

Data Structures - ENGR2715U

Tutorial 1

A series of horizontal lines in teal and light blue colors, with varying lengths and offsets, creating a modern, layered effect across the width of the slide.

General Information

- Textbook:
 - Data Structures and Algorithms in Java, by Michael T. Goodrich, Roberto Tamassia

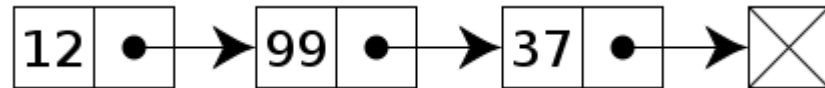
Linked Lists

- A data structure consisting of a collection of nodes which together represent a sequence
- One of the simplest and most common data structures, it is used as a basis for other structures such as stacks, queues, and symbolic expressions

Benefits and Tradeoffs

- The main counterpart to linked lists in terms of a data structure with similar functionality is the conventional Array.
- In a linked list, items can be inserted and removed without any explicit reallocation or reorganization of the entire structure.
- However, simple lists do not allow random access to the data, or indexing. So things like finding the last node, or finding a node with certain data may require more computing time.

Singly Linked Lists



- A singly linked list has nodes which contain a data field and a link to the next node in the sequence.

Basic singly-linked list operations:

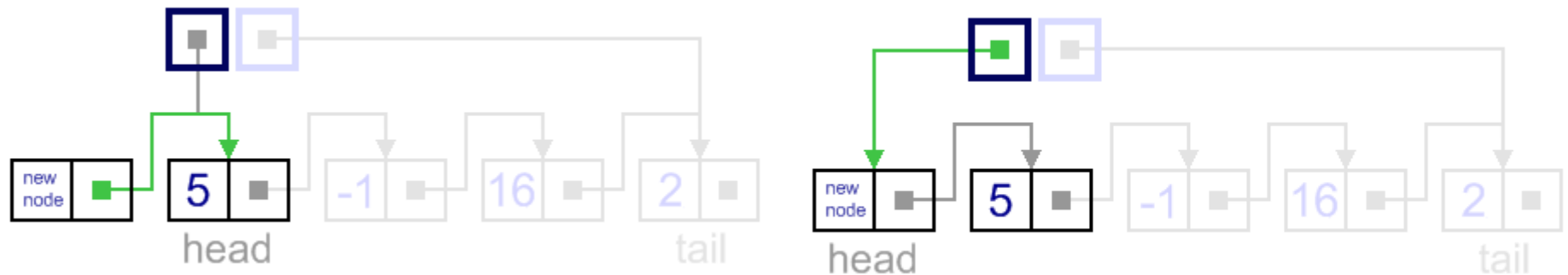
Traversal

- Traversal of a linked list is really the most basic operation involving lists.
- Algorithms may traverse a singly-linked list to find a value, find a location for insertion, etc.
- Basic algorithm:
 - Check if end hasn't been reached yet
 - Do required action with current node data
 - Current node becomes previous, and next becomes current. – Back to start

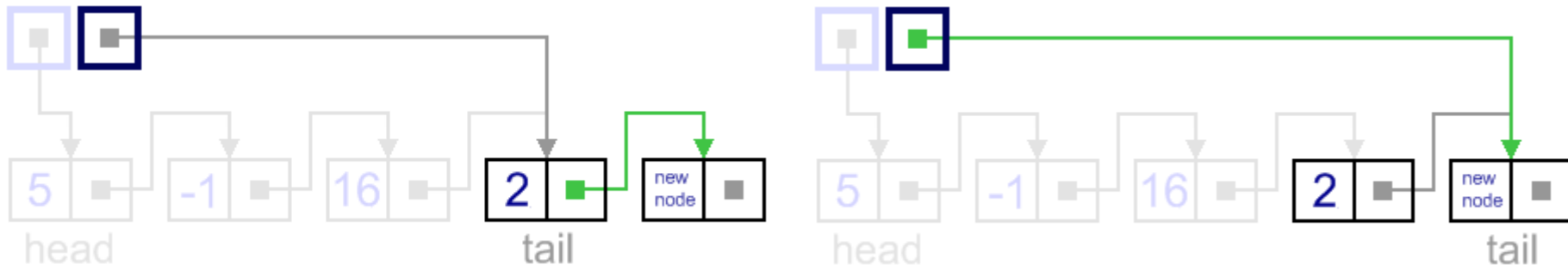
Basic singly-linked list operations:

