The linked list

EECS 214, Fall 2018
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?
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

VOLUME 1
Fundamental Algorithms
Third Edition

DONALD E. KNUTH

SURREAL NUMBERS

DONALD E. KNUTH

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
Nodes and pointers
Nodes and pointers

```
<table>
<thead>
<tr>
<th>car</th>
<th>cdr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```

The diagram illustrates the structure of a linked list with nodes containing the values 2, 4, 5, 6, and 8.
Nodes and pointers
Nodes and pointers
Nodes and pointers
Inserting at the beginning
Inserting at the beginning
Inserting at the beginning
Inserting at the beginning
Inserting at the beginning
Indirection
Indirection
Now in DSSL2
Linked lists in DSSL2

# Link is one of:
# - node { data: Number, next: Link }
# - nil()

struct node:
    let data
    let next

struct nil: pass

class LL:
    let head

    def __init__(self):
        self.head = nil()
Linked lists in DSSL2

# Link is one of:
# - node { data: Number, next: Link }
# - nil()

struct node:
    let data
    let next

struct nil: pass

class LL:
    let head

    def __init__(self):
        self.head = nil()

    def push_front(self, data):
        self.head = node(data, self.head)
class LL:
    ...

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

    def get_nth(self, n):
        curr = self.head
        while n > 0:
            if nil?(curr):
                error('get_nth: list too short')
            n = n - 1
            curr = curr.next
        return curr.data
List operations in DSSL2

```python
class LL:
    ...

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

def get_nth(self, n):
    curr = self.head
    while n > 0:
        if nil?(curr):
            error('get_nth: list too short')
            n = n - 1
            curr = curr.next
        curr.data
```
class LL:

... 

def _find_nth_node(self, n):
    curr = self.head
    while n > 0:
        if nil?(curr): error('list too short')
        n = n - 1
        curr = curr.next
    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 \textit{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

1. **Sentinel**: len 6

2. **Nodes**:
   - Data: 6
   - Prev
   - Next
   - Data: 5
   - Prev
   - Next
   - Data: 4
   - Prev
   - Next
   - Data: 3
   - Prev
   - Next
   - Data: 2
   - Prev
   - Next
   - Data: 1
   - Prev
   - Next

3. **Diagram**: The nodes are connected in a circular fashion, with each node having a `prev` and `next` pointer, creating a doubly-linked list.
Empty (circular, doubly-linked w/sentinel)
Next time: asymptotic complexity