Data Structures

Lecture 4
Linked List Variations + Doubly Linked List

Credit: Prantik Paul, Lecturer, BRAC University

Singly Linked List (Problems)

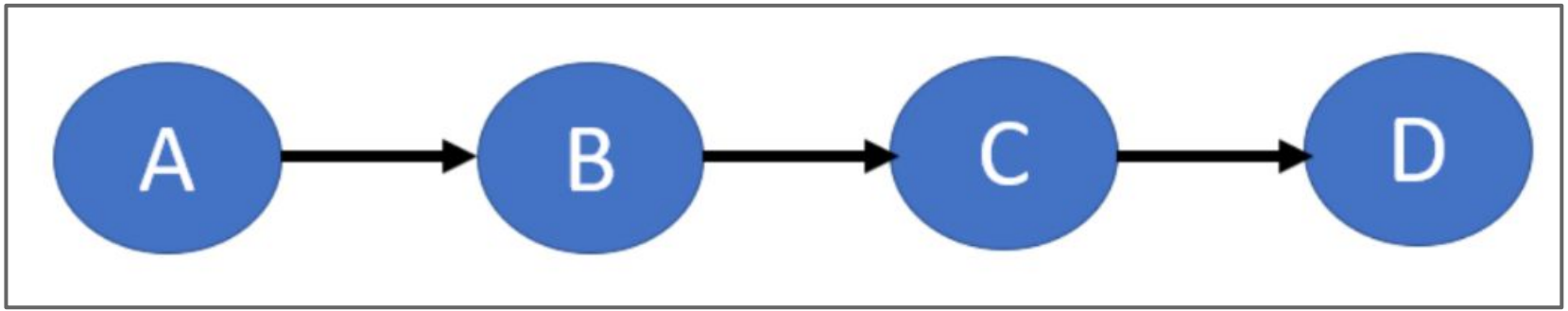
1. Traversal
2. Determine where to stop
3. Operations near the end
4. Moving back in Time

Types of Linked List

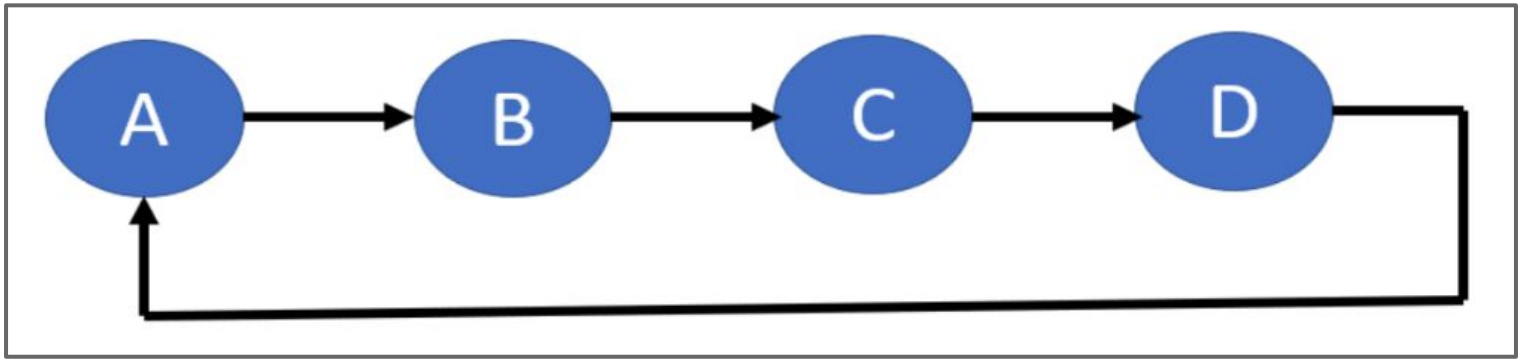
Head Type	Connection	Structure

Category 1	Category 2	Category 3
Non-Dummy Headed	Singly	Linear
Dummy Headed	Doubly	Circular

