CSc 120
Introduction to Computer Programming II

11: Linked Lists
Problem: merging exam scores

student exams

\[ \text{distribute} \]

\[ \ldots \]

section 1

\[ \text{grader} \]

\[ \text{grading} \]

section 1

\[ \text{scores} \]

section 2

\[ \text{grader} \]

\[ \text{grading} \]

section 2

\[ \text{scores} \]

\[ \ldots \]

section \( n \)

\[ \text{grader} \]

\[ \text{grading} \]

section \( n \)

\[ \text{scores} \]

class scores

\[ \text{merge} \]

want merge to be \( O(\text{no. of sections}) \)

\[ \textit{not} \]

\( O(\text{no. of exams}) \)
Python lists: reprise

L
L[i] : O(1)
L.insert: O(n)
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

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

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

Each node in the list has a reference to the next node
Insertion

To insert an element 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
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.
Linked Lists: Implementation

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

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

nodes
Linked Lists: Implementation

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

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

_\texttt{head} \quad 'aa' \quad 'bb'

+ 

new \quad 'dd'
Adding a node at the head

L

_new

'aa'

'bb'

new

'dd'

L

'aa'

'bb'

new

'dd'
Adding a node at the head

Sequence of operations:
1. `new._next = L._head`
2. `L._head = new`
Adding a node at the head

class LinkedList:

    # add a node new at the head
    # of the linked list
    def add(self, new):
        new._next = self._head
        self._head = new

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

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)
```

```plaintext
infile.txt
  aa
  bb
  cc

my_list
  _head
  None
  line
  "aa"
this_node
  None
```

27
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)
```

```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)
```
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("infiles.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

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

my_list = LinkedList()
infile = open(“infile.txt”)  
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

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

```
None

this_node

my_list

line

"cc"

"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)
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)
```

This code defines a `Node` class and a `LinkedList` class. The `LinkedList` class initializes an empty list and provides a method to add new nodes to the list. The code reads a file named `infile.txt`, creates an instance of `LinkedList`, and iterates through each line in the file, creating a `Node` for each line and adding it to the list.
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
```

---

[Diagram showing the difference between the correct and incorrect implementations]
appending to the tail of the list
Adding a node at the tail

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

1. find the last element $Y$ of $L$
2. $Y._\text{next} = X$
Adding a node at the tail

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

1. find the last element $Y$ of $L$
2. $Y._\text{next} = X$

$O(n)$

$O(1)$
Adding a node at the tail

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

1. find the last element Y of L
2. Y._next = X

Gotchas to watch out for:

• what if there is no last element?
  – how can we tell?
  – what should we do?

EXERCISE
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 n > 0:
            assert elt != None
            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:

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

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. $X._\text{next} = Y._\text{next}$
Inserting a node

The order of operations is important:

1. $X._{\text{next}} = Y._{\text{next}}$
2. $Y._{\text{next}} = X$

![Diagram showing the order of operations](image)
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

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

```
Y = L._head
for i in range(n-1):
    Y = Y._next
X._next = Y._next
Y._next = X
```

\( O(n) \)

\( O(1) \)
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" ...
```
Deleting a node

Suppose we want to delete this node:

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

... X ...

... X ...

... "aaa" "bbb" "ccc"
Deleting a node

Suppose we want to delete this node:

1. find the node Y just before X (i.e., Y._next == X)
2. Y._next = X._next
3. X._next = None

O(n)  
O(1)
Deleting a node

class LinkedList:
    # delete a node X
    def delete(self, X):
        if self._head == X:
            # X is the head
            self._head = X._next
        else:
            Y = self._head
            while Y._next != X:
                Y = Y._next
                Y._next = X._next
            X.next = None
concatenation
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:
                tail = tail._next
            tail.next = list2._head

O(n)  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

_list_head

“aa”

“bb”

“cc”

_list_tail

list2

_list_head

“dd”

“ee”

“ff”

_list_tail
Tail references and concatenation
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 $O(1)$:
  
  ```python
  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>
<tr>
<td>insert</td>
<td>O(1) if prev. node is available</td>
<td>O(n) otherwise</td>
</tr>
<tr>
<td>delete</td>
<td>O(1) if prev. node is available</td>
<td>O(n) otherwise</td>
</tr>
<tr>
<td>concatenate</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>