COMP108
Data Structures and Algorithms

Data structures - Linked Lists (Part I)

Professor Prudence Wong

pwong@liverpool.ac.uk

2022-23

Outline

Linked lists

- What are linked lists?
- Traversing a linked list
- Searching over a unsorted/sorted list
- Inserting a node to a linked list
- Deleting a node from a linked list

Learning outcome:

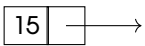
- Understand what **linked lists** are and their associated algorithms

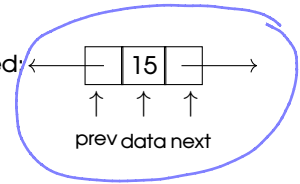
Linked lists

- elements are arranged in a linear order
- the order is determined by a pointer (rather than array indices)

Linked lists

- elements are arranged in a linear order
- the order is determined by a pointer (rather than array indices)
- each element (**node**) has a **data** field and one (or two) pointers linking to the **next** (or **previous**) element in the list

singly linked: 

doubly linked: 

node

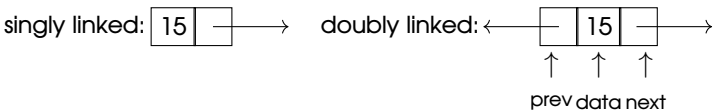
node.data

node.next

node.prev

Linked lists

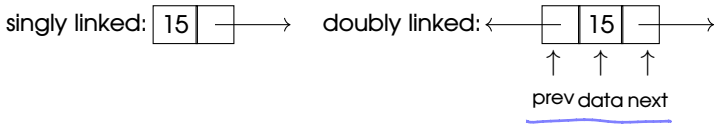
- elements are arranged in a linear order
- the order is determined by a pointer (rather than array indices)
- each element (**node**) has a **data** field and one (or two) pointers linking to the **next** (or **previous**) element in the list



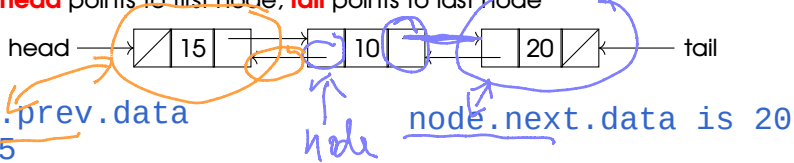
- refer as **node.data**, **node.next**, **node.prev**
  - if node is last element, then node.next is **NIL**
  - if node is first element, then node.prev is **NIL**

Linked lists

- elements are arranged in a linear order
- the order is determined by a pointer (rather than array indices)
- each element (**node**) has a **data** field and one (or two) pointers linking to the **next** (or **previous**) element in the list

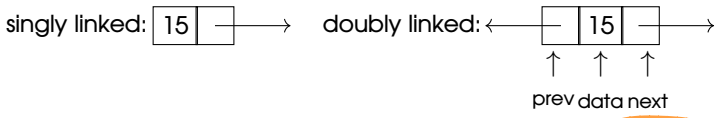


- refer as **node.data**, **node.next**, **node.prev**
  - if node is last element, then node.next is **NIL**
  - if node is first element, then node.prev is **NIL**
- **head** points to first node, **tail** points to last node

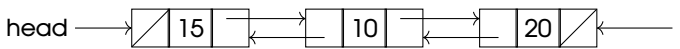


Linked lists

- elements are arranged in a linear order
- the order is determined by a pointer (rather than array indices)
- each element (**node**) has a **data** field and one (or two) pointers linking to the **next** (or **previous**) element in the list



- refer as **node.data**, **node.next**, **node.prev**
  - if node is last element, then node.next is **NIL**
  - if node is first element, then node.prev is **NIL**
- **head** points to first node, **tail** points to last node



- E.g., head.data is 15; head.next.data is 10; tail.data is 20, tail.prev.prev.data is 15

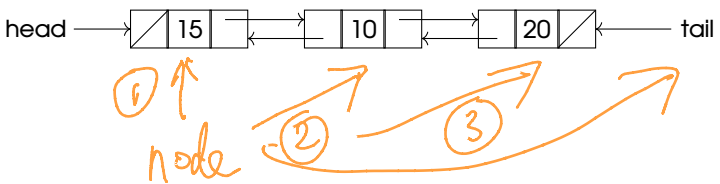
NIL .data

tail.next.data? error

head.next.prev.data is 15

Linked lists - Algorithm - traversing

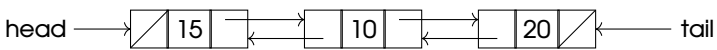
Traversing and output each element of a linked list