Insertion of new node

- Adding first (before the head):



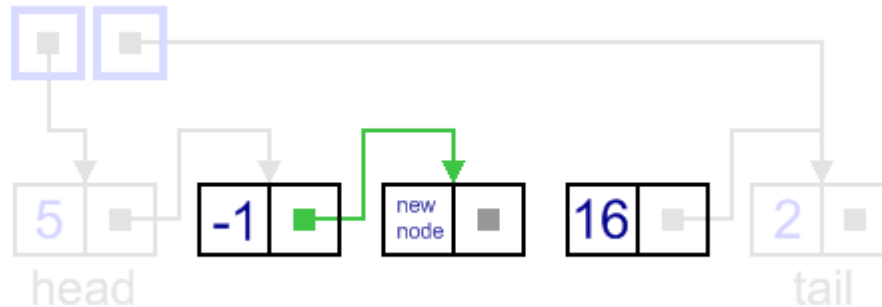
- Adding last (after the tail):



Basic singly-linked list operations:

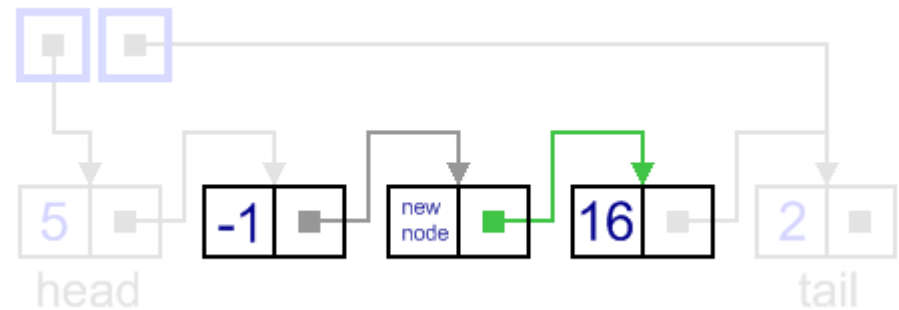
Insertion of new node (cont.d)

- General Case:



Update link of previous node to point to the new node

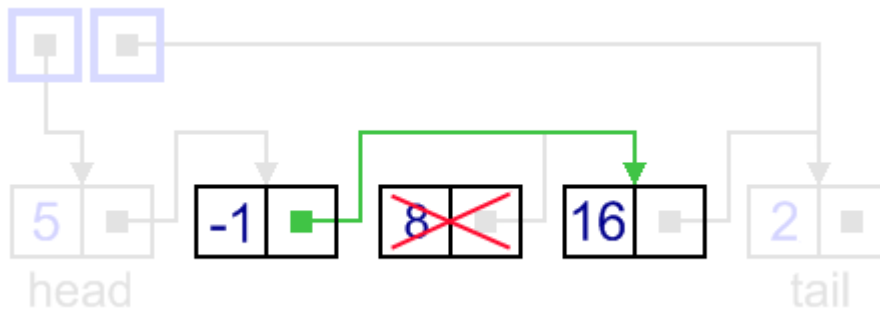
Update link of the new node to point to the next node



Basic singly-linked list operations:

