CS 61A Fall 2023

Final Review

Discussion 12: April 24, 2024

Reminder: Use Discord for voice chat with the course staff. Write to `@discuss` in the `#discuss-queue` channel on Discord at any time, and a member of the course staff will join your group's voice channel.

Pick someone in your group to join Discord. It's fine if multiple people join, but one is enough.

Now switch to Pensieve:

- **Everyone**: Go to discuss.pensieve.co and log in with your @berkeley.edu email, then enter your group number. (Your group number is the number of your Discord channel.)

Once you're on Pensieve, you don't need to return to this page; Pensieve has all the same content (but more features). If for some reason Penseive doesn't work, return to this page and continue with the discussion.

Post in the `#help` channel on Discord if you have trouble.

Pro tip: Any of you can type a question into your group's Discord channel's text chat with the `@discuss` tag, and a member of the course staff will respond.

Getting Started

If you have only 1 or 2 people in your group, you can join the other group in the room with you.

Ice breaker: Everybody say your name and the non-CS/EECS course that you're most excited about taking next semester.

Lists

The two most common mutation operations for lists are item assignment and the `append` method.

```
>>> s = [1, 3, 4]
>>> t = s  # A second name for the same list
>>> t[0] = 2  # this changes the first element of the list to 2, affecting both s and t
>>> s
[2, 3, 4]
>>> s.append(5)  # this adds 5 to the end of the list, affecting both s and t
>>> t
[2, 3, 4, 5]
```

There are many other list mutation methods:

- `append(elem)`: Add `elem` to the end of the list. Return `None`.
- `extend(s)`: Add all elements of iterable `s` to the end of the list. Return `None`.
- `insert(i, elem)`: Insert `elem` at index `i`. If `i` is greater than or equal to the length of the list, then `elem` is inserted at the end. This does not replace any existing elements, but only adds the new element `elem`. Return `None`.
- `remove(elem)`: Remove the first occurrence of `elem` in list. Return `None`. Errors if `elem` is not in the list.
- `pop(i)`: Remove and return the element at index `i`.
- `pop()`: Remove and return the last element.

Q1: Word Rope

Definition: A *rope* in Python is a list containing only one-letter strings except for the last element, which may either be a one-letter string or a rope.

Implement `word_rope`, a Python function that takes a non-empty string `s` containing only letters and spaces that does not start or end with a space. It returns a *rope* containing the letters of `s` in which each word is in a separate list.

Important: You may not use slicing or the `split`, `find`, or `index` methods of a string. Solve the problem using list operations.

Reminder: `s[-1]` evaluates to the last element of a sequence `s`.

```
def word_rope(s):
    """Return a rope of the words in string s.

    >>> word_rope('the last week')
    ['t', 'h', 'e', ['l', 'a', 's', 't', ['w', 'e', 'e', 'k']]]
    """
    assert s and s[0] != ' ' and s[-1] != ' '
    result = []
    word = result
    for x in s:
        if x == ' ':
            word.append([])
            word = word[-1]
        else:
            word.append(x)
    return result
```

In this implementation, `result` is a rope and `word` is a list within that rope which is still being constructed. When `x` is a space, add an empty list to the end of `word` and assign `word` to this empty list. Otherwise, add `x` to the end of `word`.

Linked Lists

A linked list is a `Link` object or `Link.empty`.

You can mutate a `Link` object `s` in two ways: - Change the first element with `s.first = ...` - Change the rest of the elements with `s.rest = ...`

You can make a new `Link` object by calling `Link`: - `Link(4)` makes a linked list of length 1 containing 4. - `Link(4, s)` makes a linked list that starts with 4 followed by the elements of linked list `s`.

```
class Link:
    """A linked list is either a Link object or Link.empty

    >>> s = Link(3, Link(4, Link(5)))
    >>> s.rest
    Link(4, Link(5))
    >>> s.rest.rest.rest is Link.empty
    True
    >>> s.rest.first * 2
    8
    >>> print(s)
    <3 4 5>
    """
    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest

    def __repr__(self):
        if self.rest:
            rest_repr = ', ' + repr(self.rest)
        else:
            rest_repr = ''
        return 'Link(' + repr(self.first) + rest_repr + ')'

    def __str__(self):
        string = '<'
        while self.rest is not Link.empty:
            string += str(self.first) + ' '
            self = self.rest
        return string + str(self.first) + '>'
```

Q2: Linear Sublists

Definition: A *sublist* of linked list `s` is a linked list of some of the elements of `s` in order. For example, `<3 6 2 5 1 7>` has sublists `<3 2 1>` and `<6 2 7>` but not `<5 6 7>`.

Definition: A *linear sublist* of a linked list of numbers `s` is a sublist in which the difference between adjacent numbers is always the same. For example `<2 4 6 8>` is a linear sublist of `<1 2 3 4 6 9 1 8 5>` because the difference between each pair of adjacent elements is 2.