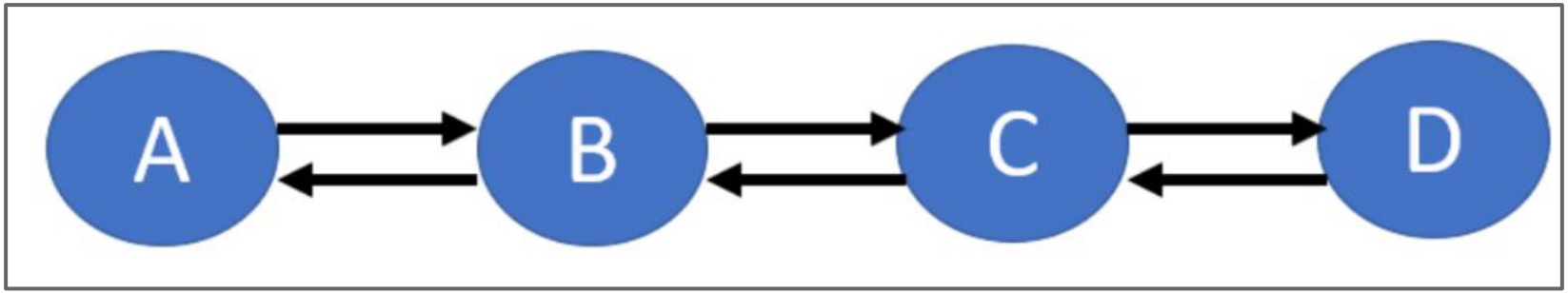
Non-Dummy Headed Singly Linear Linked List



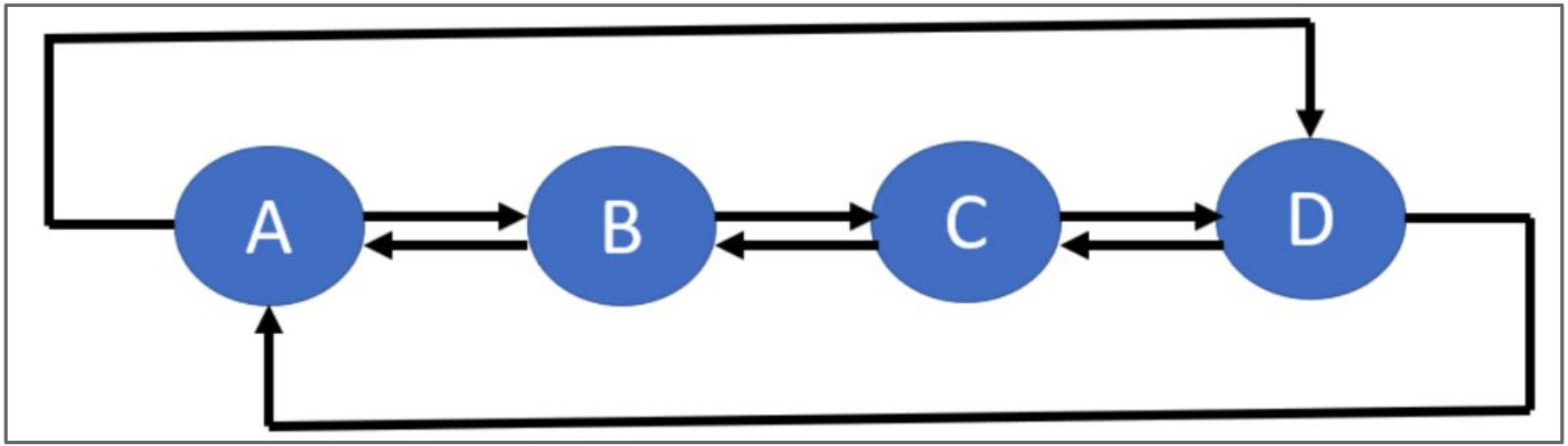
Non-Dummy Headed Singly Circular Linked List



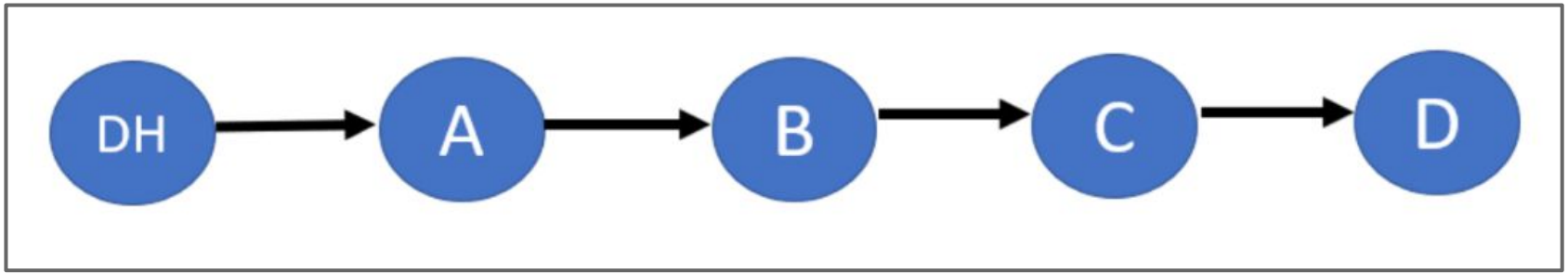
Non-Dummy Headed Doubly Linear Linked List



