Data Structures

Lecture 3
Linked List

Credit: Prantik Paul, Lecturer, BRAC University

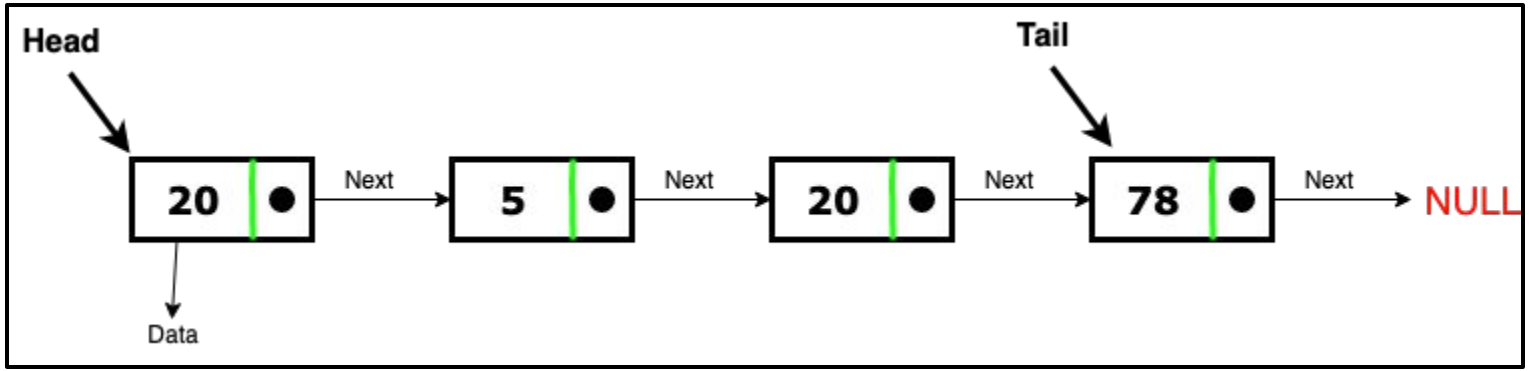
Array Problems

Fixed Capacity

Insert

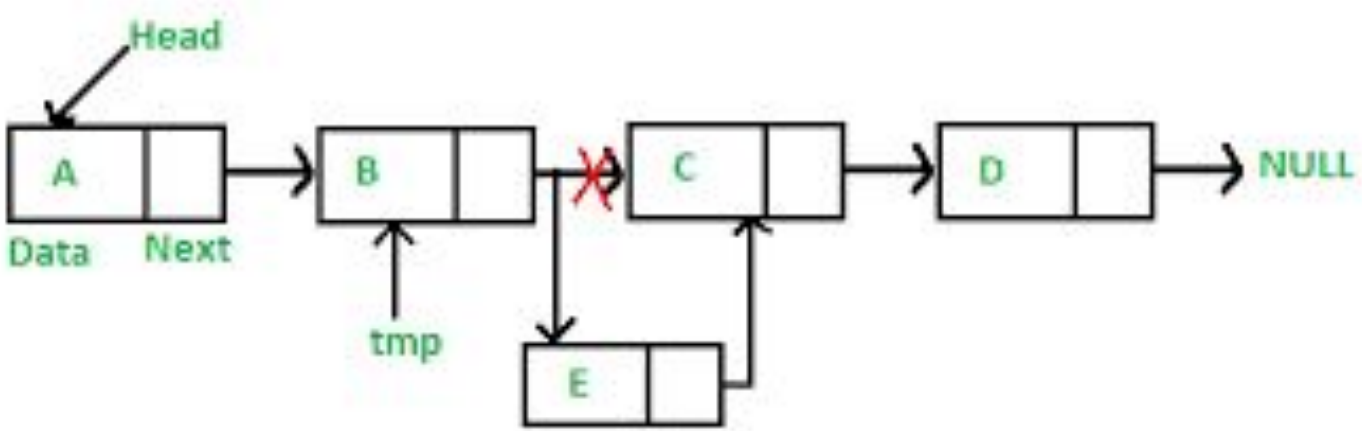
Removal

Solution? Linked List



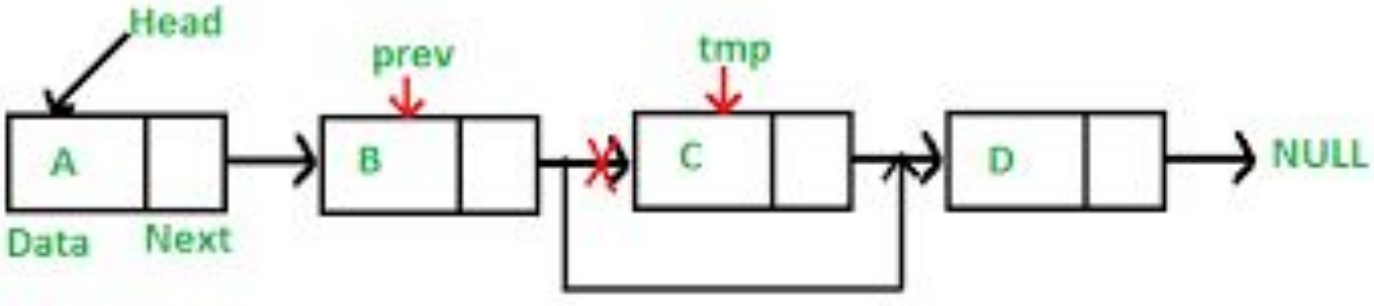
Linked List

Easy Insert



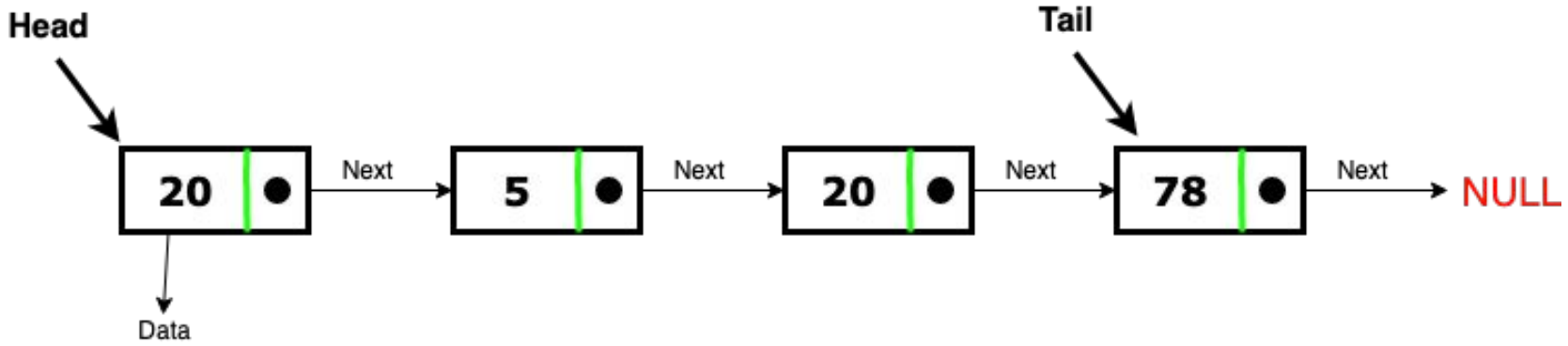
Linked List

Easy Removal



Linked List

Easy Resize



Linked List - Problems?

Random Access

Space

Linked List - Initialization

```
# Node class design
class Node:
    def __init__(self, e, n):
        self.elem = e
        self.next = n
```

Linked List - Creation (From an Array)

```
1.  FUNCTION create_list(arr)
2.      head = Node(arr[0], NULL)
3.      tail = head
4.      FOR i = 1 TO size of arr - 1
5.          n = Node(arr[i], NULL)
6.          tail.next = n
7.          tail = tail.next
8.      END FOR
9.      RETURN head
10. END FUNCTION
```

Linked List - Creation (From an Array)

```
# Creating a list
def createList(a):
    head = Node(a[0], None)
    tail = head
    for i in range(1, len(a)):
        n = Node(a[i], None)
        tail.next = n
        tail = tail.next
    return head
```

Linked List - Iteration

```
2.      temp = head
3.      WHILE temp != NULL
4.          PRINT temp.element
5.          temp = temp.next
6.      END WHILE
7.  END FUNCTION
```

Linked List - Iteration

```
# Iteration over a linked list
def iteration(head):
    temp = head
    while temp != None:
        print(temp.element)
        temp = temp.next
```