Implement `linear` which takes a linked list of numbers `s` (either a `Link` instance or `Link.empty`). It returns the longest linear sublist of `s`. If two linear sublists are tied for the longest, return either one.

```
def linear(s):
    """Return the longest linear sublist of a linked list s.

    >>> s = Link(9, Link(4, Link(6, Link(7, Link(8, Link(10))))))
    >>> linear(s)
    Link(4, Link(6, Link(8, Link(10))))
    >>> linear(Link(4, Link(5, s)))
    Link(4, Link(5, Link(6, Link(7, Link(8)))))
    >>> linear(Link(4, Link(5, Link(4, Link(7, Link(3, Link(2, Link(8))))))))
    Link(5, Link(4, Link(3, Link(2))))
    """
    def complete(first, rest):
        "The longest linear sublist of Link(first, rest) with difference d."
        if rest is Link.empty:
            return Link(first, rest)
        elif rest.first - first == d:
            return Link(first, complete(rest.first, rest.rest))
        else:
            return complete(first, rest.rest)
    if s is Link.empty:
        return s
    longest = Link(s.first) # The longest linear sublist found so far
    while s is not Link.empty:
        t = s.rest
        while t is not Link.empty:
            d = t.first - s.first
            candidate = Link(s.first, complete(t.first, t.rest))
            if length(candidate) > length(longest):
                longest = candidate
            t = t.rest
        s = s.rest
    return longest

def length(s):
    if s is Link.empty:
        return 0
    else:
        return 1 + length(s.rest)
```

There are three cases: - If `rest` is empty, return a one-element list containing just `first`. - If `rest.first` is in the linear sublist that starts with `first`, then build a list that contains `first`, and `rest.first`. - Otherwise, `complete(first, rest.rest)`.

This while loop is creating a `candidate` linear sublist for every two possible starting values: `s.first` and `t.first`. The rest of the linear sublist must be in `t.rest`.

Scheme

Q3: Increasing Rope

Definition: A *rope* in Scheme is a non-empty list containing only numbers except for the last element, which may either be a number or a rope.

Implement `up`, a Scheme procedure that takes a positive integer `n`. It returns a rope containing the digits of `n` that is the shortest rope in which each pair of adjacent numbers in the same list are in increasing order.

Reminder: the `quotient` procedure performs floor division, like `//` in Python. The `remainder` procedure is like `%` in Python.

```
(define (up n)
  (define (helper n result)
    (if (zero? n) result
      (helper
        (quotient n 10)
        (let ((first (remainder n 10)))
          (if (< first (car result))
              (cons first result)
              (list first result))
          ))))
  (helper
    (quotient n 10)
    (list (remainder n 10))
    ))

(expect (up 314152667899) '(3 (1 4 (1 5 (2 6 (6 7 8 9 (9)))))))
```

Compare `first` to `(car result)` to decide whether to `cons` the value `first` onto the `result` or whether to form a new list that contains `first` and `result` as elements.

To correctly call `helper` from within `up`, build a rope that only contains the last digit of `n`: `(remainder n 10)`.

SQL

A `SELECT` statement describes an output table based on input rows. To write one: 1. Describe the **input rows** using `FROM` and `WHERE` clauses. 2. **Group** those rows and determine which groups should appear as output rows using `GROUP BY` and `HAVING` clauses. 3. Format and order the **output rows** and columns using `SELECT` and `ORDER BY` clauses.

`SELECT` *(Step 3)* `FROM` *(Step 1)* `WHERE` *(Step 1)* `GROUP BY` *(Step 2)* `HAVING` *(Step 2)* `ORDER BY` *(Step 3)*;

Step 1 may involve joining tables (using commas) to form input rows that consist of two or more rows from existing tables.

The `WHERE`, `GROUP BY`, `HAVING`, and `ORDER BY` clauses are optional.

Q4: A Secret Message

A substitution cipher replaces each word with another word in a table in order to encrypt a message. To decode an encrypted message, replace each word `x` with its corresponding `y` in a code table.

Write a select statement to decode the `original` message *It's The End* using the `code` table.

```
CREATE TABLE original AS
  SELECT 1 AS n, "It's" AS word UNION
  SELECT 2     , "The"      UNION
  SELECT 3     , "End";

CREATE TABLE code AS
  SELECT "Up" AS x, "Down" AS y UNION
  SELECT "Now"    , "Home" UNION
  SELECT "It's"   , "What" UNION
  SELECT "See"    , "Do" UNION
  SELECT "Can"    , "See" UNION
  SELECT "End"    , "Now" UNION
  SELECT "What"   , "You" UNION
  SELECT "The"    , "Happens" UNION
  SELECT "Love"   , "Scheme" UNION
  SELECT "Not"    , "Mess" UNION
  SELECT "Happens", "Go";

SELECT y FROM original, code WHERE word=x ORDER BY n;
```

Join the `original` and `code` tables and make sure that the joined roles have the same `word` and `x`.

What happens now? Write another select statement to decode this encrypted message using the same `code` table.

```
CREATE TABLE original AS
  SELECT 1 AS n, "It's" AS word UNION
  SELECT 2     , "The"      UNION
  SELECT 3     , "End";

CREATE TABLE code AS
  SELECT "Up" AS x, "Down" AS y UNION
  SELECT "Now"    , "Home" UNION
  SELECT "It's"   , "What" UNION
  SELECT "See"    , "Do" UNION
  SELECT "Can"    , "See" UNION
  SELECT "End"    , "Now" UNION
  SELECT "What"   , "You" UNION
  SELECT "The"    , "Happens" UNION
  SELECT "Love"   , "Scheme" UNION
  SELECT "Not"    , "Mess" UNION
  SELECT "Happens", "Go";

SELECT b.y
  FROM original, code AS a, code AS b
  WHERE word=a.x AND a.y=b.x
  ORDER BY n;
```

Join `original` with `code AS a` and `code AS b` to create six-column rows like: `2|The|The|Happens|Happens|Go`, The *Go* at the end is part of the decoded message.

Scheduling time: This is the last discussion, but you could schedule a meeting with your group next week to study for the exam. Your regular discussion room and time should be available during RRR week if you want to use it.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).

Important: Please help put the furniture in the room back where you found it before you leave. Thanks!