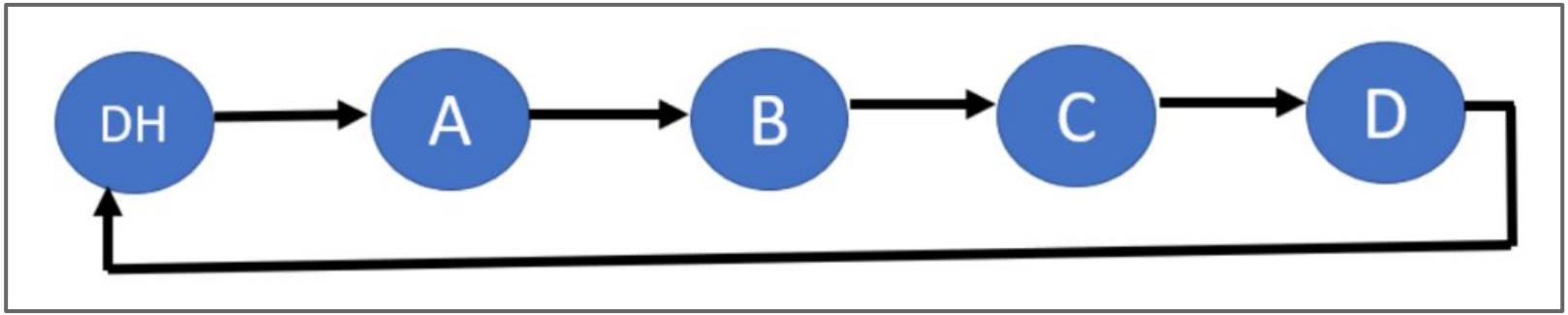
Non-Dummy Headed Doubly Circular Linked List



