CSc 120
Introduction to Computer Programming II

Adapted from slides by Dr. Saumya Debray

10: Linked Lists
Python lists: reprise

L

L.insert: O(n)

L[i] : O(1)

L.append: O(1)

concatenating two lists: O(n)

**Question:** Can we do insertion and concatenation in O(1) time?

(complexity of other operations may change).

⇒ "Linked list"
Python lists: reprise

• Key feature: $L[i]$ and $L[i+1]$ are adjacent in memory
  • This makes accessing $L[i]$ very efficient
    – $O(1)$

• Insertion and concatenation require moving $O(n)$ elements
  – $O(n)$
Linked lists

• To get $O(1)$ insertion and concatenation, we cannot afford to move $O(n)$ list elements

• We have to relax the requirement that $i^{th}$ element is adjacent to $(i + 1)^{st}$ element
  – any element can be anywhere in memory

• Each element has to tell us where to find the next element
linked lists
Linked lists

• Linked list:

A collection of ordered elements where each element has a value and a reference to the next element.

There is at least one variable that references the beginning of the list.
Linked lists

Each element of the list has a reference to the next list element.

This is how we draw a linked list.
Linked lists

With each element of the list, keep a reference to the next list element

"nodes"

each node in the list has a reference to the next node
Linked lists

With each element of the list, keep a reference to the next list element

Let's explore this idea using a file for a "node"
Linked lists

With each element of the list, keep a reference to the next list element

Let's explore this idea using a file for a "node" each file has two lines:
- the first line is a value
- the second line is a reference to the next "node" (file)
Linked lists

• Let's explore this idea using a file for a "node"
  – each file has two lines:
    • the first line is a value
    • the second line is a reference to the next "node" (file)

Sample file “node”: filename is 20.txt
  value: The
  next: 11.txt
EXERCISE

• Exploring linked lists using files as nodes

How would we add the word "total" to our linked lists of files so that the sentence reads:

*The expert in anything was once a total beginner.*
EXERCISE

• Exploring linked lists using files as nodes

How would we add the word "total" to our linked lists of files so that the sentence reads:

_The expert in anything was once a total beginner._

Create a new "node" (a new file)

- The first line is "total"
- The second line is 17.txt

What else do we have to do?

- modify the file node for the value "a" to change its reference
Linked lists

References are addresses in memory.
Here is the diagram with explicit addresses (simplified).
Insertion

Consider inserting a new node into the linked list
Specifically, add a new node between "bbb" and "ccc". What do we change?
Insertion

Specifically, add a new node between "bbb" and "ccc". What do we change?
Insertion

We want to add a new node between "bbb" and "ccc". What do we change?
Set the next references appropriately. What is the complexity of insertion?

*assuming we have a reference to the node of insertion*
Insertion

To insert an element (which can be a linked list) into a linked list: set next references appropriately

\[O(1)\]
Insertion

To insert an element into a linked list: set next references appropriately

O(1)
Concatenation

To concatenate two linked lists: set next reference of end of first list to refer to beginning of second list

* once we have a reference to the end of the first list
Concatenation

To concatenate two linked lists: set next reference of end of first list to refer to beginning of second list

* once we have a reference to the end of the first list
implementation
Nodes: Implementation

class Node:
    def __init__(self, value):
        self._value = value  # reference to the object at that node
        self._next = None    # reference to the next node in the list

Getters:
    def value(self):
        return self._value
    def next(self):
        return self._next

Setters:
    def set_value(self, value):
        self._value = value
    def set_next(self, next):
        self._next = next
Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes
A linked list is just (a reference to) a sequence of nodes

```python
class LinkedList:
    def __init__(self):
        self._head = None
```

Linked List

[Diagram of a linked list with nodes and the class definition shown]
Linked Lists: Implementation

A linked list is just (a reference to) a sequence of nodes

```python
class LinkedList:
    def __init__(self):
        self._head = None
```

head of the list  tail of the list
Linked Lists: Implementation

class LinkedList:
    def __init__(self):
        self._head = None

    def is_empty(self):
        return self._head == None

    def head(self):
        return self._head
addition
at the head of the list
Adding a node at the head

L

_head

'aa'

'bb'

new

'dd'

_next

+ 

L

'aa'

'bb'

new

'dd'

_next
Adding a node at the head

Sequence of operations for an add method:
1. new._next = L._head
2. L._head = new
Adding a node at the head

```python
class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new
```

# add a node new at the head of the linked list

$O(1)$
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
```

**infile.txt**

<table>
<thead>
<tr>
<th>aa</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb</td>
</tr>
<tr>
<td>cc</td>
</tr>
</tbody>
</table>

**my_list**

```
LinkedList

<table>
<thead>
<tr>
<th>_head</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>
```
Creating a linked list: Example

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None
    def add(self, new):
        new._next = self._head
        self._head = new

in_file = open("infile.txt")
my_list = LinkedList()
for line in in_file:
    this_node = Node(line)
    my_list.add(this_node)
```

```
<table>
<thead>
<tr>
<th>infile.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
</tr>
<tr>
<td>bb</td>
</tr>
<tr>
<td>cc</td>
</tr>
</tbody>
</table>
```

```
my_list

line

"aa"
```
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

... 