Linked List - Count

```
1.  FUNCTION count(head)
2.      count = 0
3.      temp = head
4.      WHILE temp != NULL
5.          count = count + 1
6.          temp = temp.next
7.      END WHILE
8.      RETURN count
9.  END FUNCTION
```

Linked List - Count

```
# Counting number of element in the list
def count(head):
    count = 0
    temp = head
    while temp != None:
        count += 1
        temp = temp.next
    return count
```

Linked List - Get Node

```
1.  FUNCTION nodeAtt(head, idx)
2.      count = 0
3.      temp = head
4.      WHILE temp != NULL
5.          IF count = idx
6.              RETURN temp
7.          END IF
8.          temp = temp.next
9.          count = count + 1
10.     END WHILE
11.     RETURN NULL
12. END FUNCTION
```

Linked List - Get Node

```
# Getting node of an specific index
def nodeAt(head, idx):
    count = 0
    temp = head
    obj = None
    while temp != None:
        if count == idx:
            obj = temp
            break
        temp = temp.next
        count += 1
    if obj == None:
        print("Invalid index")
    return obj
```

Linked List - Get Element

```
1.  FUNCTION elem_at(head, idx)
2.      count = 0
3.      temp = head
4.      WHILE temp != NULL
5.          IF count = idx
6.              RETURN temp.elem
7.          END IF
8.          temp = temp.next
9.          count = count + 1
10.     END WHILE
11.     RETURN NULL
12. END FUNCTION
```

Linked List - Get Element

```
# Getting element of an specific index
def elemAt(head, idx):
    count = 0
    temp = head
    obj = None
    while temp != None:
        if count == idx:
            obj = temp.element
            break
        temp = temp.next
        count += 1
    if obj == None:
        print("Invalid index")
    return obj
```

Linked List - Update Value at Index

```
1.  FUNCTION set(head, idx, elem)
2.      count = 0
3.      temp = head
4.      is_updated = False
5.      WHILE temp != NULL
6.          IF count = idx
7.              temp.elem = elem
8.              is_updated = True
9.              BREAK
10.         END IF
11.         temp = temp.next
12.         count = count + 1
13.     END WHILE
14.     IF is_updated
15.         PRINT "Value successfully updated!!!!"
16.     ELSE
17.         PRINT "Invalid index"
18.     END IF
19. END FUNCTION
```

Linked List - Update Value at Index

```
# Setting new element of an specific index
def set(head, idx, elem):
  count = 0
  temp = head
  isUpdated = False
  while temp != None:
    if count == idx:
      temp.elem = elem
      isUpdated = True
      break
    temp = temp.next
    count += 1
  if isUpdated:
    print("Value successfully updated!!!!")
  else:
    print("Invalid index")
```

Linked List - Search Element

```
1.  FUNCTION index_of(head, elem)
2.      temp = head
3.      count = 0
4.      WHILE temp != NULL
5.          IF elem = temp.elem
6.              RETURN count
7.          END IF
8.          count = count + 1
9.          temp = temp.next
10.     END WHILE
11.     RETURN -1 # Here -1 represents the absence of element in the list
12. END FUNCTION
```

Linked List - Search Element

```
# Getting index of an specific element
def indexOf(head, elem):
  temp = head
  count = 0
  while temp != None:
    if elem == temp.elem:
      return count
    count += 1
    temp = temp.next
  return -1 # Here -1 represents the absence of element in the list
```

Linked List - Search Element

```
1.  FUNCTION contains(head, elem)
2.      temp = head
3.      WHILE temp != NULL
4.          IF elem = temp.elem
5.              RETURN True
6.          END IF
7.          temp = temp.next
8.      END WHILE
9.      RETURN False
10. END FUNCTION
```

Linked List - Search Element

```
def contains(head, elem):
    temp = head
    while temp != None:
        if elem == temp.elem:
            return True
        temp = temp.next
    return False
```

Linked List - Insert Element

```
1.  FUNCTION insert(head, elem, idx)
2.      total_nodes = count(head)
3.      IF idx = 0 # Inserting at the beginning
4.          n = Node(elem, head)
5.          head = n
6.      ELSE IF idx >= 1 AND idx < total_nodes # Inserting at the middle
7.          n = Node(elem, head)
8.          n1 = node_at(head, idx - 1)
9.          n2 = node_at(head, idx)
10.         n.next = n2
11.         n1.next = n
12.     ELSE IF idx = total_nodes # Inserting at the end
13.         n = Node(elem, NULL)
14.         n1 = node_at(head, total_nodes - 1)
15.         n1.next = n
16.     ELSE
17.         PRINT "Invalid Index"
18.     END IF
19.     RETURN head
20. END FUNCTION
```