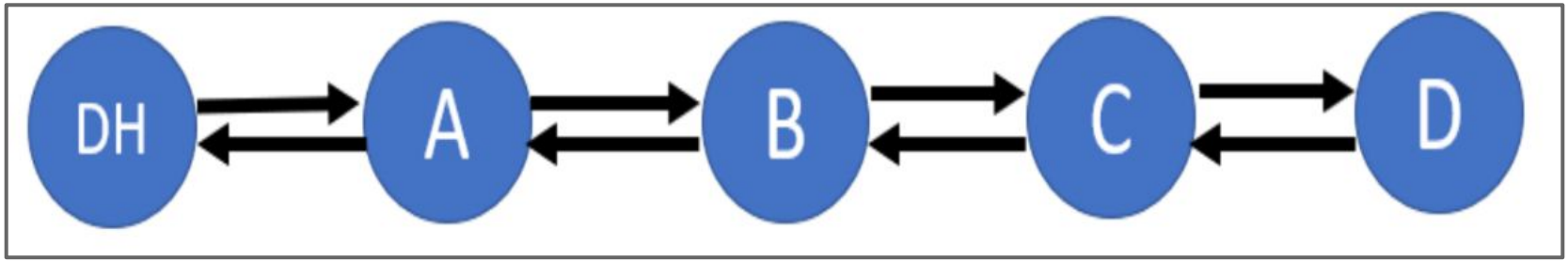
Dummy Headed Singly Linear Linked List



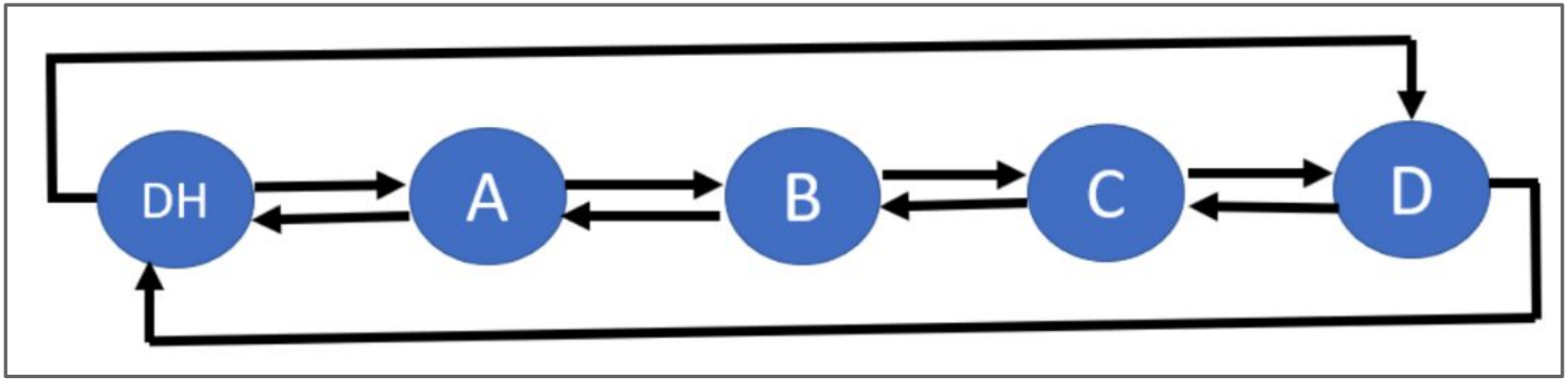
Dummy Headed Singly Circular Linked List



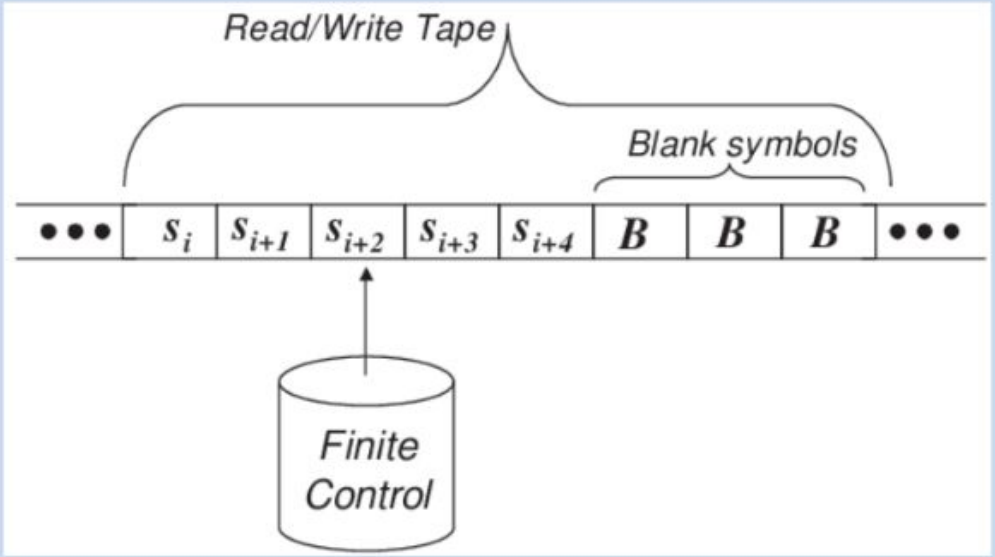
Dummy Headed Doubly Linear Linked List

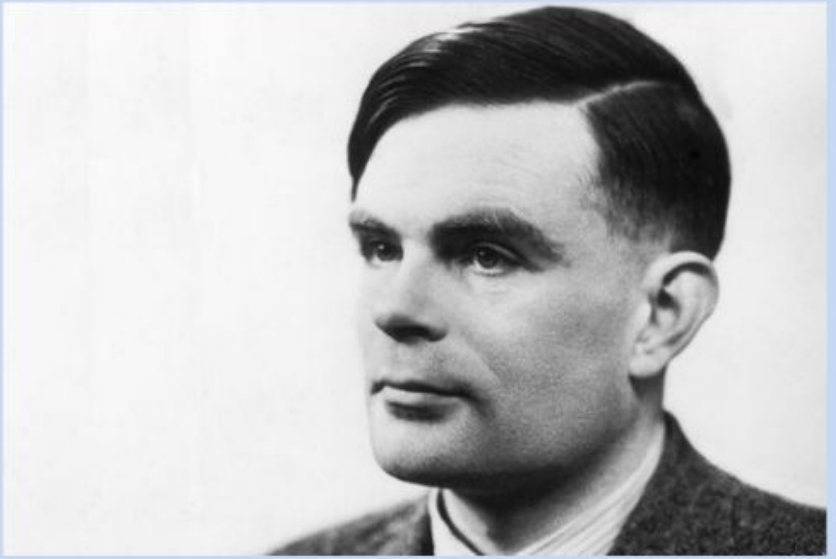


Dummy Headed Doubly Circular Linked List



Why Doubly Linked List





Doubly Linked List (Node Class)

```
class DoublyNode:
    def __init__(self, elem, next, prev):
        self.elem = elem
        self.next = next # To store the next node's reference.
        self.prev = prev # To store the previous node's reference.
```

