CS 61A
DISCUSSION TEN

Streams, Iterators, and Binary Trees

April 13, 2017

TODAY’S AGENDA

Ideally, you will learn (or reinforce your knowledge of):

- how to define streams
- how to define iterators
- how to process binary trees

Stretch goals:

- acquire the sum of all human knowledge

ANNOUNCEMENTS

- Scheme checkpoints are indeed graded on correctness.
- Hidden tests are only available when you submit.
- Project party on Wednesday 4/19.
- Homework is due on Tuesday.
- Special topics for the final discussion will actually just be bagels; I don’t want to try to fit an AI overview into 15-20 minutes. But that’s fine – more time to spend on the holey-est of bread products!

ATTENDANCE

The URL suffix is on the board

- append it to `tinyurl.com/` :)

(if you’re not in class, just email me)

STREAMS

STREAMS

Streams are linked lists that are evaluated lazily:

- The rest won’t be computed until we say `<stream>.rest`
- After we ask for it, the result will be **remembered** (we won’t have to call the `compute_rest` function again)

Python stream interface:

- `first` gives us the first element of the stream
- `rest` gives us the rest of the stream (which should also be a stream)
  - if the rest has never been computed, call `_compute_rest`
  - if the rest was ever computed, return whatever was saved in `_rest`
- `Stream.empty` is the empty stream

STREAMS

- Create a stream by passing in a value and a `compute_rest` function
  - This function should take no arguments
  - This function should return a `Stream` (or `Stream.empty`)

- We pass in a **function** so that we can have lazy evaluation. Without this detail, our streams would just be linked lists (as the "rest" would be evaluated at creation time)

- Once the stream is created, it's pretty much just used like a linked list
  - So most of our stream problems will revolve around creation

```python
class Stream:
    class empty:
        def __repr__(self):
            return 'Stream.empty'

    empty = empty()

    def __init__(self, first, compute_rest=empty):
        self.first = first
        if compute_rest is Stream.empty or isinstance(compute_rest, Stream):
            self._rest, self._compute_rest = compute_rest, None
        else:
            assert callable(compute_rest), 'compute_rest must be callable'
            self._compute_rest = compute_rest

    @property
    def rest(self):
        if self._compute_rest is not None:
            self._rest = self._compute_rest()
            self._compute_rest = None
        return self._rest

    def __repr__(self):
        return 'Stream({0}, <...>)'.format(repr(self.first))
```

STREAMS

```
def make_integer_stream(first=1):
    def compute_rest():
        return make_integer_stream(first + 1)
    return Stream(first, compute_rest)
```

It’s nice to have stream creators like `make_integer_stream`, because we can wrap them in no-argument functions and pass them as the `compute_rest` argument to the `Stream` constructor.

STREAMS

What is the advantage of using a stream over a linked list?

Elements won’t be evaluated unnecessarily if they are never used… meaning efficient space usage! Also, streams allow for the representation of infinite-length sequences.

On streams versus iterators:
Every time you call `next` on an iterator, it changes. Streams don’t.
Otherwise there are many similarities; iterators provide lazy evaluation as well.

ITERATORS

ITERATORS

According to the Python specification for iterators:

```
next(iterator) → value, or a StopIteration error
iter(iterator) → the iterator itself
```

BINARY SEARCH TREES

BINARY SEARCH TREES

Binary search trees are binary trees,

```
class BinTree:
    empty = ()
    def __init__(self, label, left=empty, right=empty):
        self.label = label
        self.left = left
        self.right = right
```

except everything in `self.left` must be less than or equal to `self.label`
and everything in `self.right` must be greater than or equal to `self.label`