CSC 148: Introduction to Computer Science

Week 5

Linked list operation running time

How do linked list operations perform compare to array-based lists?

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

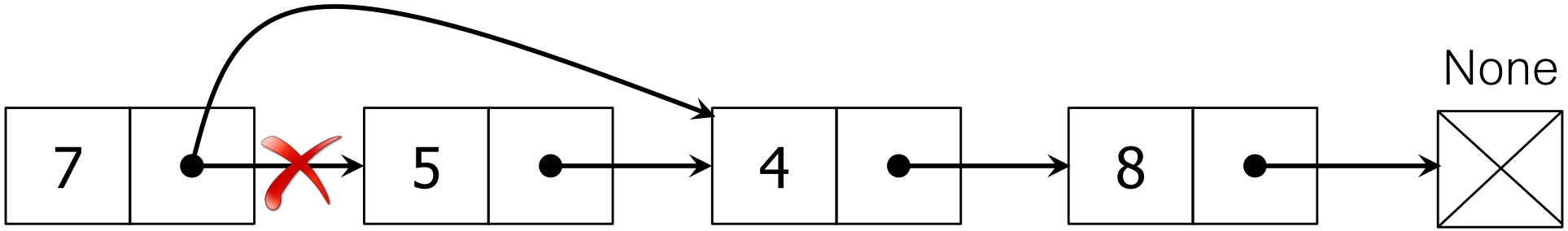
Recall …

- 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.

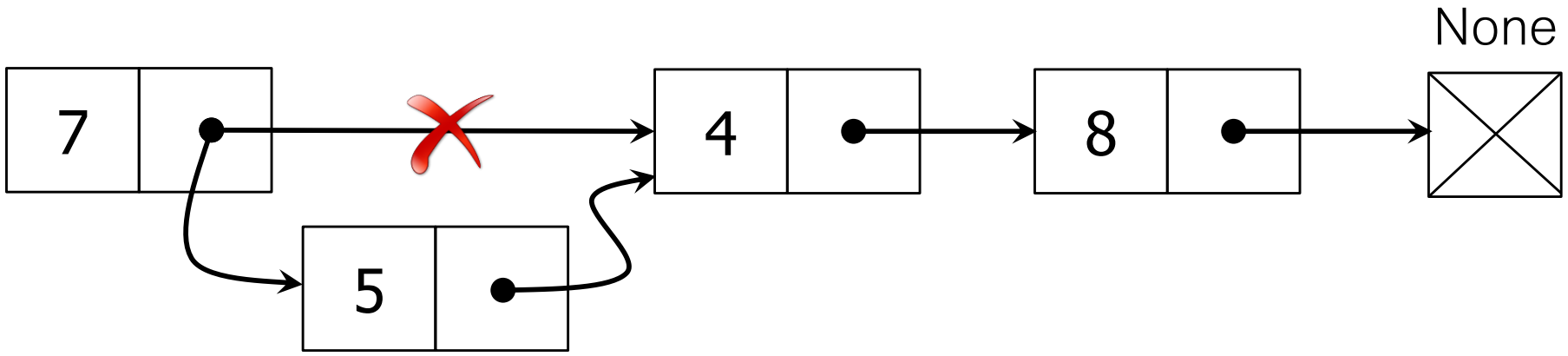
- When analysing running time, we use Big-Oh notation to capture the type of growth of running time as a function of input size.
- E.g., O(1): "constant growth", O(n): "linear growth"

What about Linked Lists?

- Remove an element – much faster (no need to "shift" anything)



- Insert an element – no need to shift subsequent elements



- Just adjust a couple of references, no moving memory!

Compare and Contrast ...

- At which end of a Python list would it be best to insert an element at? Which index would it be easiest to remove?

- At which end of a linked list would it be best to insert an element at? Which index would it be easiest to remove?

LinkedList.__contains__

- We care about running time as a function of input size:
  - “constant” O(1)
  - “linear” O(n)
  - “quadratic” $O(n^2)$

Worksheet

- Analysing the running time of linked list operations ...

LinkedList.__contains__

```
def __contains__(self, item: Any) -> bool:
    curr = self._first
    while curr is not None:
        if curr.item == item:
            return True
    return False
```

LinkedList.__contains__

- Running time can vary, even for a fixed input size!

- We'll revisit this idea later in the course.

