CSC508 Data Structures

Topic 4 : Linked List Variation

Compiled & edited by: Zahid Zainal

Recap

- Linked list
  - Definition
  - Characteristics
  - Properties
- `LinkedList` class
- Linked list operation

Topic Structure

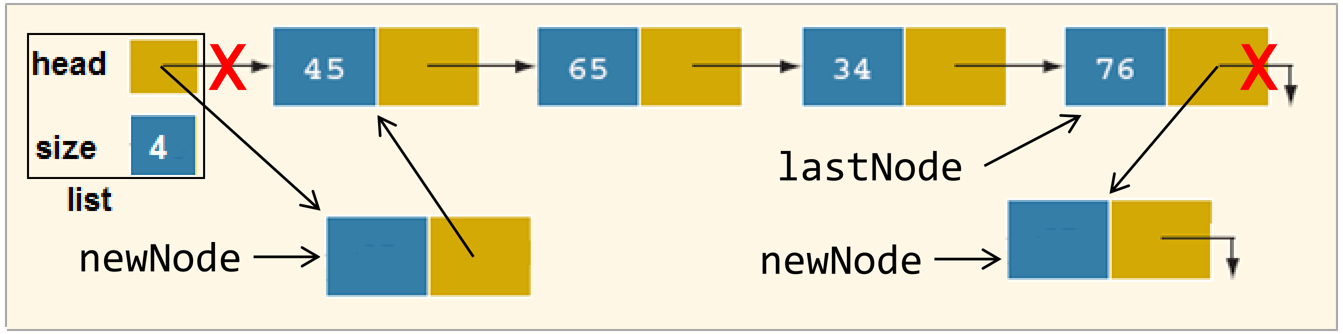
- Improving Linked List
- Doubly Linked
- Circular Linked List
- Multidimensional Linked List

Learning Outcomes

- At the end of this lesson, students should be able to:
  - Explain improvement needed for linked list efficiency
  - Describe variation of linked lists
  - Compare the need for array and linked lists

Improving Linked List

- Inserting a new item at beginning of a linked list is fast (no traversal required)



- While inserting at the end of the list required us to traverse the whole list to reach the last element
- If in a program there is a need to frequently insert items at end of the list, it's worth to change the data structure to allow more efficient implementation

Improving Linked List (cont.)

- Introducing a `tail` reference, that points to the last node will make the process more efficient.

```
class MyLinkedList{
    Node head;
    Node tail;
    int size;

    MyLinkedList(){
        head = null;
        tail = null;
        size = 0;
    }
}
```

- `insertLast()` and `insertFirst()` methods need to be revised to accommodate the `tail`

Compiled & edited by: Zahid Zainal

Improving Linked List (cont.)

```
public void insertFirst(int x) {
    Node newNode = new Node();
    newNode.data = x;
    newNode.next = head;
    head = newNode;
    if (tail == null)
        tail = head;
    size++;
}
```

Case when a node is added into an empty list

```
public void insertLast (int x) {
    if (head == null)
        insertFirst(x);
    else {
        Node newNode = new Node();
        newNode.data = x;
        newNode.next = null;

        tail.next = newNode;
        tail = newNode;

        size++;
    }
}
```

Link the last node to the new node added into the list

Update `tail` to point to the new last node

No traversing needed.

Compiled & edited by: Zahid Zainal

Improving Linked List (cont.)

- Tail does not improve `removeLast()` operation.
- How about if we want to display the element in reverse?
  - Use nested loop

    Higher time complexity
  - Use recursion

    Higher space complexity
- Efficiency of accessing element at specific index depends on the size of the linked list.

  How can we improve?

Compiled & edited by: Zahid Zainal

Reverse Print

- Nested Loop

First loop to set the limit in decreasing order

```
public void printReverse(){
    for(int limit = size-1; limit>=0;
limit--){
        Node temp = head;
        for(int i= 0; i<limit; i++) {
            temp = temp.next;
            System.out.println(temp.data);
        }
    }
}
```

