The linked list

EECS 214, Fall 2018
A problem with vectors
A problem with vectors

What if we want to add 6 between 5 and 7?
Books on a string
Books on a string

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

SURREAL NUMBERS
D. 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
Books on a string

The Art of Computer Programming
D. E. Knuth

Surreal Numbers
D. E. Knuth

Concrete Mathematics
A Foundation for Computer Science
Graham, Knuth, Patashnik

The Classic Work
Newly Updated and Revised

Volume 1
Fundamental Algorithms
Third Edition

Volume 2
Seminumerical Algorithms
Third Edition

Donald E. Knuth
Nodes and pointers

car 2
cdr

car 4
cdr

car 6
cdr

car 8
cdr

...
Nodes and pointers
Nodes and pointers
Nodes and pointers

```
car 2
  car 4
    car 6
      car 8
```

```
cdr 2
  cdr 4
    cdr 6
      cdr 8
```

```
cdr 4
  cdr 5
```

```
car 2
```
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

lst

head

car 2
cdr

car 3
cdr

car 4
cdr

car 5
cdr

car 6
cdr

car 7
cdr

car 8
cdr

head

lst
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()
# 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)
List operations in DSSL2

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 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
More DSSL2 list operations

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 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
2. len
3. data
4. prev
5. next
6. data
7. prev
8. next
9. data
10. prev
11. next
12. data
13. prev
14. next
15. data
16. prev
17. next

len: 6
Empty (circular, doubly-linked w/sentinel)
Next time: asymptotic complexity