Doubly Linked Lists

• Node with two links
  • next/previous, forward/back

• Sentinel nodes
  • head and tail are actual nodes instead of references
  • allows code reuse – no special conditions for head/tail
  • list always contains at least 2 nodes
    • but report just non-sentinels to user
Doubly Linked List Operations

• Insertion
• Deletion
• Now we can implement a deque
  • why would deques not work well with singly linked lists?
Positional Lists

• sometimes need non-canonical operations
  • insert/delete element at arbitrary location
  • canonical API can be expensive
    • pop everything off onto auxiliary stack
    • alter or whatever on item in question
    • push everything back onto original stack

• Position class gives [] array element access to linked list
PositionalList

• positional_list.py
• Validate position objects
• Access and update positions
• Wraps all doubly linked list operations so we only deal with one API
• position objects are abstractions of pointers/references
PositionalList vs Python List

- `insertion_sort.py`
  - Python list version
- `insertion_sort_positional.py`
  - PositionalList version

- (We’ll revisit insertion sort after midterm.)
Arrays vs Linked Lists

• Arrays
  • access is $O(1)$
  • operations often really are simple
  • might use less memory (unless dynamic)

• Linked lists
  • guaranteed worst case complexities
  • constant time insertions and deletions when site is known