Linked Lists

CSC148, INTRODUCTION TO COMPUTER SCIENCE

DIANE HORTON, JONATHAN CALVER & SADIA SHARMIN

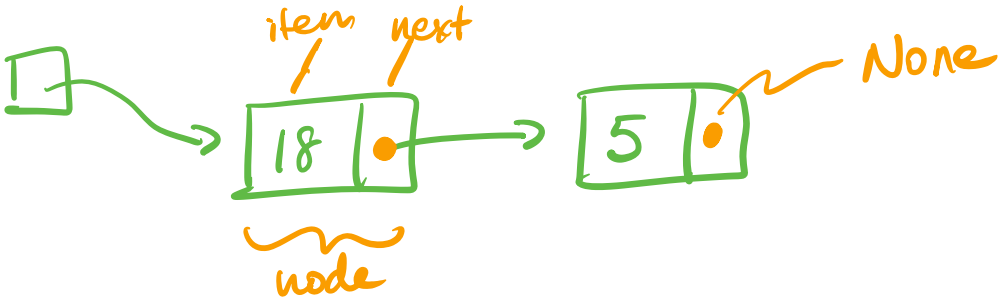
There are two major list implementations

1. **Array-based** lists store references to elements in contiguous blocks of memory.

2. **Linked** lists can store elements anywhere, but each element must store a reference to the *next* element in the list.

Our goals this week

1. Work with linked lists by implementing the same operations as Python's built-in `list`.
2. Analyze the running time of our linked list methods and compare them to the array-based `list`.



Code summary

private

```
class _Node:
    item: Any
    next: Optional[_Node]

class LinkedList:
    _first: Optional[_Node]
```

template:

```
curr = self._first
while curr is not None:
    ... curr.item ...
    curr = curr.next
```

Takeaways

Code templates are useful.

Code templates aren't everything.

Writing a stopping condition is often *easier to understand* than writing a loop condition.

Linked list insertion and deletion

IT'S ALL ABOUT THE LINKS.

```
def insert(self, index: int, item: Any) -> None:
    """Insert the given item at the given index.

    Raise IndexError if index > len(self)
    or index < 0.
    Adding to the end of the list is okay.
    """
```

Recapping the key ideas

1. Figure out when we need to modify `self._first` vs. a `_Node` in the list.

2. When `index` > 0, iterate to the (`index`-1)th node and update links.

```
def pop(self, index: int) -> Any:
    """Remove and return the item at the given
    index.

    Raise IndexError if index >= len(self)
    or index < 0.
    """
```

Same key ideas!

1. Figure out when we need to modify `self._first` vs. a `_Node` in the list.

2. When `index` > 0, iterate to the (`index`-1)th node and update links.

The “problem of previous”

Strategy #1: iterate to the node *before* the desired position.

```
i = 0
curr = self._first
while not (curr is None or i == index – 1):
    curr = curr.next
    i += 1
```

The “problem of previous”

Strategy #2: track the previous node explicitly

```
i = 0
prev = None
curr = self._first
while not (curr is None or i == index):
    prev, curr = curr, curr.next
    i += 1
```

Linked list operation running time

HOW DO LINKED LIST OPERATIONS PERFORM COMPARE TO ARRAY-BASED LISTS?

Recall from last week…

Python's lists are *array-based*. Each list stores the ids of its elements in a contiguous block of memory.

Every insertion and deletion causes every element *after* the changed index to move.

When analysing running time, we use *Big-Oh notation* to capture the *type of growth* of running time as a function of input size.

E.g., O(1): "constant growth", O(n): "linear growth"