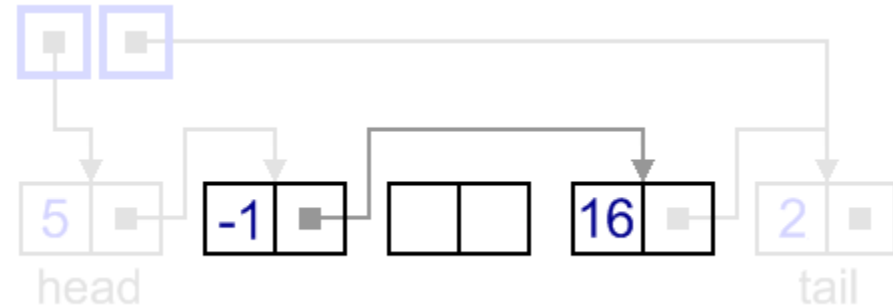
Removing a node

- General case:



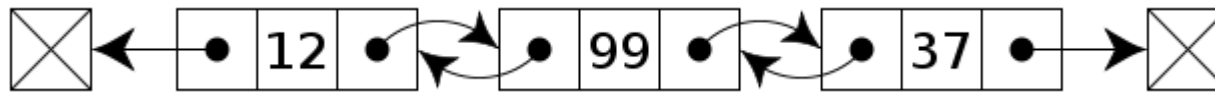
Update link of prev node
to next node

Dispose of unlinked node



Removing the first or last node are special cases, and removing the tail node will require traversal of the entire list to get the node that has the tail as its 'next' reference.

Doubly Linked List



- A doubly linked list is similar to a singly linked list, except that each contains in addition a link to its previous node in the sequence.
- While adding nodes requires changing more links, the operations become simpler because there is no need to keep track of the previous node during traversal or no need to traverse the list to find nodes to modify links.

Basic doubly-linked list operations:

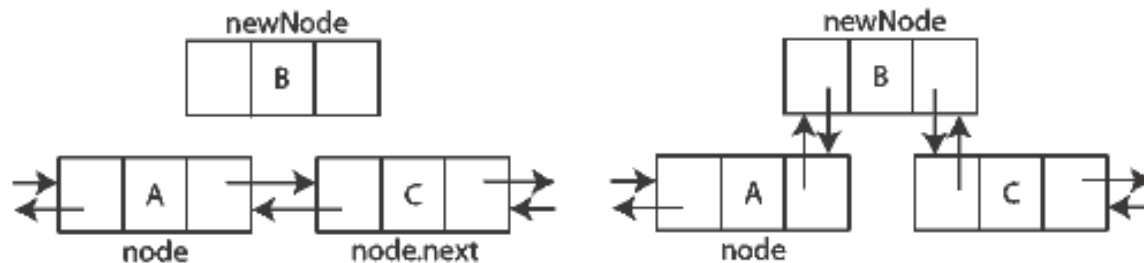
Traversal

- Main difference from a singly-linked list is that traversal can be in either direction.
- For backwards:
 - Check if beginning has been reached
 - Do required action with current node data
 - Current node becomes next, previous node becomes current. – Back to start

Basic doubly-linked list operations:

Insert a node

- General case (insert after node):



Basic algorithm:

- Set new node's prev link to argument node.
- Set new node's next link to argument node's next node.
- check if argument node is last in list
- If not, set argument node's next node's link to new node.

Basic doubly-linked list operations:

Remove a node

- Removing a node is much simpler in a doubly linked list.

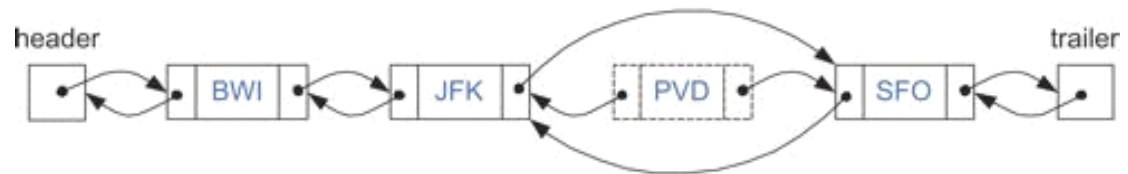
- Link node's prev node's 'next' link to node's next

- Link node's next node's 'prev' link to node's prev

- Dispose unlinked node




(a)



(b)



(c)



Some Java Code Implementation of these
two linked list types