

Lecture 12, 15 September 2022

Linked lists

```
In [1]: class Node:
def __init__(self, v = None):
    self.value = v
    self.next = None
    return

def isempty(self):
    if self.value == None:
        return(True)
    else:
        return(False)

def append(self,v): # append, recursive
    if self.isempty():
        self.value = v
    elif self.next == None:
        self.next = Node(v)
    else:
        self.next.append(v)
    return

def appendi(self,v): # append, iterative
    if self.isempty():
        self.value = v
        return

    temp = self
    while temp.next != None:
        temp = temp.next

    temp.next = Node(v)
    return

def insert(self,v):
    if self.isempty():
        self.value = v
        return

    newnode = Node(v)

    # Exchange values in self and newnode
    (self.value, newnode.value) = (newnode.value, self.value)

    # Switch links
    (self.next, newnode.next) = (newnode, self.next)

    return

def delete(self,v): # delete, recursive
    if self.isempty():
        return

    if self.value == v:
        self.value = None
        if self.next != None:
            self.value = self.next.value
            self.next = self.next.next
        return
    else:
        if self.next != None:
            self.next.delete(v)
            if self.next.value == None:
                self.next = None

    return

def __str__(self):
    # Iteratively create a Python list from linked list
    # and convert that to a string
    selflist = []
    if self.isempty():
        return(str(selflist))

    temp = self
    selflist.append(temp.value)

    while temp.next != None:
        temp = temp.next
        selflist.append(temp.value)

    return(str(selflist))
```

```
In [2]: l = Node()
l.append(5)
print(l)
```

[5]

```
In [3]: l.append(7)
print(l)
```

```
[5, 7]
```

Rewrite `__str__` using `_tolist` that recursively constructs a Python list from the linked list

- Note, convention to prefix "hidden" function names with `_`

```
In [4]: class Node:
def __init__(self, v = None):
    self.value = v
    self.next = None
    return

def isempty(self):
    if self.value == None:
        return(True)
    else:
        return(False)

def append(self,v): # append, recursive
    if self.isempty():
        self.value = v
    elif self.next == None:
        self.next = Node(v)
    else:
        self.next.append(v)
    return

def appendi(self,v): # append, iterative
    if self.isempty():
        self.value = v
        return

    temp = self
    while temp.next != None:
        temp = temp.next

    temp.next = Node(v)
    return

def insert(self,v):
    if self.isempty():
        self.value = v
        return

    newnode = Node(v)

    # Exchange values in self and newnode
    (self.value, newnode.value) = (newnode.value, self.value)

    # Switch links
    (self.next, newnode.next) = (newnode, self.next)

    return

def delete(self,v): # delete, recursive
    if self.isempty():
        return

    if self.value == v:
        self.value = None
        if self.next != None:
            self.value = self.next.value
            self.next = self.next.next
        return
    else:
        if self.next != None:
            self.next.delete(v)
            if self.next.value == None:
                self.next = None
        return

def _tolist(self):
    # Recursively construct a Python list from linked list
    if self.isempty():
        return([])
    elif self.next == None:
        return([self.value])
    else:
        return([self.value] + self.next._tolist())

def __str__(self):
    # Convert Python list representation of linked list to string
    return(str(self._tolist()))
```

```
In [5]: l = Node()
l.append(5)
print(l)
```

```
[5]
```

```
In [6]: l.append(7)
print(l)
```

```
[5, 7]
```

Some performance measurement

```
In [7]: import time
```

Insert items at the start of a linked list, multiples of 10^5 , linear blowup

```
In [8]: for i in range(1,5):
        l1 = Node()
        start = time.perf_counter()
        for j in range(i*100000):
            l1.insert(j)
        elapsed = time.perf_counter() - start
        print(i*100000,elapsed)
```

```
100000 0.2101113050011918
200000 0.38968896699952893
300000 0.47890829099924304
400000 0.6604866980051156
```

Insert items at the start of a Python list, multiples of 10^5 , quadratic blowup

```
In [9]: for i in range(1,5):
        l2 = []
        start = time.perf_counter()
        for j in range(i*100000):
            l2.insert(0,j)
        elapsed = time.perf_counter() - start
        print(i*100000,elapsed)
```

```
100000 2.7054551640030695
200000 10.720501578995027
300000 24.08416212099837
400000 53.89232549800363
```