DH Doubly Linked List (Creation)

```
1.  FUNCTION createList(arr):
2.    dh = DoublyNode(None, None, None)
3.    dh.next = dh
4.    dh.prev = dh
5.    tail = dh
6.
7.    FOR i in range(len(arr)):
8.      n = DoublyNode(arr[i], dh, tail)
9.      tail.next = n
10.     tail = tail.next
11.     dh.prev = tail
12.
13.   RETURN dh
14. END FUNCTION
```

Doubly Linked List (Creation)

```
def createList(a):
    dh = DoublyNode(None, None, None)
    dh.next = dh
    dh.prev = dh
    tail = dh

    for i in range(len(a)):
        n = DoublyNode(a[i], dh, tail)
        tail.next = n
        tail = tail.next
        dh.prev = tail

    return dh
```

DH Doubly Linked List (Iteration)

```
1.  FUNCTION iteration(dh):
2.    temp = dh.next
3.    WHILE temp != dh:
4.      PRINT temp.elem
5.      temp = temp.next
6.  END FUNCTION
```

DH Doubly Linked List (Iteration)

```
def iteration(dh):
    temp = dh.next
    while temp != dh:
        print(temp.elem)
        temp = temp.next
```

DH Doubly Linked List (GetNode)

```
1.  FUNCTION nodeAt(dh, idx):
2.    temp = dh.next
3.    c = 0
4.    WHILE temp != dh:
5.      IF c == idx:
6.        RETURN temp
7.      c += 1
8.      temp = temp.next
9.    RETURN None # Invalid Index
10. END FUNCTION
```

DH Doubly Linked List (GetNode)

```
def nodeAt(dh, idx):
  temp = dh.next
  c = 0
  while temp != dh:
    if c == idx:
      return temp
    c += 1
    temp = temp.next
  return None # Invalid Index
```

DH Doubly Linked List (Insertion)

```
1.  FUNCTION insertion(dh, elem, idx):
2.    # Assuming the idx is valid
3.    node_to_insert = DoublyNode(elem, None, None)
4.    indexed_node = nodeAt(dh, idx) # Retriving the node at that index
5.    prev_node = indexed_node.prev # There will always be a previous node
6.    # Change the connection
7.    # Observe that no special case is needed
8.    node_to_insert.next = indexed_node
9.    node_to_insert.prev = prev_node
10.   prev_node.next = node_to_insert
11.   indexed_node.prev = node_to_insert
12. END FUNCTION
```

DH Doubly Linked List (Insertion)

```
def insertion(dh, elem, idx):
  # Assuming the idx is valid
  node_to_insert = DoublyNode(elem, None, None)
  indexed_node = nodeAt(dh, idx) # Retriving the node at that index
  prev_node = indexed_node.prev # There will always be a previous node
  # Change the connection
  # Observe that no special case is needed
  node_to_insert.next = indexed_node
  node_to_insert.prev = prev_node
  prev_node.next = node_to_insert
  indexed_node.prev = node_to_insert
```

DH Doubly Linked List (Removal)

```
1.  FUNCTION removal(dh, idx):
2.    # Assuming the idx is valid
3.    node_to_remove = nodeAt(dh, idx)
4.    prev_node = node_to_remove.prev
5.    next_node = node_to_remove.next
6.    # Change the connection
7.    # No special case is needed
8.    prev_node.next = next_node
9.    next_node.prev = prev_node
10.   node_to_remove.next = None
11.   node_to_remove.prev = None
12.   RETURN node_to_remove.elem # Returning the removed element
13. END FUNCTION
```

DH Doubly Linked List (Removal)

```
def removal(dh, idx):
  # Assuming the idx is valid
  node_to_remove = nodeAt(dh, idx)
  prev_node = node_to_remove.prev
  next_node = node_to_remove.next
  # Change the connection
  # No special case is needed
  prev_node.next = next_node
  next_node.prev = prev_node
  node_to_remove.next = None
  node_to_remove.prev = None
  return node_to_remove.elem # Returning the removed element
```