Linked lists - Algorithm - traversing

Traversing and output each element of a linked list

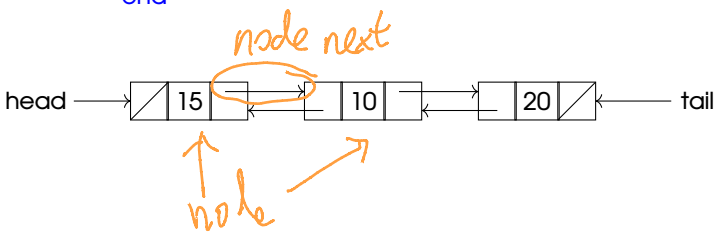
```
node ← head
while node ≠ NIL do
begin


end
```



Linked lists - Algorithm - traversing

Traversing and output each element of a linked list

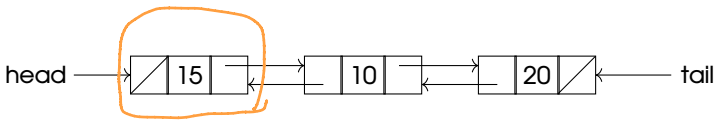
```
node ← head
while node ≠ NIL do
begin

    node ← node.next
end
```



Linked lists - Algorithm - traversing

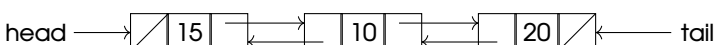
Traversing and output each element of a linked list

```
node ← head
while node ≠ NIL do
begin
    output node.data
    node ← node.next
end
```



Linked lists - Algorithm - searching

Searching if a value **key** is in a linked list

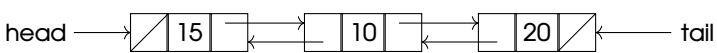


Linked lists - Algorithm - searching

Searching if a value **key** is in a linked list

```
node ← head, found ← false        flag variable found
while node ≠ NIL AND found == false do
begin




end
if found == true then
      output ``FOUND!''
else output ``NOT FOUND!''
```

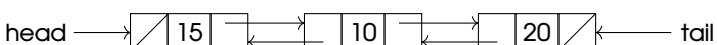
} expected output code



Linked lists - Algorithm - searching

Searching if a value **key** is in a linked list

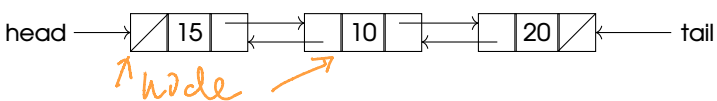
```
node ← head, found ← false
while node ≠ NIL AND found == false do
begin



        node ← node.next
end
if found == true then
    output ``FOUND!''
else output ``NOT FOUND!''
```



Linked lists - Algorithm - searching

Searching if a value **key** is in a linked list

```
node ← head, found ← false
while node ≠ NIL AND found == false do
begin
    if node.data == key then
        found ← true
    else
        node ← node.next
end
if found == true then
    output ``FOUND!''
else output ``NOT FOUND!''
```



Linked lists - Algorithm - searching

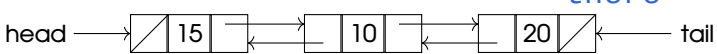
Searching if a value **key** is in a linked list

```
node ← head, found ← false
while node ≠ NIL AND found == false do
begin
    if node.data == key then
        found ← true
    else
        node ← node.next
end
if found == true then
    output ``FOUND!''
else output ``NOT FOUND!''
```

Alternatively,

```
node ← head
while node ≠ NIL AND node.data ≠ key do
    node ← node.next
if node == NIL then
    output ``NOT FOUND!''
else output ``FOUND!''
```

`if node.data == key??`
`ANSWER: no, because there may be error (nil pointer exception) when key is not there`



