The Linked List

CS 214, Fall 2019
A problem with vectors

What if we want to add 6 between 5 and 7? No can do! Elements 7, 8, 9, 10, and 11 are all in the way, and the vector is full. Need to create a new, bigger vector, and copy everything over…
A problem with vectors

What if we want to add 6 between 5 and 7?

2 3 4 5 7 8 9 10 11
What if we want to add 6 between 5 and 7?
No can do! Elements 7, 8, 9, 10, and 11 are all in the way, and the vector is full.
A problem with vectors

What if we want to add 6 between 5 and 7?
No can do! Elements 7, 8, 9, 10, and 11 are all in the way, and the vector is full.
Need to create a new, bigger vector, and copy everything over…
Books on a string

The Art of Computer Programming
VOLUME 1
Fundamental Algorithms
Third Edition
DONALD E. KNUTH

SURREAL NUMBERS

The Art of Computer Programming
VOLUME 2
Seminumerical Algorithms
Third Edition
DONALD E. KNUTH
Books on a string
Books on a string

The Art of Computer Programming
D. E. Knuth

SURREAL NUMBERS

The Art of Computer Programming
VOLUME 1
Fundamental Algorithms
Third Edition

DONALD E. KNUTH

CONCRETE MATHEMATICS
A FOUNDATION FOR COMPUTER SCIENCE

DONALD E. KNUTH

THE CLASSIC WORK
NEWLY UPDATED AND REVISED

THE CLASSIC WORK
NEWLY UPDATED AND REVISED

THE CLASSIC WORK
NEWLY UPDATED AND REVISED
Nodes and pointers

You saw cons in 111.

- car holds the *first* element, and
- cdr holds a pointer to the *rest* of the list.
Nodes and pointers

Inserting in the middle? No problem!
Just change the pointers
Nodes and pointers

Inserting in the middle? No problem!
Just change the pointers

You saw cons in 111.
• car holds the first element, and
• cdr holds a pointer to the rest of the list.

[Diagram showing pointers and nodes]

- car 2
- cdr
- car 4
- cdr
- car 6
- cdr
- car 8
- cdr
- car 5
- cdr
- car 3
- cdr
Nodes and pointers

Inserting in the middle? No problem!

Just change the pointers
Inserting in the middle? No problem! Just change the pointers
Nodes and pointers

Inserting in the middle? No problem!
Just change the pointers

2 → 7 → 4 → 5 → 6 → 8

Car holds the first element, and cdr holds a pointer to the rest of the list.
Inserting at the beginning
Inserting at the beginning
Inserting at the beginning

lst

car 2 cdr
car 3 cdr
car 4 cdr
car 5 cdr
car 6 cdr
car 7 cdr
car 8 cdr

6
Inserting at the beginning
Inserting at the beginning
Indirection
Indirection
Indirection
Now in DSSL2
Linked lists in DSSL2

# Link is one of:
# - node { data: Number, next: Link }
# - None

struct node:
    let data
    let next

class SLL:
    let head

    def __init__(self):
        self.head = None
Linked lists in DSSL2

# Link is one of:
# - node { data: Number, next: Link }
# - None

struct node:
  let data
  let next

class SLL:
  let head

  def __init__(self):
    self.head = None

  def push_front(self, data):
    self.head = node(data, self.head)
List operations in DSSL2

class SLL:
    ...

    def get_front(self):
        if node?(self.head):
            self.head.data
        else:
            error('SLL.get_front: empty list')
class SLL:

    ...

def get_front(self):
    if node?(self.head): self.head.data
    else: error('SLL.get_front: empty list')

def get_nth(self, n):
    let curr = self.head
    while n > 0:
        if curr is None:
            error('SLL.get_nth: too short')
            curr = curr.next
            n = n - 1
    curr.data
More DSSL2 list operations

A (re)factoring:

class SLL:
    ...

    def _find_nth_node(self, n):
        curr = self.head
        while n > 0:
            if curr is None: error('too short')
            curr = curr.next
            n = n - 1
        return curr

    def get_nth(self, n):
        return self._find_nth_node(n).data

    def set_nth(self, n, val):
        self._find_nth_node(n).data = val
What else might we want to do?
What else might we want to do?

- Insert or remove at the given position or the end.
- Split a list in two or splice two into one.
- Know how long the list is without counting.
Keeping the length

How can we make sure the len field is always right?
Keeping the length

How can we make sure the \texttt{len} field is always right?
Quick access to the tail

Which operations are simple now? Which are still more work?
Quick access to the tail

Which operations are simple now? Which are still more work?
Circular, doubly-linked with sentinel

- Sentinel
  - len: 6

- Nodes (1 to 6)
  - Data:
    - Node 1: data 1
    - Node 2: data 2
    - Node 3: data 3
    - Node 4: data 4
    - Node 5: data 5
    - Node 6: data 6
  - prev
  - next
Empty (circular, doubly-linked w/sentinel)
Let’s look at a singly-linked list class in DSSL2.
Next time: abstract data types