T, U, R> Stream<R> product(List<? extends T> list1, List<? extends U> list2,
2  BiFunction<? super T, ? super U, ? extends R> func) {
3     :
4 }
```

Q3: Idea

- We want a `Stream` so make `list1` and `list2` into streams (call them `stream1` and `stream2`)

Q3: Idea

- We want a `Stream` so make `list1` and `list2` into streams (call them `stream1` and `stream2`)
- For every element `e1` in `stream1`, go through every element `e2` in `stream2` and apply f on `e1` and `e2`

Q3: Reminder for brian

- Remind brian to code live
- Answer on the next slide

Q3: Naive Solution

```
1 <T, U, R> Stream<R> product(List<? extends T> list1, List<? extends U> list2,
2  BiFunction<? super T, ? super U, ? extends R> func) {
3     return list1.stream.map(e1 →
4         list2.stream().map(e2 →
5             func.apply(e1, e2))
6     );
7 }
```

Q3: Naive Solution

```
1 <T, U, R> Stream<R> product(List<? extends T> list1, List<? extends U> list2,
2  BiFunction<? super T, ? super U, ? extends R> func) {
3     return list1.stream.map(e1 →
4         list2.stream().map(e2 →
5             func.apply(e1, e2))
6     );
7 }
```

- But notice that for each element in `stream1` we end up producing a new stream
  - This means we have a stream of streams

Q3: Naive Solution

```
1 <T, U, R> Stream<R> product(List<? extends T> list1, List<? extends U> list2,
2  BiFunction<? super T, ? super U, ? extends R> func) {
3     return list1.stream.map(e1 →
4         list2.stream().map(e2 →
5             func.apply(e1, e2))
6     );
7 }
```

- But notice that for each element in `stream1` we end up producing a new stream
  - This means we have a stream of streams
- Use `flatMap` to flatten from stream of streams to stream

Q3: Actual Solution

```
1 <T, U, R> Stream<R> product(List<? extends T> list1, List<? extends U> list2,
2  BiFunction<? super T, ? super U, ? extends R> func) {
3     return list1.stream.flatMap(e1 →
4         list2.stream().map(e2 →
5             func.apply(e1, e2))
6     );
7 }
```

Q4: Omega numbers

Definition 1 (Prime number). $n \in \mathbb{N}$ is a prime number if $n > 1$, and is not a product of 2 smaller natural numbers

Q4: Omega numbers

Definition 3 (Prime number). $n \in \mathbb{N}$ is a prime number if $n > 1$, and is not a product of 2 smaller natural numbers

Definition 4 (Prime factor). d is a prime factor of n if d is a prime number and divides n

Q4: Omega numbers

Definition 1 (Omega number). The omega number of n is the number of distinct prime factors of n

Q4: Omega numbers

Definition 3 (Omega number). The omega number of n is the number of distinct prime factors of n

Theorem 4 (Unique prime factorization). *Every natural number n is a unique product of prime numbers*

Q4: Omega numbers

Definition 5 (Omega number). The omega number of n is the number of distinct prime factors of n

Theorem 6 (Unique prime factorization). *Every natural number n is a unique product of prime numbers*

Proof. Ask a toddler on the street □

Q4: Omega numbers

Definition 7 (Omega number). The omega number of n is the number of distinct prime factors of n

Theorem 8 (Unique prime factorization). *Every natural number n is a unique product of prime numbers*

Proof. Ask a toddler on the street □

just kidding just shows that Omega numbers are unique

Q4

- Write a method `omega` that returns the first n omega numbers as a `Stream<Integer>`

```
1 Stream<Long> omega(int n) {
2   :
3 }
```

Q4: Idea

- Create a stream of numbers $(1, 2, 3, \ldots, n)$

Q4: Idea

- Create a stream of numbers $(1, 2, 3, \ldots, n)$
- For the i-th number,

Q4: Idea

- Create a stream of numbers $(1, 2, 3, \ldots, n)$
- For the i-th number,
  - Create a new stream of numbers $(1, 2, \ldots, i)$

Q4: Idea

- Create a stream of numbers $(1, 2, 3, \ldots, n)$
- For the i-th number,
  - Create a new stream of numbers $(1, 2, \ldots, i)$
  - Filter and keep prime factors of i

Q4: Idea

- Create a stream of numbers $(1, 2, 3, \ldots, n)$
- For the i-th number,
  - Create a new stream of numbers $(1, 2, \ldots, i)$
  - Filter and keep prime factors of i
  - count the elements
- Can write additional methods (to check if `isPrime`)
- JavaDocs

Q4: Reminder for brian

- Remind brian to code live
- Answer on the next slide

Q4: Solution

```
1  static boolean isPrime(int n) {
2    return Stream.iterate(2, i → i + 1)
3      .limit(n - 2)
4      .noneMatch(x → n % x == 0);
5  }
6  static Stream<Long> omega(int n) {
7    return Stream
8        .iterate(1, i → i + 1)
9        .limit(n)
10       .map(ele → Stream
11           .iterate(2, x → x + 1)
12           .limit(ele)
13           .filter(d → (ele % d == 0) && isPrime(d))
14           .count());
15 }
```

The End

bye!