Other linked list designs and efficiency considerations

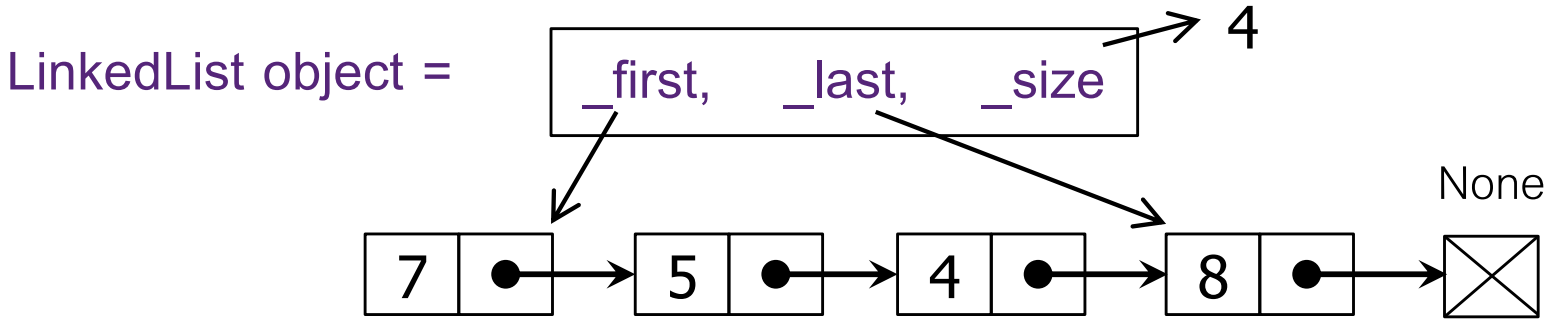
How does the new design affect the implementation?

Why would we want to do this? Efficiency?

How else can we organize nodes? Why?

Other Designs?

- Consider storing more info:
  - _first
  - _last
  - _size



- Ask yourselves these questions:
  - 1. Does the implementation of operations change?
    - Draw diagrams!
  - 2. What are the performance implications?

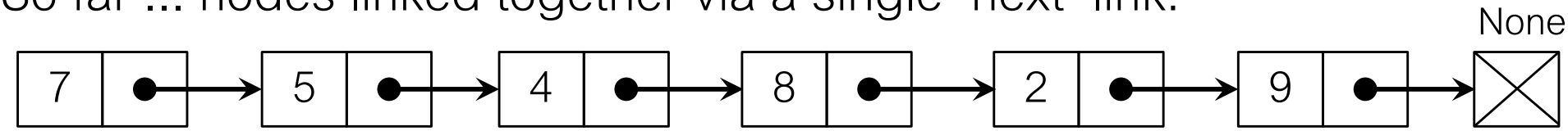
Implications?

- Consider that this new linked list implementation only supports insert and delete operations
  - How does it benefit us to store _last and _size?

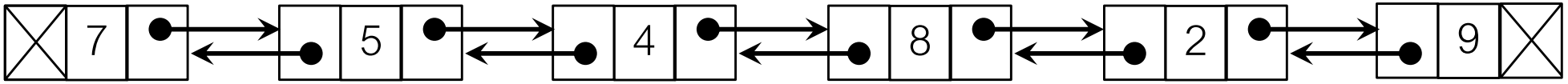
- Let's re-assess efficiency for this implementation:
  - At which end of a linked list would it be best to insert an element at? Which index would it be easiest to remove?
  - How does this implementation fare against the previous one?
  - Time efficiency vs. Space efficiency ...

Other Options?

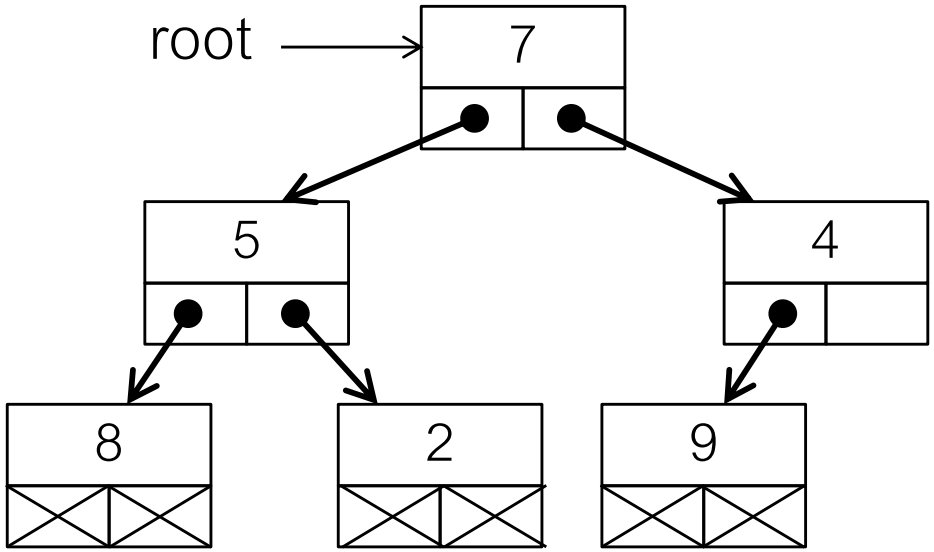
- So far ... nodes linked together via a single "next" link:



- How about a doubly linked list (next + prev links)? Circular, optionally?



- What about a "hierarchical" structure of Nodes? .. Aka Tree



- Advantage: search path to each item in a tree is much shorter than in a linked list! (if the tree is reasonably balanced..) *To be continued ...*