Linked lists - Algorithm - searching

Searching if a value **key** is in a linked list

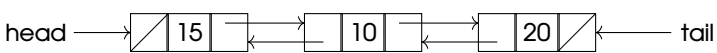
```
node ← head, found ← false
while node ≠ NIL AND found == false do
begin
    if node.data == key then
        found ← true
    else
        node ← node.next
end
if found == true then
    output ``FOUND!''
else output ``NOT FOUND!''
```

n elenments
best case: O(1)
worst case: O(n)

Alternatively,

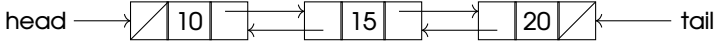
```
node ← head
while node ≠ NIL AND node.data ≠ key do
    node ← node.next
if node == NIL then
    output ``NOT FOUND!''
else output ``FOUND!''
```

Time complexity?



Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?



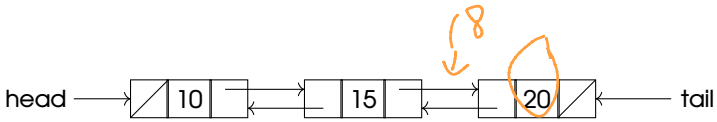
Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?

```
node ← head
while node ≠ NIL AND node.data < key do
    node ← node.next
```

- Under what conditions would the while loop stop?

>=

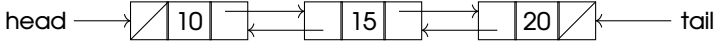


Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?

node $\leftarrow$ head
while node $\neq$ NIL AND **node.data** $<$ **key** do
    node $\leftarrow$ node.next

- Under what conditions would the while loop stop?
  1. **node is NIL:** whole list is smaller than key

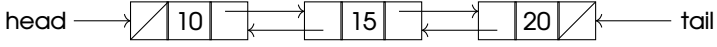


Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?

```
node ← head
while node ≠ NIL AND node.data < key do
    node ← node.next
```

- Under what conditions would the while loop stop?
  1. **node is NIL:** whole list is smaller than key
  2. **node.data is > key:** hit a larger number without seeing key

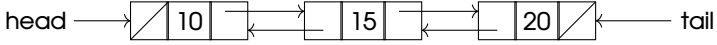


Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?

```
node ← head
while node ≠ NIL AND node.data < key do
    node ← node.next
```

- Under what conditions would the while loop stop?
  1. **node is NIL:** whole list is smaller than key
  2. **node.data is > key:** hit a larger number without seeing key
  3. **node.data is key:** FOUND!

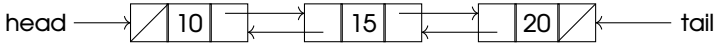


Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?

```
node ← head
while node ≠ NIL AND node.data < key do
    node ← node.next
if node == NIL then
    output ``NOT FOUND!''
else if node.data == key then
        output ``FOUND!''
    else output ``NOT FOUND!''
```

- Under what conditions would the while loop stop?
  1. **node is NIL:** whole list is smaller than key
  2. **node.data is > key:** hit a larger number without seeing key
  3. **node.data is key:** FOUND!

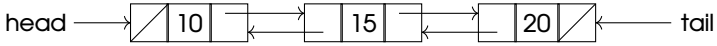


Linked list - Searching over a sorted list

- Recall that if the values in an array are sorted, say ascending, we can use binary search to search efficiently.
- However, even if the values in a linked list are sorted, we still can't use binary search. Why?
- We can modify the sequential search algorithm and may terminate the loop earlier. How?

```
node ← head
while node ≠ NIL AND node.data < key do
      node ← node.next
if node == NIL then
      output ``NOT FOUND!''
else if node.data == key then
            output ``FOUND!''
      else output ``NOT FOUND!''
```

- Under what conditions would the while loop stop?
  1. **node is NIL:** whole list is smaller than key
  2. **node.data is > key:** hit a larger number without seeing key
  3. **node.data is key:** FOUND!
- Why need to check if node == NIL and not simply if node.data == key?



Time complexity

Are the following statements correct about an array / a linked list with n elements?

The time complexity of searching over an array is $O(n)$. ✓

The time complexity of searching over a sorted array is $O(\log n)$. ✓

The time complexity of searching over a linked list is $O(n)$. ✓

The time complexity of searching over a sorted linked list is $O(\log n)$. ✗

$O(n)$

Summary: Linked lists - Traversing & Searching

Next: Linked lists - Insertion & Deletion

For note taking