The Linked List

CS 214, Fall 2019
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?
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.
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

The Art of Computer Programming
D. E. Knuth

SURREAL NUMBERS
D. E. Knuth

The Classic Work
Newly Updated and Revised

CONCRETE MATHEMATICS
A Foundation for Computer Science
Graham, Knuth, Patashnik

The Classic Work
Newly Updated and Revised
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

CONCRETE MATHEMATICS
A FOUNDATION FOR COMPUTER SCIENCE
GRAHAM, KNUTH, PATASHNIK
SECOND EDITION
Nodes and pointers

You saw cons in 111.

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

```
2  4  6  8
```

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.

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

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

![Diagram showing nodes and pointers]

- car holds the first element,
- cdr holds a pointer to the rest of the list.

Inserting at the middle? No problem!
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.

Inserting in the middle?

\[
\begin{align*}
\text{car} & \quad \text{cdr} \\
2 & \quad - \\
4 & \quad 6 \\
7 & \quad 5 \\
5 & \quad 3 \\
6 & \quad 8 \\
\end{align*}
\]
Inserting at the beginning
Inserting at the beginning
Inserting at the beginning
Inserting at the beginning

```
(car 8 (cdr car 7 (cdr car 6 (cdr car 5 (cdr car 4 (cdr car 3 (cdr car 2 (cdr car 1 lst))))))))
```
Inserting at the beginning
Indirection
Indirection

```
Indirection
```

```
list

head

car 1

car 2

car 3

car 4

car 5

car 6

car 7

car 8

head
```
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

```python
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)
```

```python
```
List operations in DSSL2

class SLL:
    ...

def get_front(self):
    if node?(self.head): self.head.data
    else: error('SLL.get_front: empty list')
List operations in DSSL2

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):
        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
        curr

    def get_nth(self, n):
        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 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?
Doubly-linked
Circular, doubly-linked with sentinel

- Sentinel
  - len: 6
- Nodes:
  - data: 1, prev, next
  - data: 2, prev, next
  - data: 3, prev, next
  - data: 4, prev, next
  - data: 5, prev, next
  - data: 6, prev, next

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