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
```

```
infile.txt
| aa |
| bb |
| cc |
```

```
my_list

<table>
<thead>
<tr>
<th>_head</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedList</td>
</tr>
</tbody>
</table>

```

```
this_node

| None |
| Node |
| "aa" |
```
Creating a linked list: Example

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None
    def add(self, new):
        new._next = self._head
        self._head = new
```

```python
infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
```

```
my_list
  |__head
  |__LinkedList
  |
    __init__
    __init__
    add

this_node
  |__Node
  |__None
```

```
infile.txt
  |__aa
  |__bb
  |__cc
```

```
line
  |__"bb"
```

```
"aa"
  |__Node
  |__None
```
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

...  

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

    ...

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

```python
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
```

```
my_list

LinkedList

 line

"cc"

this_node

Node

None

Node

"bb"

"aa"

infile.txt

aa

bb

cc
```
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None
    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)

my_list

infile.txt

<table>
<thead>
<tr>
<th>aa</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb</td>
</tr>
<tr>
<td>cc</td>
</tr>
</tbody>
</table>

Node

"aa"

"bb"

"cc"

Linked List

Linked List

this_node

Node

Node

Node

None

Node

Node

Node

None
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None
    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Creating a linked list: Example

class Node:
    def __init__(self, value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new

infile = open("infile.txt")
my_list = LinkedList()
for line in infile:
    this_node = Node(line)
    my_list.add(this_node)
Adding a node at the head

⚠️ Changing the order of assignments does not work:

```python
def add(self, new):
    new._next = self._head
    self._head = new
```

```python
def broken_add(self, new):
    self._head = new
    new._next = self._head
```

---

```python
'aa'
'bb'
'dd'
```

---

```python
'aa'
'bb'
'dd'
```
Review

• Draw a linked list, L, with nodes that have the following values (in this order):
  'A', 'E', 'I', 'O', 'U'

• What is a linked list?

• Do problem 1 in the Exercises.
Visiting all of the nodes

Suppose we want to do something to each node of a list:

```
L
  "aaa"
  "bbb"
  "ccc"
```

How do we loop through the nodes (elements)?

- with a built-in list, we would use a for loop
- can we do something similar?
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Use a variable to refer the first element (current)
Progress through the list until you hit the end
Visiting all of the nodes

Suppose we want to do something to each node of a list:

```
progress through the list until you hit the end:
current = self._head
while current != None:
    <do something with current._value>
current = current._next
```
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Progress through the list until you hit the end:

```python
current = self._head
while current != None:
    <do something with current._value>
    current = current._next
```
Visiting all of the nodes

Suppose we want to do something to each node of a list:

Progress through the list until you hit the end:

\[
\text{current} = \text{self._head}
\]

\[
\text{while current \neq None:}
\]

\[
\text{<do something with current._value>}
\]

\[
\text{current} = \text{current._next}
\]
Visiting all of the nodes

Suppose we want to do something to each node of a list:

```
progress through the list until you hit the end:
current = self._head
while current != None:
    <do something with current._value>
current = current._next
```

```python
L
  "aaa"
  "bbb"
  "ccc"
None
```

```text
progress through the list until you hit the end:
current = self._head
while current != None:
    <do something with current._value>
current = current._next
```
Example: print each element

class LinkedList:
    def __init__(self):
        self._head = None

    def print_elements(self):
        current = self._head
        while current != None:
            print(str(current._value))
            current = current._next
EXERCISE-2

• Consider a linked list whose value attributes consist of integers.

• Write a method `double(self)` that doubles the value attributes of all nodes in a linked list.
• Consider a linked list whose value attributes consist of integers.

• Write a method `first_even(self)` that returns the first node in the linked list whose value is even. If there are no even values, the method returns `None`. 
EXERCISE

• Consider a linked list whose value attributes consist of strings.

• Write a method `replace(self, s1, s2)` that replaces the value attributes of all nodes that equal `s1` with `s2`. 
adding to the end (tail) of the list
Adding a node to the tail

Suppose we want to add a node to the end of a list:
Adding a node to the tail

Suppose we want to add a node to the end of a list:

To add a node new at the end (i.e., tail) of a list L:

1. find the last element Y of L
2. Y._next = new
Adding a node to the tail

To add a node new at the end (i.e., tail) of a list L:

1. find the last element Y of L
2. Y._next = new
Adding a node to the tail

To add a node new at the end (i.e., tail) of a list L:

1. find the last element Y of L \text{ \textbf{O(n)}}
2. Y._next = new \text{ \textbf{O(1)}}
Adding a node to the tail

To add a node new at the end (i.e., tail) of a list L:
1. find the last element Y of L
2. Y._next = new

Questions:
- Can we use the same code pattern as for double() and replace()?
- Draw this using current as the reference.
- What should we do if the list is empty?
Adding a node to the tail

Suppose we want to add a node to the end of a list:

Progress through the list until you hit the end:

```python
current = self._head
while current != None:
    current = current._next
```
Adding a node to the tail