Second loop to traverse the list and display

- Recursion

stop recursion if we reach list end

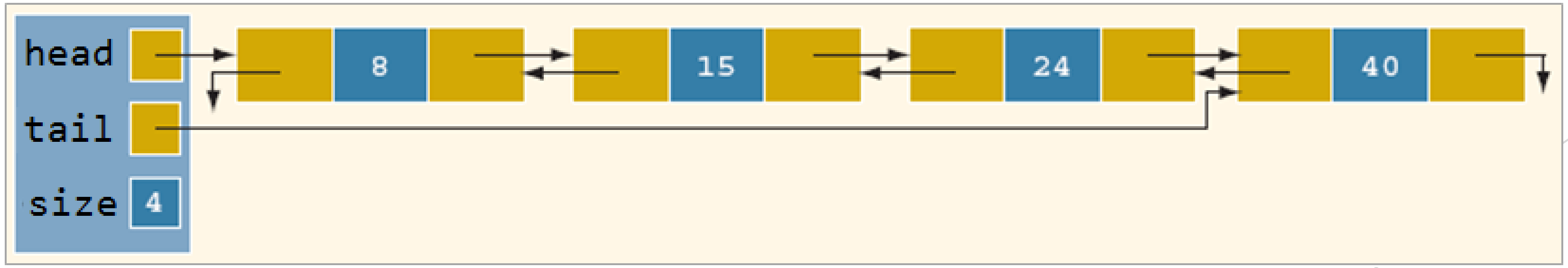
```
public void printReversell(){
    printRecursive(head);
}
public void printRecursive(Node temp){
    if(temp != null) {
    printRecursive(temp.next);
    System.out.println(temp.data);
    }
}
```

print remaining items first

then print current item

Doubly Linked List

- A linked list where every node has access to the next and previous node.
  - has a next reference variable and a back reference variable
  - contains the address of the next node (except the last node)
  - contains the address of the previous node (except the first node)
- Traversal can happened in both direction



Compiled & edited by: Zahid Zainal

Doubly Linked List Operation

- Node definition

```
class Node{
    int data;
    Node next;
    Node prev;
}
```

- Insert at the beginning

```
public void doublyInsertFirst(int x) {
    Node newNode = new Node();
    newNode.data = x;
    newNode.next = head;
    newNode.prev = null;
    head = newNode;
    if (tail == null)
        tail = head;
    else
        newNode.next.prev = newNode;
    size++;
}
```

prev for first node is always null

case on insert into an empty list

Update prev of former first node to point to the new first node

Compiled & edited by: Zahid Zainal

Doubly Linked List Operation (cont.)

- Insert at the beginning

```
public void insertLast (int x) {
    Node newNode= new Node();
    newNode.data = x;
    newNode.next = null;
    newNode.prev = tail;
    tail = newNode;
    if (head == null)
        head = newNode;
    else
        newNode.prev.next = newNode;
    size++;
}
```

case on insert into an empty list

- Reverse print

```
public void reversePrint () {
    Node temp = tail;
    while (temp != null) {
        System.out.println(temp.data);
        temp = temp.prev;
    }
}
```

Traversing from the end of the list

Compiled & edited by: Zahid Zainal

Doubly Linked List Operation (cont.)

Write you own methods for

Removing node from doubly link list???

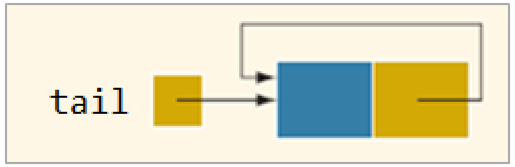
Compiled & edited by: Zahid Zainal

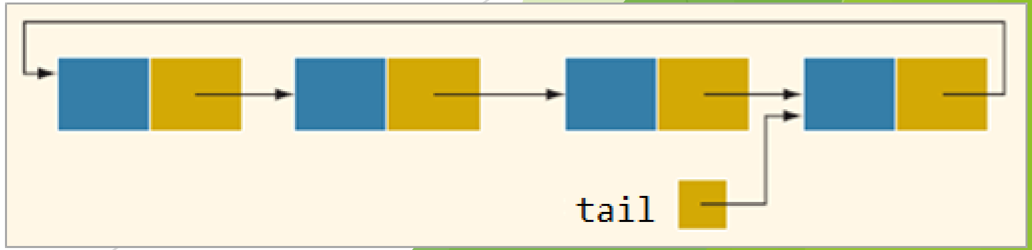
Notes on Doubly Linked Lists

- The main advantage of Doubly Linked Lists is to efficiently traverse list nodes in both directions
  - Applications: to implement back and forward buttons in a web browser, undo and redo actions, etc.
- We can also use two-way traversal to reduce time required to reach an item at a given index
  - if index <= size/2, start from head and move forward
  - otherwise, start at tail and move backward
  - on average, would make insertItemAt(), deleteItemAt(), getItemAt() and setItemAt() 2 times faster