Linked List - Insert Element

```
def insert(head, elem, idx):
  total_nodes = count(head)
  if idx == 0: # Inserting at the beginning
    n = Node(elem, head)
    head = n
  elif idx >= 1 and idx < total_nodes: # Inserting at the middle
    n = Node(elem, head)
    n1 = nodeAt(head, idx - 1)
    n2 = nodeAt(head, idx)
    n.next = n2
    n1.next = n
  elif idx == total_nodes: # Inserting at the end
    n = Node(elem, None)
    n1 = nodeAt(head, total_nodes - 1)
    n1.next = n
  else:
    print("Invalid Index")
  return head
```

Linked List - Remove Element

```
1.  FUNCTION remove(head, idx)
2.      IF idx = 0 # Removing first element
3.          head = head.next
4.      ELSE IF idx >= 1 AND idx < count(head) # Removing middle element
5.          n1 = node_at(head, idx - 1)
6.          removed_node = n1.next
7.          n1.next = removed_node.next
8.      ELSE
9.          PRINT "Invalid Index"
10.     END IF
11.     RETURN head
12. END FUNCTION
```

Linked List - Remove Element

```
def remove(head, idx):
  if idx == 0: # Removing first element
    head = head.next
  elif idx >= 1 and idx < count(head): # Removing middle element
    n1 = nodeAt(head, idx - 1)
    removed_node = n1.next
    n1.next = removed_node.next
  else:
    print("Invalid Index")
  return head
```

Linked List - Rotate Right

```
1.  FUNCTION rotate_right(head):
2.    last_node = head.next
3.    second_last_node = head
4.    WHILE last_node.next != None:
5.      last_node = last_node.next
6.      second_last_node = second_last_node.next
7.    last_node.next = head
8.    second_last_node.next = None
9.    head = last_node
10.   RETURN head
11. END FUNCTION
```

Linked List - Rotate Right

```
def rotate_right(head):
  last_node = head.next
  second_last_node = head
  while last_node.next != None:
    last_node = last_node.next
    second_last_node = second_last_node.next
  last_node.next = head
  second_last_node.next = None
  head = last_node
  return head
```

Linked List - Rotate Left

```
1.  FUNCTION rotate_left(head):
2.    new_head = head.next
3.    temp = new_head
4.    WHILE temp.next != None:
5.      temp = temp.next
6.    temp.next = head
7.    head.next = None
8.    head = new_head
9.    RETURN head
10. END FUNCTION
```

Linked List - Rotate Left

```
def rotate_left(head):
    new_head = head.next
    temp = new_head
    while temp.next != None:
        temp = temp.next
    temp.next = head
    head.next = None
    head = new_head
    return head
```

Linked List - Reverse List (Out of Place)

```
1.  FUNCTION reverse_out_of_place(head)
2.      new_head = Node(head.elem, NULL)
3.      temp = head.next
4.      WHILE temp != NULL
5.          n = Node(temp.elem, new_head)
6.          new_head = n
7.          temp = temp.next
8.      END WHILE
9.      RETURN new_head
10. END FUNCTION
```

Linked List - Reverse List (Out of Place)

```
def reverse_out_of_place(head):
    new_head = Node(head.elem, None)
    temp = head.next
    while temp != None:
        n = Node(temp.elem, new_head)
        new_head = n
        temp = temp.next
    return new_head
```

Linked List - Reverse List (In Place)

```
1.  FUNCTION reverse_in_place(head)
2.      new_head = NULL
3.      temp = head
4.      WHILE temp != NULL
5.          n = temp.next
6.          temp.next = new_head
7.          new_head = temp
8.          temp = n
9.      END WHILE
10.     RETURN new_head
11. END FUNCTION
```

Linked List - Reverse List (In Place)

```
def reverse_in_place(head):
    new_head = None
    temp = head
    while temp != None:
        n = temp.next
        temp.next = new_head
        new_head = temp
        temp = n
    return new_head
```