Suppose we want to add a node to the end of a list:

To add a node new at the end (i.e., tail) of a list L:

1. Traverse in a loop
2. Keep track of the previous node
Adding a node to the tail

Suppose we want to add a node to the end of a list:

To add a node new at the end (i.e., tail) of a list L:

1. Traverse in a loop

2. Keep track of the previous node in prev
Adding to the end

class LinkedList:

def add_to_end(self, new):
    if self._head == None:
        self._head = new

    else:
        current = self._head
        prev = None
        while current != None:
            prev = current
            current = current._next

        prev._next = new
finding the $n^{th}$ element
Finding the $n^{th}$ element

class LinkedList:

    # return the node at position n of the linked list
    def get_element(self, n):
        elt = self._head
        while elt != None and n > 0:
            elt = elt._next
            n -= 1
        return elt

$O(n)$
insertion
Inserting a node

Suppose we want to insert a node $X$ into a list here:

Then we have to adjust the next-node reference on the node $Y$ just before that position.
Inserting a node

Suppose we want to insert a node X into a list here:

Then we have to adjust the next-node reference on the node Y just before that position
Inserting a node

The order of operations is important:

1. X._next = Y._next
Inserting a node

The order of operations is important:

1. X._next = Y._next
2. Y._next = X
Inserting a node

Inserting a node X at position \( n \) in a list L:

1. find the node Y at position \( n - 1 \)
   - iterate \( n - 1 \) positions from the head of the list*

2. insert X after Y
   - adjust next-node references as in previous example

\[
Y = \text{L._head}\n\]
\[
\text{for } i \text{ in range}(n-1):\quad O(n)\n\]
\[
Y = Y._\text{next}\n\]
\[
X._\text{next} = Y._\text{next}\n\]
\[
Y._\text{next} = X\n\]

* do something sensible if the list has fewer than \( n-1 \) nodes

\( O(n) \)
Inserting a node

class LinkedList:

    # insert a node new at position n

def insert(self, new, n):
    if n == 0:
        self.add(new)
    else:
        prev = self.get_element(n-1)
        new.next = prev.next
        prev.next = new
deletion
Deleting a node

Suppose we want to delete this node:

```
  "aaa"
  "bbb"
  "ccc"
```

... X ...

Diagram showing the deletion of the node labeled "aaa".
Deleting a node

Suppose we want to delete this node:
Defining a node

Suppose we want to delete this node:

```
def delete(self, X):
    Y = self._head
    while Y != None:
        if Y == X:
            <delete node X>
            return
        Y = Y._next
```

- No, does not work
- We need a reference to the previous node
Deleting a node

Suppose we want to delete this node:

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aaa</td>
<td>bbb</td>
<td>ccc</td>
</tr>
</tbody>
</table>
```

1. find the node Y just before X (i.e., Y._next == X)  \[O(n)\]
2. Y._next = X._next  \[O(1)\]
3. X._next = None
Deleting a node

class LinkedList:

    # delete a node X
    def delete(self, X):
        if self._head == X:  # X is the head of the list
            self._head = X._next
        else:
            Y = self._head
            while Y._next != X:  # alternative method- look ahead
                Y = Y._next
            Y._next = X._next
        X.next = None
concatenation
Concatenating two linked lists

class LinkedList:
    # concatenate list2 at the end of the list
    def concat(self, list2):
        if self._head == None:  # list is empty
            self._head = list2._head
        else:
            tail = self._head
            while tail._next != None:  # O(n)
                tail = tail._next
            tail._next = list2._head  # O(1)
maintaining a tail reference
Maintaining a tail reference

A variation is to also maintain a reference to the tail of the list

```python
class LinkedList:
    def __init__(self):
        self._head = None
        self._tail = None
```
Tail references and concatenation

list1

list2
Tail references and concatenation

_ head
_ tail

“aa” “bb” “cc”

_ head
_ tail

“dd” “ee” “ff”
Tail references and concatenation

```
list1
  _head
  _tail
    "aa"
    "bb"
    "cc"
```

```
list2
  _head
  _tail
    "dd"
    "ee"
    "ff"
```
Tail references and concatenation

list1

_list1_head

_list1_tail

“aa” “bb” “cc”

list2

_list2_head

_list2_tail

“dd” “ee” “ff”
Maintaining a tail reference

• Concatenation and append become O(1):
  
def concat(self, list2):
      if self._head == None:
          self._head = list2._head
          self._tail = list2._tail
      else:
          self._tail._next = list2._head
          self._tail = list2._tail

• All linked list operations must now make sure that the tail reference is kept properly updated
## Linked lists: summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Without tail reference</th>
<th>With tail reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>add to front of list</td>
<td>O(1)</td>
<td></td>
</tr>
<tr>
<td>append to end of list</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>find nth element</td>
<td>O(n)</td>
<td></td>
</tr>
</tbody>
</table>
| insert                     | O(1) if prev. node is available  
                           | O(n) otherwise       |
| delete                     | O(1) if prev. node is available  
                           | O(n) otherwise       |
| concatenate                | O(n)                   | O(1)                |