Circular Linked List

- Some problems are inherently circular (squares on a Monopoly board), and many solutions can be solved more naturally using a circular data structure (round robin scheduling, players taking turns, playing video and sound files in “looping” mode, etc.)
- A linked list is made circular if its end points to its beginning, i.e. the last node points to the first one
  - Circular traversal is made more natural, rather than always testing if we have reached to the end and starting over

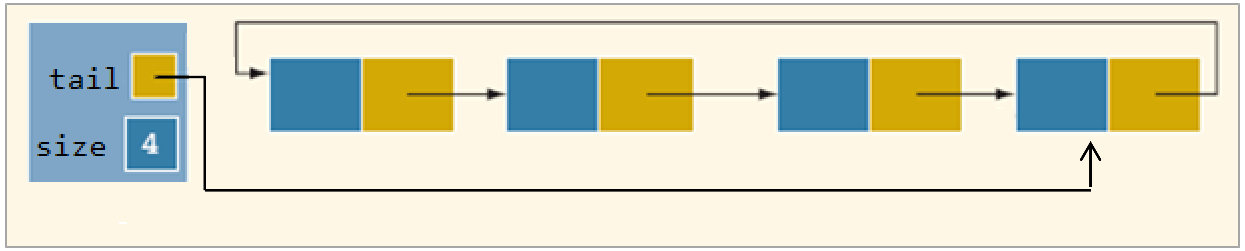




Compiled & edited by: Zahid Zainal

Circular Linked List Characteristics

- Data representation for a Circular Linked List is not different from the normal Linked List
  - The concept is just not to store NULL at the link field of last node, and respect that in implementing all operations
  - Often, tail pointer is only used instead of head

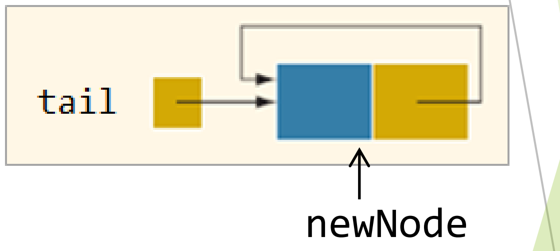


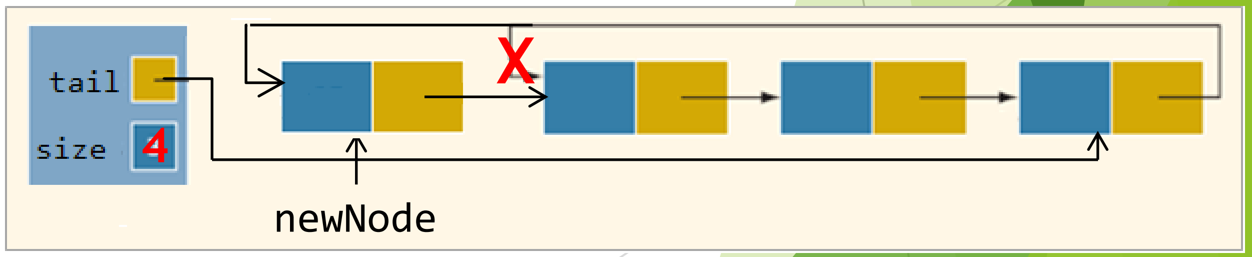
Compiled & edited by: Zahid Zainal

Circular Linked List Operation

- Insert at the beginning of the list

```
public void insertAtBeginning(int x){
    Node newNode= new Node();
    newNode.data = x;
    if (tail == null
        tail = newNode;
        tail.next = tail;
    }
    else{
        newNode.next = tail.next;
        tail.next = newNode;
    }
    size++;
}
```





Compiled & edited by: Zahid Zainal

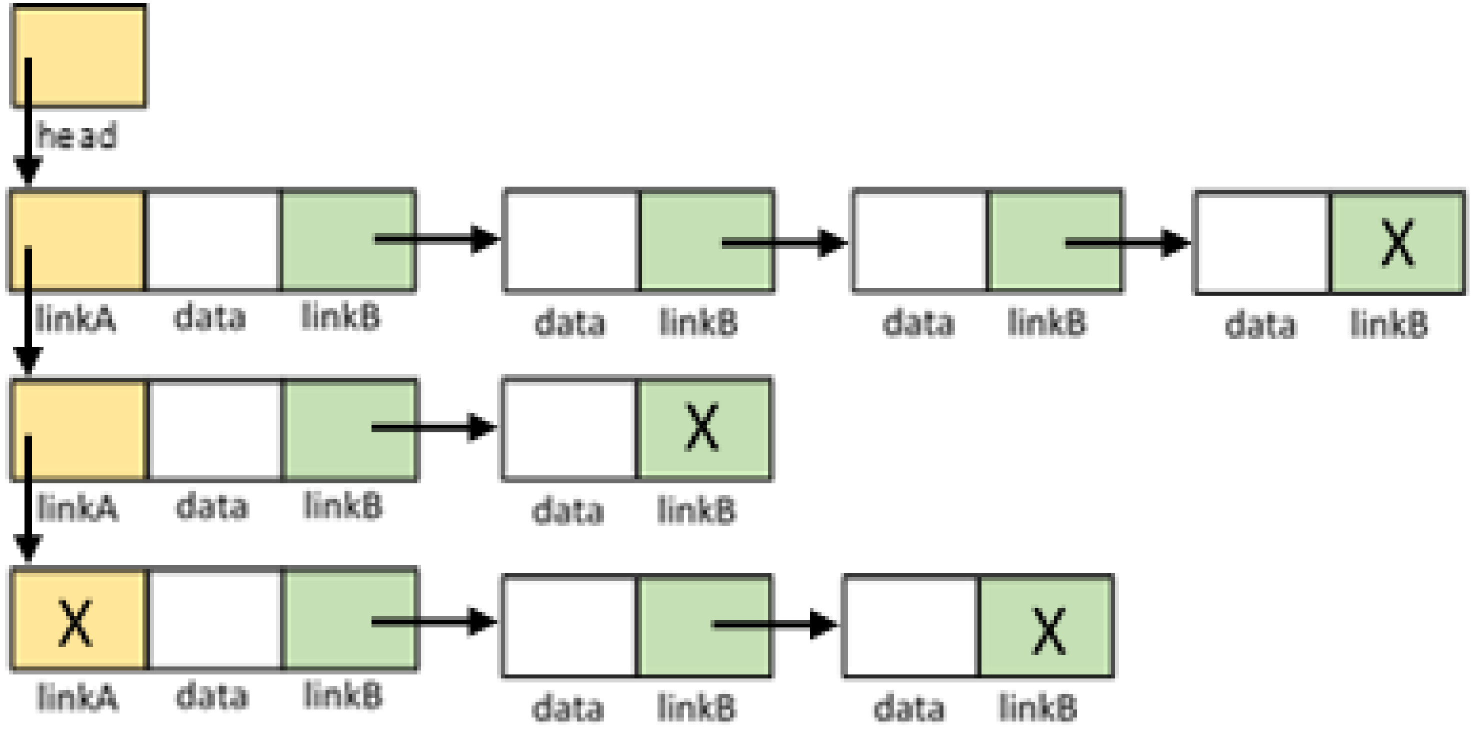
Circular Linked List Operation (cont.)

- Print circular linked list

```
public void print(){
    if(tail != null){
    Node Temp = tail.next;
    do{
        System.out.println(temp.data);
        temp = temp.next;
    }while (temp != tail.next);
    }
}
```

Compiled & edited by: Zahid Zainal

Multidimensional Linked List



Compiled & edited by: Zahid Zainal

Array or Linked List?

- Finally, should we always use linked lists?
- Decision depends on what operations will be used most frequently, and which factors (speed/memory) are more critical. Following are *some* hints:
  - Number of elements is known: use array
  - Dynamic addition and expansion: linked list
  - Deletion at any position: linked list
  - Need lots of random access: use array
  - Searching and items are unsorted: both same
  - Searching and items are sorted: array (binary search)
  - Sorting using bubble sort: both same
  - Sorting using other methods: depends on the method

Compiled & edited by: Zahid Zainal

Summary

- Improvement of `insertLast()` method.
- Variation of linked list
  - Doubly linked list
  - Circular linked list
  - Multidimensional linked list

Compiled & edited by: Zahid Zainal

Next Topic…

- Stack
  - Concept
  - Application
  - Implementation

References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.